Πανεπιστήμιο Πειραιώς – Τμήμα Πληροφορικής
Πρόγραμμα Μεταπτυχιακών Σπουδών
«Προηγμένα Συστήματα Πληροφορικής»

Μεταπτυχιακή Διατριβή

Τίτλος Διατριβής	**Εύρεση ευπαθειών σε μεταγλωττισμένες βιβλιοθήκες λογισμικού** **Finding vulnerabilities in compiled software libraries**
Ονοματεπώνυμο Φοιτητή	**Ευάγγελος Γρηγορίου**
Πατρώνυμο	**Γρηγόρης**
Αριθμός Μητρώου	**ΜΠΣΠ/ 14018**
Επιβλέπων	**Παναγιώτης Κοτζανικολάου, Επίκουρος Καθηγητής**
Συνεπιβλέπων	**Δρ. Δημήτρης Γλυνός**

Ημερομηνία Παράδοσης **Σεπτέμβριος 2017**

Τριμελής Εξεταστική Επιτροπή

(υπογραφή)	(υπογραφή)	(υπογραφή)
Παναγιώτης Κοτζανικολάου Επίκουρος Καθηγητής	Χρήστος Δουληγέρης Καθηγητής	Μιχαήλ Ψαράκης Επίκουρος Καθηγητής

Περίληψη

Οι ευπάθειες ασφαλείας ενός λογισμικού μπορούν να προκαλέσουν την δυσλειτουργία μιας εφαρμογής, εκτέλεση μη εγκεκριμένου κώδικα, μη εγκεκριμένη πρόσβαση σε κακόβουλο λογισμικό ή άλλο χρήστη και άλλες κακόβουλες ενέργειες για τις οποίες ο χρήστης μιας συσκευής δεν έχει εγκρίνει. Αυτά με την σειρά τους μπορούν να προκαλέσουν διαρροή ευαίσθητων πληροφοριών, όπως στοιχεία τραπεζικού λογαριασμού, μη εγκεκριμένο έλεγχο της συσκευής και κατά συνέπεια την εκτέλεση μη εγκεκριμένων ενεργειών, όπου εμφανίζονται ως ενέργειες του χρήστη. Στο πλαίσιο της παρούσας διπλωματικής εξετάστηκαν μεθόδοι για τον εντοπισμό ευπαθειών σε μεταγλωττισμένες βιβλιοθήκες και αναπτύχθηκε σχετική εφαρμογή. Η εφαρμογή σχεδιάστηκε να ελέγχει απλές περιπτώσεις ευπαθειών ασφαλείας που οφείλονται στη χρήση βασικών συναρτήσεων της βιβλιοθήκης Libc. Αποτελεί εργαλείο στατικής ανάλυσης δυαδικών αρχείων, όπου αναλύει και ελέγχει μεταγλωττισμένο κώδικα αρχιτεκτονικών x86, x86_64, arm32 και arm64. Η πειραματική αξιολόγηση του λογισμικού έγινε με πραγματικές βιβλιοθήκες συστημάτων Ubuntu και Android, όπου με την χρήση του πηγαίου κώδικα, όπου αυτός ήταν διαθέσιμος, ελέγχθηκε η ορθότητα των αποτελεσμάτων και διαπιστώθηκε ότι αποτελεί ένα χρήσιμο εργαλείο κατά τη διάρκεια ανάπτυξης ή διερεύνησης λογισμικού.

Abstract

Security vulnerabilities of a software can cause an application to malfunction, execute an unauthorized code, unauthorized access to malicious software or other user and other malicious actions for which the user of a device has not approved. These in turn can cause leakage of sensitive information, such as bank account information, unauthorized control of the device, and therefore the execution of unauthorized actions where they appear as user's actions. In the context of this dissertation, methods will investigate for the identification of vulnerabilities in compiled software libraries and an application was implemented. The application was designed to check simple cases of security vulnerabilities due to the use of Libc's core functions. It is a static analysis tool that analyzes and checks compiled code of x86, x86_64, arm32, and arm64 architectures. The experimental evaluation of the software was performed with real Ubuntu and Android libraries, using the available source code to confirm the correctness of the results and was found to be a useful tool during software development or exploration.

Περιεχόμενα

1 Εισαγωγή

Η ραγδαία ανάπτυξη του υλικού των περισσοτέρων συσκευών που χρησιμοποιούνται στην καθημερινότητά, καταστεί αναγκαία όλο και περισσότερο την χρήση λογισμικού για την λειτουργία τους, καθώς επίσης και την ανάπτυξη εφαρμογών για την βέλτιστη χρήση των δυνατοτήτων του. Οι απαιτήσεις μεταξύ των συσκευών αυτών, όπως το μέγεθος της συσκευής, η απόδοση της, η κατανάλωση ηλεκτρικής και υπολογιστικής ισχύος, ποικίλουν αναλόγως με την χρήση για την οποία προορίζονται. Αυτό με τη σειρά του οδηγεί στην δημιουργία συσκευών με διαφορετικές, ανά περίπτωση, αρχιτεκτονικές υλικού και κατά συνέπεια στην δημιουργία νέων, ανά αρχιτεκτονική υλικού, λογισμικών, τα οποία παρέχουν τόσο την διεπαφή μεταξύ υλικού-λογισμικού, όσο και έτοιμες ενέργειες, με στόχο την απλούστευση της ανάπτυξης εφαρμογών για το υλικό της εκάστοτε συσκευής.

Οι σταθεροί και φορητοί υπολογιστές στις μέρες μας χρησιμοποιούν την αρχιτεκτονική x86-64, μία επέκταση του συνόλου εντολών x86, όπου αρχικά δημιουργήθηκε αρχικά από την AMD και πλέον υλοποιείται από την ίδια την AMD, καθώς επίσης και από την Intel.

Η αρχιτεκτονική ARM είναι αυτή που επικρατεί στην πλειοψηφία συσκευών καθημερινής, και όχι μόνο, χρήσης, όπως smartphones, tablets κλπ. Η ARM (Advanced RISC Machine) είναι μια αρχιτεκτονική συνόλου εντολών RISC (Reduced Instruction Set Computing) των 32-bit, η οποία έχει αναπτυχθεί από την ARM Holdings.

Η κάθε μία από τις δύο αυτές αρχιτεκτονικές υλικού, υποστηρίζει καταχωρητές μεγέθους 32-bit και 64bit και χρησιμοποιεί την δική της γλώσσα χαμηλού επιπέδου (assembly). Λόγω των διαφορών αυτών μεταξύ τους, για να εκτελεστεί ένα αρχείο πηγαίου κώδικα, θα πρέπει να μεταγλωττιστεί στην εκάστοτε αρχιτεκτονική, δημιουργώντας έτσι διαφορετικά εκτελέσιμα αρχεία ανά αρχιτεκτονική. Η απεικόνιση ενός εκτελέσιμου αρχείου γίνεται με την χρήση των εντολών της γλώσσας χαμηλού επιπέδου της αρχιτεκτονικής για την οποία έχει μεταγλωττιστεί το αρχείο αυτό. Οπότε, για κάθε αρχείο πηγαίου κώδικα, προκύπτουν διαφορετικά εκτελέσιμα αρχεία, που απεικονίζονται με διαφορετικές γλώσσες μηχανής. Αυτό δεν επηρεάζει την ανάπτυξη πηγαίου κώδικα, αλλά προκαλεί δυσκολία στον έλεγχο εκτελέσιμων αρχείων.

Μια τέτοια περίπτωση είναι ο έλεγχος ευπαθειών ασφαλείας σε αρχεία βιβλιοθηκών. Στις περιπτώσεις που η βιβλιοθήκη είναι έργο ανοικτού λογισμικού ο πηγαίος κώδικας είναι διαθέσιμος. Βιβλιοθήκες, όμως, που είναι για συγκεκριμένα hardware components ενός κατασκευαστή (π.χ. η βιβλιοθήκη που διαχειρίζεται το fingerprint στα android κινητά) ή που περιλαμβάνουν IP (intellectual property) των κατασκευαστών, συνήθως παρέχονται μόνο σε μορφή binary. Οι βιβλιοθήκες αυτές μπορεί να χρησιμοποιούνται από πολλές διαφορετικές εφαρμογές, με αποτέλεσμα τα όποια προβλήματα ασφάλειας των βιβλιοθηκών αυτών να προσβάλλουν όλες ή μερικές από αυτές τις εφαρμογές. Οι εγκαταστάσεις αυτών μπορεί να είναι μεγάλες (π.χ. βρίσκονται εγκατεστημένες σε όλες τις συσκευές ενός συγκεκριμένου κατασκευαστή), διευκολύνοντας έτσι κακόβουλους χρήστες να πλήξουν μαζικά χρήστες των συσκευών αυτών, αναζητώντας αδυναμίες σε τέτοιες βιβλιοθήκες. Η ανάγκη χρήσης βιβλιοθηκών για την ανάπτυξη εφαρμογών, καθιστά αναγκαίο τον έλεγχο ασφαλείας. Δεδομένου, επίσης, της μεγάλης διείσδυσης embedded συσκευών (internet of things, CPE, κινητών συσκευών κ.α.) που χρησιμοποιούν τέτοιου τύπου βιβλιοθήκες κλειστού λογισμικού, κάθε προσπάθεια αυτόματου εντοπισμού για αδυναμίες σε αυτό το λογισμικό είναι μεγάλης σημασίας. Το πρόβλημα είναι ότι τα αρχεία της βιβλιοθήκης είναι δυαδικά αρχεία, μεταγλωττισμένα στην εκάστοτε γλώσσα χαμηλού επιπέδου, πράγμα που καθιστά δύσκολο, τόσο τον έλεγχο, όσο και την ανάπτυξη ενός εργαλείου ελέγχου ευπαθειών.

Παρ' όλα αυτά, κάποιες από τις γνωστές αυτές ευπάθειες, είναι απλές στον έλεγχο τους και μπορούν να κατηγοριοποιηθούν. Κάποιες από αυτές προκύπτουν από λανθασμένη χρήση συναρτήσεων βασικών βιβλιοθηκών. Η έλλειψη ελέγχων των τιμών που επιστρέφουν οι συναρτήσεις αυτές και έλλειψη ελέγχου εξωτερικών ορισμάτων, είτε αριθμητικών είτε αλφαριθμητικών, που χρησιμοποιούνται κατά την κλήση των συναρτήσεων αυτών, είναι δύο από τις βασικές περιπτώσεις που μπορούν να οδηγήσουν σε ευπάθειες ασφαλείας. Τέτοιες ευπάθειες ασφαλείας είναι buffer overflow, format string bugs, code injection, κλπ. Ο έλεγχος τους είναι σχετικά εύκολος, όταν πρόκειται για γλώσσα υψηλού επιπέδου. Σε γλώσσα χαμηλού επιπέδου όμως, η δυσκολία αυξάνεται, μιας και η υλοποίηση μιας εντολής γλώσσας

υψηλού επιπέδου, αντιστοιχεί σε πολλές εντολές χαμηλού επιπέδου. Επίσης, οι εντολές αυτές, μορφοποιούνται αναλόγως την αρχιτεκτονική. Οπότε ο έλεγχος ασφαλείας σε δυαδικά αρχεία καθίσταται σχεδόν αδύνατος, γιατί πέρα του γεγονότος ότι θα πρέπει κάποιος να γνωρίζει αρκετά καλά την αντίστοιχη γλώσσα χαμηλού επιπέδου, το πλήθος των εντολών της γλώσσας χαμηλού επιπέδου περιπλέκει πολύ την παρακολούθηση της ροής του κώδικα, ακόμα και σε απλές περιπτώσεις.

Δύο βασικές κατηγορίες μεθόδων για την εξέταση ευπαθειών λογισμικού είναι η στατική ανάλυση στον πηγαίο κώδικα ή στο εκτελέσιμο μιας εφαρμογής και η δυναμική ανάλυση μιας εφαρμογής (κατά την εκτέλεσή της, π.χ. με μεθόδους όπως το fuzzing), που δρουν πολλές φορές συμπληρωματικά. Επειδή ακριβώς κάποιες αδυναμίες είναι δύσκολο να εντοπιστούν κατά την ανάλυση του πηγαίου κώδικα, μπορούν πολλές φορές να εντοπιστούν πιο γρήγορα στη δυαδική αναπαράσταση, ενώ άλλες μπορούν πιο εύκολα να εντοπιστούν με δυναμικές προσεγγίσεις. Πάντως σε κάθε περίπτωση όταν ο πηγαίος κώδικας δεν είναι διαθέσιμος οι δυνατότητες εξεύρεσης αδυναμιών ασφάλειας είναι πιο περιορισμένες.

Ένας τρόπος υλοποίησης ελέγχου ευπαθειών ασφαλείας, θα ήταν η ανάπτυξη ενός εργαλείου που θα ελέγχει για απλές, βασικές ευπάθειες, που υπό συνθήκες, μπορούν να δημιουργήσουν σοβαρά προβλήματα ασφαλείας στην εφαρμογή που χρησιμοποιεί την βιβλιοθήκη αυτή. Η κατηγοριοποίηση των βασικών αυτών ευπαθειών, διευκολύνει την ανάπτυξη σεναρίων για τον έλεγχό τους, όμως οι διαφορετικές γλώσσες χαμηλού επιπέδου εξακολουθούν να δυσκολεύουν τον αυτοματοποιημένο έλεγχο. Κάθε σενάριο εκφράζεται διαφορετικά στην εκάστοτε γλώσσα χαμηλού επιπέδου, πράγμα που σημαίνει ότι, για κάθε αρχιτεκτονική που θα εξετάζει το εργαλείο, θα πρέπει να δημιουργηθούν διαφορετικές υλοποιήσεις σεναρίων στην εκάστοτε αρχιτεκτονική.

Η λύση του προβλήματος των διαφορετικών γλωσσών χαμηλού επιπέδου, θα ήταν η χρήση μιας γλώσσας ενδιάμεσης αναπαράστασης (Intermediate Representation), η οποία θα μεταφράζει τις διαφορετικές γλώσσες χαμηλού επιπέδου, σε μία κοινή γλώσσα. Έτσι, εξαλείφεται το πρόβλημα των διαφορετικών, ανά αρχιτεκτονική, εντολών.

Το πρόβλημα που παραμένει, είναι η διαφοροποίηση που ενδέχεται να υπάρχει μεταξύ των αρχιτεκτονικών, όσον αναφορά τον τρόπο που διαχειρίζονται ορισμένες ενέργειες, όπως για παράδειγμα το πέρασμα ορισμάτων σε μία εξωτερική συνάρτηση που θα κληθεί από το πρόγραμμα. Σε απλές περιπτώσεις είναι εφικτή η ξεχωριστή υλοποίηση για την κάθε αρχιτεκτονική.

Η υλοποίηση ενός εργαλείου ελέγχου βασικών γνωστών ευπαθειών σε βιβλιοθήκες ποικίλων αρχιτεκτονικών, θα ήταν χρήσιμο στην ανάπτυξη εφαρμογών που θα χρησιμοποιήσουν τις βιβλιοθήκες αυτές, γνωρίζοντας αν οι βιβλιοθήκες είναι ευάλωτες σε ευπάθειες ασφαλείας, αποφεύγοντας έτσι την δημιουργία ευπαθειών στην εφαρμογή. Για την ανάπτυξη του εργαλείου αυτού, είναι απαραίτητη η κατηγοριοποίηση των ευπαθειών και η δημιουργία σεναρίων, η χρήση μιας γλώσσας ενδιάμεσης αναπαράστασης και ξεχωριστές υλοποιήσεις απλών ενεργειών ανά αρχιτεκτονική.

Ένα τέτοιο εργαλείο αναπτύχθηκε στο πλαίσιο της παρούσας διπλωματικής εργασίας και ονομάστηκε "carrion". Τα κεφάλαια που ακολουθούν περιγράφουν την έρευνα που διεξήχθει στις τεχνικές ανάλυσης ενός αρχείου στα πλαίσια της εξέτασης ευπαθειών λογισμικού, τα διαφορετικά στάδια που προέκυψαν κατά τον σχεδιασμό του εργαλείου και τον τρόπο που αυτά υλοποιήθηκαν, οδηγίες χρήσης του εργαλείου και ανάλυση των παραμέτρων που δέχεται, αποτελέσματα πειραματικής αξιολόγησης και ανάλυση αυτών, μελλοντικές προσθήκες, διορθώσεις και αναβαθμίσεις του εργαλείου και τέλος τα συμπεράσματα αυτής της διπλωματικής εργασίας.

2 Έρευνα Πεδίου

2.1 Εισαγωγή

Ανάλυση προγράμματος ονομάζεται η διαδικασία με την οποία αναλύεται αυτόματα η συμπεριφορά προγραμμάτων που εκτελούνται σε έναν υπολογιστή. Κατά την διαδικασία της ανάλυσης προγράμματος παράγονται μοντέλα του προγράμματος προς ανάλυση, τα οποία χρησιμοποιούνται σε τεχνικές εξομοίωσης, με στόχο την μέτρηση και τον έλεγχο της ροής του προγράμματος.

Η ανάλυση προγράμματος χρησιμοποιείται επίσης, για τον έλεγχο ευπαθειών ασφαλείας. Οι περισσότερες ευπάθειες ασφαλείας είναι δυνατό να εντοπιστούν κατά την ανάλυση προγράμματος, σχετικά εύκολα και γρήγορα, προλαμβάνοντας τα όποια προβλήματα αυτές θα δημιουργούσαν, καθιστώντας την αναγκαίο στάδιο για την ανάπτυξη ενός λογισμικού.

Αναλόγως με το είδος της ανάλυσης, προκύπτουν διαφορετικές τεχνικές και κατά συνέπεια, διαφορετικοί ελέγχοι ευπαθειών, καλύπτοντας έτσι ένα μεγάλο εύρος ευπαθειών αυτών. Υπάρχουν δύο βασικές προσεγγίσεις στην ανάλυση προγράμματος, η στατική ανάλυση και η δυναμική ανάλυση.

2.2 Στατική Ανάλυση

Στατική ανάλυση, ονομάζεται η διαδικασία κατά την οποία αναλύεται ο πηγαίος κώδικας ενός προγράμματος, χωρίς αυτό να εκτελεστεί [1] [2]. Συνήθως, πραγματοποιείται στα πλαίσια ελέγχου ασφαλείας (τεχνική white-box) και υλοποιείται από κάποιο αυτοματοποιημένο εργαλείο στατικής ανάλυσης πηγαίου κώδικα, που σκοπό έχει να αναδείξει πιθανές ευπάθειες ασφαλείας σε σημεία του κώδικα, χωρίς την εκτέλεση του προγράμματος.

Η μέτρηση μιας οποιασδήποτε ιδιότητας ενός προγράμματος, όπως και ο αντίστροφος σχεδιασμός (reverse engineering) θεωρούνται επίσης είδη στατικής ανάλυσης, μιας και το πρόγραμμα δεν εκτελείται.

Η ανάλυση που μπορεί να γίνει από αυτόματα εργαλεία ποικίλλει από αυτά που εξετάζουν τη συμπεριφορά μόνο συγκεκριμένων εκφράσεων ή δηλώσεων στο πρόγραμμα, έως αυτά που συμπεριλαμβάνουν ολόκληρο τον πηγαίο κώδικα ενός προγράμματος στην ανάλυση. Σε μία ιδανική περίπτωση, τα εργαλεία αυτά βρίσκουν αυτόματα ευπάθειες ασφαλείας με υψηλό βαθμό εμπιστοσύνης. Ωστόσο, κάτι τέτοιο δεν είναι πάντα εφικτό, λόγω της φύσεως των ευπαθειών ασφαλείας. Έτσι, τέτοια εργαλεία συχνά χρησιμεύουν ως βοηθήματα αναλυτών, για την υπόδειξη σημείων όπου μπορεί να προκύψει κάποιο πρόβλημα ασφαλείας.

Μερικά εργαλεία αρχίζουν να κινούνται στο Περιβάλλον Ολοκληρωμένης Ανάπτυξης (Integrated Development Environment - IDE). Παρέχουν άμεση παροχή πληροφοριών στην φάση της υλοποίησης σχετικά με κάποιες ευπάθειες ασφαλείας που μπορούν να ανιχνευθούν εκείνη την στιγμή. Η εύρεση και διόρθωση ευπαθειών κατά την υλοποίηση ενός προγράμματος είναι πολύ πιο αποδοτική, συγκριτικά με την εύρεση και διόρθωση των ευπαθειών αυτών σε μετέπειτα φάσεις.

Υπάρχουν διάφορες τεχνικές στατικής ανάλυσης, οι οποίες συχνά συνδυάζονται για την εύρεση ευπαθειών. Οι κυριότερες είναι οι εξής:

- Data Flow Analysis
- Control Flow Graph (CFG)
- Taint Analysis
- Lexical Analysis

Υπάρχουν επίσης, πολλά και διάφορα εργαλεία που υλοποιούν την διαδικασία της στατικής ανάλυσης, καλύπτοντας διαφορετικές περιπτώσεις, γλώσσες και είδη εφαρμογών. Τα κυριότερα εργαλεία ανοιχτού κώδικα είναι [3] [4]:

- Google CodeSearchDiggity (Multiple)
- PMD (Java)
- FlawFinder (C/C++)

- Microsoft FxCop (.NET)
- Splint (C)
- FindBugs (Java)
- RIPS (PHP)
- Agnitio (Objective-C, C#, Java & Android)
- Microsoft PreFast (C/C++)
- Fortify RATS (C, C++, Perl, PHP & Python)
- DevBug (PHP)
- Brakeman (Rails)
- VisualCodeGrepper (C/C++, C#, VB, PHP, Java & PL/SQL)

Ενώ αντίστοιχα εμπορικά εργαλεία είναι τα εξής:

- Fortify (OWASP Member)
- Veracode (OWASP Member)
- GrammaTech
- ParaSoft
- Armorize CodeSecure (OWASP Member)
- Checkmarx Static Code Analysis (OWASP Member)
- Rational AppScan Source Edition
- Coverity
- PVS-Studio
- Insight
- Polyspace Static Analysis

2.3 Δυναμική Ανάλυση

Δυναμική ανάλυση προγράμματος, είναι ανάλυση που γίνεται με την εκτέλεση των προγραμμάτων σε έναν πραγματικό ή εικονικό περιβάλλον [5]. Για να είναι αποδοτική, πρέπει το πρόγραμμα προς ανάλυση να εκτελεστεί με αρκετές εισόδους ελέγχου για να εμφανιστεί ενδιαφέρουσα συμπεριφορά.

Κατά την δυναμική ανάλυση, μπορούν να παρατηρηθούν λάθη που προκαλούνται κατά την εκτέλεση του προγράμματος (run-time errors), δυναμικές εξαρτήσεις του προγράμματος, τιμές μεταβλητών, δυναμική χρήση μνήμης και η πολυμορφία του προγράμματος.

Ένα από τα βασικά προβλήματα της δυναμικής ανάλυσης είναι ότι δεν εγγυάται την πλήρη κάλυψη του κώδικα ενός προγράμματος, μιας και υπάρχει το ενδεχόμενο η ροή του προγράμματος να εξαρτάται από τον χρήστη. Άλλο πρόβλημα που προκύπτει, είναι η πολυπλοκότητα της ανάλυσης, η οποία δημιουργεί δυσκολίες στην αυτοματοποίηση της.

Η πιο γνωστή οικογένεια εργαλείων δυναμικής ανάλυσης είναι τα προγράμματα εντοπισμού σφαλμάτων (debuggers). Άλλα χαρακτηριστικά εργαλεία δυναμικής ανάλυσης με διαφορετικά πεδία έρευνας είναι τα εξής [3] [4]:

- BoundsChecker
- Cenzic
- ClearSQL
- CodeDynamics
- Daikon (system)
- Dmalloc
- DynInst
- Gcov
- HP Security Suite
- IBM Rational AppScan
- Intel Thread Checker
- Intel Parallel Inspector

- Jalangi
- OpenPAT
- Parasoft Insure++
- Parasoft Jtest
- Prism from CriticalBlue
- Purify
- Valgrind
- VB Watch
- Vector Fabrics Pareon Verify

2.4 Συμβολική εκτέλεση

Συμβολική εκτέλεση (symbolic execution) είναι μία τεχνική κατά την οποία αναλύεται ένα πρόγραμμα με σκοπό τον καθορισμό των παραμέτρων που προκαλούν την εκτέλεση των εκάστοτε μερών του προγράμματος. Η ανάλυση στην συμβολική εκτέλεση γίνεται με βάση τις διαφορετικές ροές του προγράμματος, αντίθετα με την δυναμική ανάλυση, όπου η ανάλυση γίνεται με βάση τις παραμέτρους εισόδου του προγράμματος. Ένας διερμηνέας ακολουθεί το πρόγραμμα, θεωρώντας συμβολικές τιμές για παραμέτρους [6] [7].

Το πλεονέκτημα της συμβολικής εκτέλεσης, είναι το γεγονός ότι ως είσοδο δέχεται σύμβολα, και όχι αλφαριθμητικές τιμές, γενικοποιόντας έτσι τον έλεγχο ενός προγράμματος. Αν η ροή του προγράμματος εξαρτάται από την τιμή του συμβόλου, τότε ελέγχονται όλες οι πιθανές εκδοχές και εξετάζοντας όλα τα πιθανά μονοπάτια. Κατά συνέπεια προκύπτει η κατάσταση στην οποία θα πρέπει να βρίσκεται το σύμβολο, έτσι ώστε το πρόγραμμα να ακολουθήσει το συγκεκριμένο μονοπάτι ροής. Σε αντίθεση με την δυναμική ανάλυση, η συμβολική εκτέλεση καλύπτει όλες τις πιθανές εκδοχές ροής του προγράμματος, υπερτερώντας ως προς το πλήθος των επιμέρους διαδρομών που εξετάζει.

Προβλήματα που μπορούν να προκύψουν κατά την συμβολική εκτέλεση, είναι η αύξηση των διαδρομών του προγράμματος, η αποδοτικότητα της σε σχέση με την δυναμική ανάλυση και οι αλληλεπιδράσεις με εξωτερικές συναρτήσεις, όπως κλήσεις συστήματος ή άλλων βιβλιοθηκών. Όσο ένα πρόγραμμα μεγαλώνει, η αύξηση των διαδρομών του είναι εκθετική, με περιπτώσεις να τείνουν στο άπειρο, όταν υπάρχουν ατέρμονοι βρόγχοι. Η απόδοση της συμβολικής εκτέλεσης επηρεάζεται επίσης από το πόσες παράμετροι χρησιμοποιούν την ίδια διαδρομή στο πρόγραμμα. Αντίθετα με την δυναμική ανάλυση η οποία εξετάζει το πρόγραμμα με βάση τις παραμέτρους που λαμβάνει ως εισόδους, η συμβολική εκτέλεση εξετάζει με βάση τις διαδρομές που παράγει το πρόγραμμα. Έτσι, αν για κάθε διαδρομή έχουμε λίγες παραμέτρους που την προκαλούν, η απόδοσή της μειώνεται σημαντικά. Τέλος, η χρήση εξωτερικών παραγόντων στους οποίους ένα εργαλείο συμβολικής εκτέλεσης δεν έχει πρόσβαση, καθιστούν σχεδόν αδύνατη την ανάλυση του προγράμματος.

Τα κυριότερα εργαλεία συμβολικής εκτέλεσης είναι τα εξής [8]:

- KLEE (LLVM)
- FuzzBALL (VineIL / Native)
- JPF (Java)
- jCUTE (Java)
- janala2 (Java)
- KeY (Java)
- S2E (x86, x86-64, ARM / User and kernel-mode binaries)
- Pathgrind (Native 32-bit Valgrind-based)
- Mayhem (Binary)
- Otter (C)
- SymDroid (Dalvik bytecode)
- Rubyx (Ruby)
- Pex (.NET Framework)
- SymJS (JavaScript)

- Jalangi2 (JavaScript)
- Kite (LLVM)
- pysymemu (x86-64 / Native)
- Triton (x86 and x86-64)
- angr (libVEX based)

2.5 Ενδιάμεση Αναπαράσταση

Πολλές περιπτώσεις ευπαθειών ασφαλείας οφείλονται σε μικρά λάθη, εύκολα στην εύρεση τους, όπως η μη εξέταση της τιμής επιστροφής της συνάρτησης malloc. Οι περιπτώσεις αυτές δεν διαφοροποιούνται, ως προς τον έλεγχο, ανά αρχιτεκτονική, παρ' όλα αυτά όμως, η κάθε αρχιτεκτονική χρησιμοποιεί το δικό της σύνολο εντολών για την υλοποίηση συγκεκριμένων ενεργειών, όπως για παράδειγμα η αντιγραφεί της τιμής ενός εξωτερικού ορίσματος στην στοίβα της συνάρτησης.

Η λύση του προβλήματος των ευρέως διαφορετικών αρχιτεκτονικών, είναι η χρήση μιας ενδιάμεσης αναπαράστασης (Intermediate Representation). Μια ενδιάμεση αναπαράσταση αφαιρεί αρκετές διαφορές που υπάρχουν μεταξύ των αρχιτεκτονικών, επιτρέποντας έτσι μια ενιαία κοινή ανάλυση για όλες τις αρχιτεκτονικές. Διαφορές όπως, ονόματα καταχωρητών, ο τρόπος που γίνεται η πρόσβαση στη μνήμη και η τμηματοποίησή της και καταστάσεις που δημιουργούνται από κάποιες εντολές, π.χ. αλλαγή κάποιου flag κατά την εκτέλεση κάποιας εντολής, καλύπτονται από την ενδιάμεση αναπαράσταση.

Υπάρχουν πολλές ενδιάμεσες αναπαραστάσεις. Μια από της κυριότερες είναι η VEX. Υποστηρίζει την ανάλυση δυαδικών αρχείων, αγνοεί την αρχιτεκτονική προέλευσης του προγράμματος και δεν επηρεάζεται από παραπλήσιες ενέργειες που δημιουργούνται από συγκεκριμένες εντολές. Για την επίτευξη της ενδιάμεσης αναπαράστασης η VEX χρησιμοποιεί πέντε κυρίως κλάσεις:

- **Εκφράσεις (Expressions).** Αντιπροσωπεύουν μια υπολογισμένη ή σταθερή τιμή.
- **Λειτουργίες (Operations).** Περιγράφουν μια τροποποίηση των Εκφράσεων δίνοντας μία Έκφραση ως αποτέλεσμα.
- **Προσωρινές Μεταβλητές (Temporary Variables).** Χρησιμοποιούνται ως εσωτερικοί καταχωρητές για την αποθήκευση των περιεχομένων των Εκφράσεων κατά την χρήση τους και το περιεχόμενό τους μπορεί να ανακτηθεί χρησιμοποιώντας μία Έκφραση. Είναι αριθμημένες, ξεκινώντας από το t0.
- **Δηλώσεις (Statements).** Δηλώνουν αλλαγές στην κατάσταση του μηχανήματος, όπως η αποθήκευση στην μνήμη και η εγγραφεί σε έναν καταχωρητή. Χρησιμοποιούν Εκφράσεις για τιμές που ενδεχομένως να χρειάζονται, όπως η θέση μνήμης ή η τιμή που θα γραφτεί.
- **Μπλοκ (Block).** Αποτελείται από ένα σύνολο Καταστάσεων και αντιπροσωπεύει μια εκτεταμένη έκδοση του βασικού μπλοκ (που ονομάζεται "IR Super Block" ή "IRSB") της αρχικής αρχιτεκτονικής. Ένα μπλοκ μπορεί να έχει πολλές εξόδους.

Η βιβλιοθήκη PyVEX [9] [10] αποτελεί την υλοποίηση της VEX στην γλώσσα Python. Άλλη μία προσέγγιση ενδιάμεσης αναπαράστασης, η REIL (Reverse Engineering Intermediate Language) [11] [12] [13] [14] [15] [16], υλοποιείται στην βιβλιοθήκη openREIL [17], όπου κατά την διαδικασία της μετάφρασης χρησιμοποιεί την VEX. Και οι δύο υλοποιήσεις παράγουν αποτέλεσμα με πανομοιότυπες εντολές, μεταφράζοντας αρκετές αρχιτεκτονικές, όπως intel x86/x64 και arm x86/x64. Παρόλο που η υλοποίηση openREIL παρέχει αρκετές δυνατότητες, ακόμα και πέρα από το πεδίο της ενδιάμεσης αναπαράστασης, η υλοποίηση PyVEX παρέχει μία πιο σταθερή έκδοση ενδιάμεσης αναπαράστασης.

3 Σχεδιασμός

3.1 Εισαγωγή

Στόχος είναι η υλοποίηση ενός εργαλείου το οποίο θα παρέχει ελέγχους για βασικές περιπτώσεις ευπαθειών ασφαλείας, που εμφανίζονται υπό συνθήκες κατά την χρήση συναρτήσεων της βασικής βιβλιοθήκης libc [18] [19], σε βιβλιοθήκες αρχιτεκτονικών intel x86 [20], intel x64 [21], arm x32 και arm x64 [22]. Τέτοιες περιπτώσεις μπορούν να προκύψουν λόγω εσφαλμένου εξωτερικού ορίσματος που χρησιμοποιήθηκε, είτε αυτό είναι αριθμητικό είτε αλφαριθμητικό, είτε ακόμη και κάποιο αρχείο, έλλειψη ελέγχου της τιμής που επιστρέφει κάποια συνάρτηση ή ο συνδυασμός των δύο. Οι ευπάθειες αυτές κάποιες φορές έχουν αντίκτυπο μέσα στην ίδια βιβλιοθήκη και κάποιες άλλες στην εφαρμογή που την χρησιμοποιεί.

3.2 Διαφορετικές Αρχιτεκτονικές

Για την αντιμετώπιση του προβλήματος των διαφορετικών γλωσσών χαμηλού επιπέδου ανά αρχιτεκτονική, θα γίνει η χρήση μιας ενδιάμεσης αναπαράστασης στο πρώτο στάδιο της ανάλυσης. Με τον τρόπο αυτό, μεταφράζονται όλες οι γλώσσες χαμηλού επιπέδου σε μία ενιαία αναπαράσταση, λύνοντας το πρόβλημα των διαφορετικών εντολών και καταστάσεων ανά αρχιτεκτονική. Παρόλα αυτά, υπάρχουν διαφορές στις οποίες μία ενδιάμεση αναπαράσταση δεν μπορεί να ξεπεράσει. Τέτοιες διαφορές βρίσκονται είτε πριν την χρήση της ενδιάμεσης αναπαράστασης, όσον αναφορά την ανάλυση του αρχείου, είτε μετά, όπως είναι τα ονόματα των καταχωριτών. Η λύση στο πρόβλημα αυτό, είναι η ξεχωριστή ανάλυση ανά αρχιτεκτονική, που θα παρέχει βασικές πληροφορίες για την εκάστοτε αρχιτεκτονική, με την χρήση ίδιου interface, για την απαλοιφή της αρχιτεκτονικής κατά την ανάλυση. Για παράδειγμα, μία συνάρτηση με όνομα frame_pointer() θα επιστρέφει κάθε φορά μία λίστα με πληροφορίες σχετικά με τον frame pointer. Η ίδια συνάρτηση έχει διαφορετικές υλοποιήσεις ανά αρχείο αρχιτεκτονικής, αλλά έχει ίδιο interface σε όλα τα αρχεία. Οπότε, όταν κληθεί κατά την διαδικασία της ανάλυσης, θα επιστρέψει μία λίστα με πληροφορίες για τον frame pointer της αρχιτεκτονικής που αναλύεται, χωρίς όμως να φαίνεται η ίδια η αρχιτεκτονική στο επίπεδο της ανάλυσης. Επίσης, στο ίδιο επίπεδο θα γίνεται η ανάλυση του αρχείου, δηλαδή, αφού καθοριστεί η αρχιτεκτονική του, καλείται η αντίστοιχη συνάρτηση εύρεσης των δυναμικών συμβόλων του αρχείου, των διευθύνσεων που καλούνται αυτά, των συναρτήσεων που είναι υλοποιημένες μέσα σε αυτό, καθώς και το μέγεθος της στοίβας της κάθε συνάρτησης.

3.3 Σύμβολα και Συναρτήσεις

Η ύπαρξη βιβλιοθηκών εξυπηρετεί την πολλαπλή χρήση του ίδιου κώδικα με την μορφή συναρτήσεων. Οι συναρτήσεις αυτές υλοποιούνται μία φορά μέσα σε μια βιβλιοθήκη και κάθε φορά που γίνεται χρήση αυτών, είτε από εφαρμογές, είτε από άλλες βιβλιοθήκες, μεταφέρεται η ροή του προγράμματος στην υλοποίηση της συνάρτησης, συγκρατείται στην στοίβα μία διεύθυνση επιστροφής, στην οποία επιστρέφει η ροή του προγράμματος όταν ολοκληρωθεί η συνάρτηση. Η σύνδεση του προγράμματος με την υλοποίηση των συναρτήσεων γίνεται με την χρήση συμβόλων. Για κάθε εξωτερική συνάρτηση χρησιμοποιείται ένα σύμβολο για την διασύνδεση με τη υλοποίησή της. Αν η συνάρτηση αυτή χρησιμοποιηθεί περισσότερες από μία φορές, τότε χρησιμοποιείται το ίδιο σύμβολο για την εύρεση της.

Εξάγοντας πληροφορίες για τα σύμβολα μίας βιβλιοθήκης, γνωρίζουμε ποιες είναι οι εξωτερικές συναρτήσεις άλλης βιβλιοθήκης που χρησιμοποιούνται, καθώς και από ποια βιβλιοθήκη προέρχονται. Έτσι, μπορούν να εντοπιστούν σημεία που γίνεται η χρήση ευπαθών εξωτερικών συναρτήσεων ή σημεία στα οποία συγκεκριμένες εξωτερικές συναρτήσεις μπορούν να προκαλέσουν υπό συνθήκες ευπάθειες ασφαλείας, όπως για παράδειγμα η χρήση εξωτερικού ορίσματος χωρίς έλεγχο στην system() της βιβλιοθήκης libc.

Στο δεύτερο στάδιο της ανάλυσης του αρχείου, οι πληροφορίες του αρχείου βιβλιοθήκης προς έλεγχο, θα διαχωριστούν σε δύο κατηγορίες. Η πρώτη κατηγορία παρέχει πληροφορίες για ένα σύμβολο

που καλείται μέσα σε μία συνάρτηση. Τέτοιες πληροφορίες είναι οι διευθύνσεις που κλήθηκε το σύμβολο μέσα σε μία συνάρτηση (ενδεχομένως να καλείτε περισσότερες από μία φορά), το όνομα του συμβόλου και η βιβλιοθήκη στην οποία έχει υλοποιηθεί. Η δεύτερη κατηγορία παρέχει πληροφορίες για μία συνάρτηση που υλοποιείται μέσα στο αρχείο της βιβλιοθήκης προς έλεγχο. Περιέχει το όνομα, την διεύθυνση, τον τομέα, το μέγεθος και το μέγεθος της στοίβας της συνάρτησης, καθώς επίσης και μία λίστα από σύμβολα που καλούνται μέσα σε αυτήν και μία λίστα με τα μεγέθη των τοπικών μεταβλητών της στοίβας. Η συλλογή και η κατηγοριοποίηση των πληροφοριών αυτών γίνεται στο πρώτο επίπεδο ανάλυσης της αντίστοιχης αρχιτεκτονικής, όπως αναφέρθηκε στην προηγούμενη υποενότητα (βλ. 3.2 Διαφορετικές Αρχιτεκτονικές).

3.4 Ανάλυση Αρχείου

Στο τρίτο και τελευταίο στάδιο της ανάλυσης του αρχείου, συλλέγονται όλες οι πληροφορίες του αρχείου της βιβλιοθήκης προς έλεγχο, συμπεριλαμβανομένου και αυτές που προέκυψαν από τα προηγούμενα δύο στάδια της ανάλυσης (βλ. 3.2 Διαφορετικές Αρχιτεκτονικές κ 3.3 Σύμβολα και Συναρτήσεις). Γίνεται διαχώριση της αρχιτεκτονικής του αρχείου, εμπεριέχει την δυαδική αναπαράσταση του αρχείου, πληροφορίες της αρχιτεκτονικής του, μία λίστα από συναρτήσεις που υλοποιούνται μέσα στο αρχείο και συναρτήσεις που καλούν συναρτήσεις της αντίστοιχης αρχιτεκτονικής.

Εικόνα 1 Στάδια Ανάλυσης Αρχείου

3.5 Ανάλυση Σεναρίων

Η ανάλυση σεναρίων θα γίνει με βάση μία ενδιάμεση αναπαράσταση. Στόχος είναι η αναγνώριση και η κατηγοριοποίηση ορισμένων σεναρίων και περιπτώσεων, με σκοπό την απλοποίηση της ανάλυσης. Μπορούμε να διακρίνουμε περιπτώσεις, οι οποίες εμφανίζονται παραπάνω από μία φορά στα σενάρια.

Οι περιπτώσεις αυτές μπορούν να υλοποιηθούν η κάθε μία ξεχωριστά ως συναρτήσεις, με σκοπό την επαναλαμβανόμενη χρήση τους κατά την ανάλυση σεναρίων . Τέτοιες περιπτώσεις είναι η εύρεση του offset της στοίβας μίας μεταβλητής, ο έλεγχος για το αν η τιμή μίας μεταβλητής προήλθε από κάποιο όρισμα, ο έλεγχος αν η τιμή μίας αλφαριθμητικής μεταβλητής βρίσκεται στον τομέα .rodata, καθώς επίσης και η εύρεση τιμής ενός καταχωρητή, είτε αυτός είναι προσωρινός καταχωρητής της ενδιάμεσης αναπαράστασης, είτε είναι καταχωρητής της αρχιτεκτονικής. Ο έλεγχος των σεναρίων μπορεί να δομηθεί με την χρήση των περιπτώσεων αυτών και τον συνδυασμό τους.

3.6 Σενάρια

Στόχος είναι ο έλεγχος των σεναρίων σε βασικές συναρτήσεις της libc, οι οποίες υπό περιπτώσεις προκαλούν ευπάθειες ασφαλείας. Ο έλεγχος αυτός θα γίνεται με την χρήση και τον συνδυασμό συναρτήσεων που περιγράφονται στην προηγούμενη υποενότητα (βλ. 3.4 Ανάλυση). Τα σενάρια που θα εξεταστούν είναι τα παρακάτω:

- **Άνοιγμα αρχείων με την χρήση των συναρτήσεων fopen(), open().** Ένα κακόβουλο αρχείο μπορεί να προκαλέσει code injection [23], code execution και κατά συνέπεια privilege escalation. Επίσης, το όνομα του αρχείου προς άνοιγμα μπορεί να είναι ένα symbolic link [24] σε κρίσιμα αρχεία, όπως για παράδειγμα αρχεία συστήματος.
- **Δέσμευση μνήμης με την χρήση της συνάρτησης malloc().** Αν ζητηθεί μεγάλος αριθμός μνήμης με την χρήση της malloc(), υπάρχει το ενδεχόμενο να μην επαρκεί η διαθέσιμη υπάρχουσα μνήμη. Σε αυτήν την περίπτωση, η συνάρτηση malloc() επιστρέφει την τιμή NULL, δηλαδή την τιμή μηδέν (0). Αν δεν ελεγχθεί η τιμή επιστροφής, τότε ο δείκτης θέσης μνήμης που υπό κανονικές συνθήκες θα έδειχνε την θέση μνήμης που δεσμεύτηκε, πλέον δείχνει στην αρχή της μνήμης (θέση μηδέν(0)). Θεωρώντας ότι ο δείκτης δείχνει σε σωστή θέση, οι λειτουργίες του προγράμματος σχετικά με τον δείχτη αυτό, γίνονται στην αρχή της μνήμης (θέση μηδέν(0)), με ενδεχόμενο αποτέλεσμα την επικάλυψη κρίσιμων δεδομένων στην συγκεκριμένη θέση ή την ανάγνωσή τους.
- **Αντιγραφή μνήμης με την χρήση της συνάρτησης memcpy().** Η συνάρτηση memcpy() δεν ελέγχει το μέγεθος μνήμης του προορισμού, με αποτέλεσμα να είναι ευάλωτη σε ευπάθειες buffer overflow [25]. Αυτό μπορεί να συμβεί, στην περίπτωση που το μέγεθος μνήμης προορισμού είναι μικρότερο από αυτό της πηγής. Κατά συνέπεια, η συνάρτηση θα γράψει έξω από τα όρια μνήμης προορισμού, προκαλώντας ενδεχομένως αλλοίωση δεδομένων, απότομο τερματισμό του προγράμματος και εκτέλεση κακόβουλου κώδικα.
- **Αντιγραφή αλφαριθμητικών τιμών με την χρήση των συναρτήσεων snprintf(), sprintf(), strcat(), strcpy(), strncat(), strncpy().** Οι συναρτήσεις αυτές, επίσης δεν ελέγχουν το μέγεθος προορισμού με αποτέλεσμα να είναι και αυτές ευάλωτες σε ευπάθειες buffer overflow [25] [24]. Επίσης, είναι ευάλωτες σε ευπάθειες format string bugs [26]. Συγκεκριμένες μορφές αλφαριθμητικών τιμών μπορούν να προκαλέσουν διαρροή πληροφοριών και μεταφορά της ροής του προγράμματος σε κακόβουλο κώδικα μέσα από την επικάλυψη μνήμης.
- **Εκτέλεση εντολών συστήματος με την χρήση της συνάρτησης system().** Εντολές που προέρχονται από εξωτερικούς παράγοντες είναι ευάλωτες σε ευπάθειες command injection [27]. Οι εξωτερικοί αυτοί παράγοντες ενδέχεται να περιέχουν κακόβουλες εντολές, οι οποίες θα εκτελεστούν με τα δικαιώματα του εκάστοτε προγράμματος, μαζί με τις εντολές που ήταν προγραμματισμένες να εκτελεστούν εξαρχής.

  Αναλυτικά προκύπτουν οι εξής έλεγχοι:

- **fopen(), open():** Αν το όνομα του αρχείου (πρώτο όρισμα) προέρχεται από εξωτερική μεταβλητή και αν ελέγχεται η τιμή επιστροφής της συνάρτησης
- **malloc():** Αν η τιμή της μεταβλητής που δίνεται ως όρισμα προέρχεται από εξωτερική τιμή (προειδοποίηση) και αν ελέγχεται η τιμή επιστροφής της συνάρτησης
- **memcpy():** Προειδοποίηση αν το δεύτερο όρισμα προέρχεται από εξωτερική μεταβλητή και έλεγχος αν το πρώτο όρισμα είναι εσωτερική μεταβλητή της στοίβας, ενώ η τιμή του δεύτερου ορίσματος προέρχεται από εξωτερική μεταβλητή.

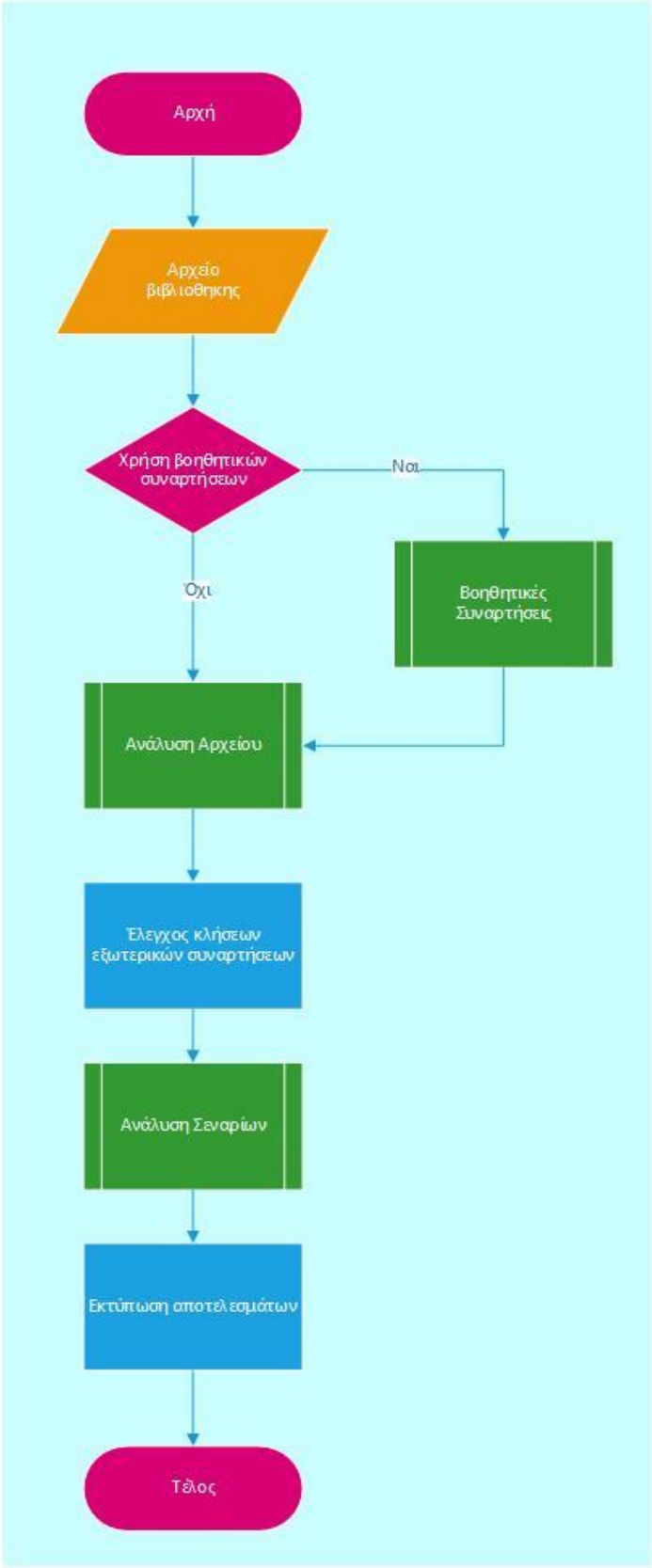
- **snprintf():** Προειδοποίηση αν το τρίτο όρισμα προέρχεται από εξωτερική μεταβλητή και αν ελέγχεται η τιμή επιστροφής της συνάρτησης.
- **sprintf():** Προειδοποίηση αν το δεύτερο όρισμα προέρχεται από εξωτερική μεταβλητή.
- **strcat (), strcpy(), strncat(), strncpy():** Προειδοποίηση αν το δεύτερο όρισμα προέρχεται από εξωτερική μεταβλητή και έλεγχος αν το πρώτο όρισμα είναι εσωτερική μεταβλητή της στοίβας, ενώ η τιμή του δεύτερου ορίσματος προέρχεται από εξωτερική μεταβλητή.
- **system():** Έλεγχος αν το όρισμα προέρχεται από εξωτερική μεταβλητή.

3.7 Βοηθητικές Συναρτήσεις

Για την ευκολότερη κατανόηση των βιβλιοθηκών που θα χρησιμοποιηθούν και την καλύτερη υλοποίηση των σεναρίων, θα αναπτυχθούν βοηθητικές συναρτήσεις, κάποιες με την χρήση των βιβλιοθηκών που θα χρησιμοποιηθούν και στο στάδιο της ανάλυσης, οι οποίες θα είναι προαιρετικές και θα καλούνται με τα αντίστοιχα ορίσματα. Θα παρέχεται η εκτύπωση των συμβόλων του αρχείου, αναλυτική λίστα των συναρτήσεων και των διευθύνσεων αυτών που βρέθηκαν στο αρχείο, εκτύπωση ενδιάμεσης αναπαράστασης, disassemble, πληροφορίες τομέων, bytes του αρχείου σε δεκαεξαδική μορφή, καθώς επίσης και εκτύπωση σημείων του αρχείου με την μορφή αλφαριθμητικών μεταβλητών, για έλεγχο ύπαρξης αυτών. Η διεύθυνση ή ο τομέας και το μέγεθος προς ανάλυση της εκάστοτε συνάρτησης θα ελέγχονται επίσης από τον χρήστη με αντίστοιχα ορίσματα. Πολλές από τις παραπάνω λειτουργίες παρέχονται ήδη από εργαλεία, όπως το objdump (εργαλείο για disassemble αρχείων) [28] και το readelf (εργαλείο ανάγνωσης κεφαλίδων αρχείων βιβλιοθηκών) [29]. Παρ' όλα αυτά η χρήση των βοηθητικών αυτών συναρτήσεων θα προσφέρει ακρίβεια στα αποτελέσματα των βιβλιοθηκών, πράγμα αναγκαίο για την σωστή υλοποίηση της ανάλυσης.

3.8 Κύρια ροή

Το εργαλείο ανάλυσης θα δέχεται ως υποχρεωτικό όρισμα ένα αρχείο προς ανάλυση και προαιρετικά ορίσματα βοηθητικών συναρτήσεων. Αν έχει γίνει χρήση ορισμάτων βοηθητικών συναρτήσεων, γίνεται η κλήση της εκάστοτε συνάρτησης. Γίνεται η ανάλυση του αρχείου και έπειτα, γίνεται έλεγχος στις εξωτερικές συναρτήσεις που καλούνται μέσα στο αρχείο και για όποια συνάρτηση πληρούνται οι προϋποθέσεις ελέγχου, τότε καλείτε η συνάρτηση ανάλυσης του αντίστοιχου σεναρίου. Οι συναρτήσεις προς έλεγχο είναι αυτές που προέρχονται από εξωτερικές βιβλιοθήκες της libc και έχει υλοποιηθεί σενάριο προς ανάλυση για την εκάστοτε συνάρτηση. Για κάθε συνάρτηση προς έλεγχο θα υλοποιηθεί διαφορετική συνάρτηση ανάλυσης, για την ευκολία ανάπτυξης και προσθήκης εξειδικευμένων σεναρίων, παρόλο που κάποια σενάρια έχουν πολλές ομοιότητες μεταξύ τους.



Εικόνα 2 Κύρια Ροή

4 Υλοποίηση

4.1 Εισαγωγή

Για την υλοποίηση του εργαλείου γίνεται χρήση εξωτερικών βιβλιοθηκών, όπως η βιβλιοθήκη pyvex (ενδιάμεση αναπαράσταση) [9] και η βιβλιοθήκη pybfd [30] (εξάγει τις πληροφορίες ενός αρχείου) οι οποίες θα αναλυθούν στο επόμενο κεφάλαιο. Για το πρώτο στάδιο ανάλυσης δημιουργήθηκαν διαφορετικά αρχεία για κάθε αρχιτεκτονική (carrion_arm.py, carrion_aarch64.py, carrion_i386.py, carrion_64.py), τα οποία καλύπτουν βασικές διαφορές μεταξύ των αρχιτεκτονικών. Η εξαγωγή των πληροφοριών σχετικά με τα εσωτερικά και εξωτερικά σύμβολα του αρχείου προς έλεγχο, κατά το δεύτερο στάδιο της ανάλυσης, υλοποιείται εξίσου στο αρχείο της κάθε αρχιτεκτονικής, μιας και οι πληροφορίες αυτές είναι διαφορετικά δομημένες ανά αρχιτεκτονική. Στο τρίτο και τελευταίο στάδιο της ανάλυσης, δημιουργήθηκαν τα αρχεία carrion_base.py και carrion.py, τα οποία αποτελούν την υλοποίηση της ανάλυσης του πηγαίου κώδικα του αρχείου προς έλεγχο, από απλές πράξεις μέχρι και την υλοποίηση των σεναρίων (βλ. 3.6 Σενάρια). Οι υλοποιήσεις του αρχείου carrion_base.py αποτελούν βασικές πράξεις και λειτουργίες και δεν χρησιμοποιούν κάποια κλάση ή συνάρτηση που η υλοποίησή τους βρίσκεται σε άλλο αρχείο, όπως συμβαίνει στο αρχείο carrion.py. Στο αρχείο carrion.py βρίσκεται επίσης η κύρια κλάση Carrion, η οποία εμπεριέχει όλες της πληροφορίες του αρχείου προς ανάλυση, καθώς και κάποιες συναρτήσεις σχετικές με αυτές και τη main συνάρτηση του εργαλείου. Τέλος, δημιουργήθηκε το αρχείο carrion_info.py, στο οποίο περιέχονται βοηθητικές συναρτήσεις που εξάγουν διαφόρων ειδών πληροφορίες, χρήσιμες για περεταίρω ανάλυση του αρχείου.

4.2 Σύμβολα και Συναρτήσεις

Όπως αναφέρθηκε στο κεφάλαιο 3.3, οι πληροφορίες του αρχείου προς έλεγχο, όσον αναφορά τα σύμβολα και τις συναρτήσεις αυτών, θα διαχωριστούν σε δύο κατηγορίες. Η πρώτη κατηγορία είναι οι συναρτήσεις που υλοποιούνται σε άλλη βιβλιοθήκη και γίνεται η κλήση αυτών στο αρχείο προς έλεγχο και η δεύτερη κατηγορία είναι οι συναρτήσεις που υλοποιούνται μέσα στο αρχείο προς έλεγχο. Οι πληροφορίες αυτές υλοποιούνται με την μορφή μίας κλάσης ανά κατηγορία, οι οποίες βρίσκονται μέσα στο αρχείο carrion_base.py. Αναλυτικά προκύπτουν οι κλάσεις:

class ExternalFunction: Αντιπροσωπεύει σύμβολα συναρτήσεων που υλοποιούνται εξωτερικά της βιβλιοθήκης προς ανάλυση. Περιέχει:

- το όνομα της συνάρτησης (*name*),
- το όνομα της εξωτερικής βιβλιοθήκης που υλοποιείται η συνάρτηση (*library*) και
- μία λίστα με διευθύνσεις που καλείται το σύμβολο μέσα σε μία συνάρτηση της βιβλιοθήκης προς ανάλυση. Αν το σύμβολο καλείται σε περισσότερες από μία συναρτήσεις, τότε θα δημιουργηθούν από ένα αντικείμενο της κλάσης αυτής για κάθε συνάρτηση που βρέθηκε να καλείται το σύμβολο (*addresses*).

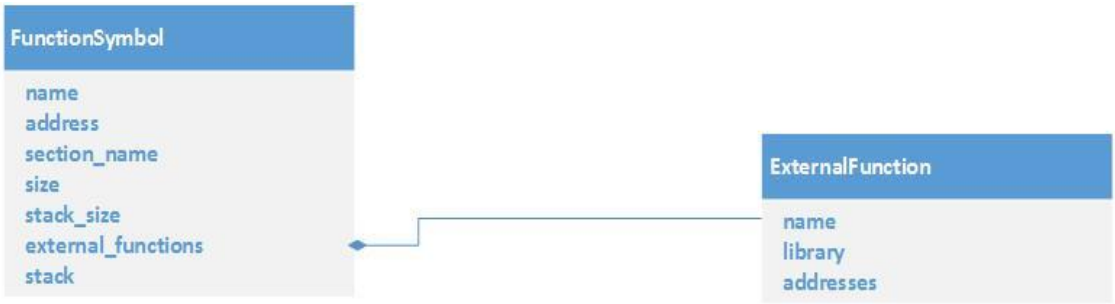
__init__(name, library): Δέχεται ως ορίσματα το όνομα της συνάρτησης και της βιβλιοθήκης στην οποία υλοποιείται και αρχικοποιείται η λίστα των διευθύνσεων που θα βρεθεί το σύμβολο.

class FunctionSymbol: Αντιπροσωπεύει σύμβολα συναρτήσεων που υλοποιούνται εσωτερικά της βιβλιοθήκης προς ανάλυση. Περιέχει:

- το όνομα της συνάρτησης (*name*),
- την διεύθυνση της υλοποίησης της συνάρτησης (*address*),
- το όνομα του τομέα της συνάρτησης (*section_name*),
- το μέγεθος της (*size*),
- το μέγεθος του stack (*stack_size*),
- ανάλυση του stack, με την μορφή dictionary, έχοντας ως key το offset από την αρχή του stack της εκάστοτε εγγραφής και ως value το μέγεθος της εγγραφής αυτής (*stack*) και
- ένα dictionary με τις πληροφορίες των εξωτερικών συμβόλων που βρέθηκαν να καλούνται μέσα στην συνάρτηση, δομημένο ως εξής: ως key έχει

[όνομα_εξωτερικού_συμβόλου]@[όνομα_βιβλιοθήκης_υλοποίησης_του_εξωτερικού_συμβόλου], και ως value ένα αντικείμενο της κλάσης ExternalFunction με όλες τις σχετικές πληροφορίες του εξωτερικού συμβόλου. Η χρήση του dictionary αποσκοπεί στην εύκολη εύρεση των ήδη υπάρχων συμβόλων μέσα στη λίστα, έτσι ώστε να γίνεται εύκολη η ομαδοποίηση των κλήσεων ανά σύμβολο (*external_functions*).

__init__(name, address, section_name, size=0, stack_size=0): Δέχεται ως ορίσματα το όνομα (name), την διεύθυνση (address), το όνομα τομέα της συνάρτησης (section_name) και προαιρετικά το μέγεθος της συνάρτησης (size) και το μέγεθος του stack (stack_size) (αρχικοποίηση στο 0). Επίσης, αρχικοποιούνται τα dictionaries για τα εξωτερικά σύμβολα της συνάρτησης και για την ανάλυση του stack.



Εικόνα 3 Κλάσεις απεικόνισης συμβόλων

4.3 Ανάλυση Συμβόλων και Συναρτήσεων

Η ανάλυση των εσωτερικών και εξωτερικών συμβόλων και των συναρτήσεων τους ενός αρχείου προς έλεγχο παρουσιάζει μικρές ομοιότητες και αρκετές διαφορές μεταξύ των αρχιτεκτονικών. Η εύρεση των συναρτήσεων που υλοποιούνται εσωτερικά του αρχείου, είναι εύκολη και ίδια, ανεξαρτήτως αρχιτεκτονικής, με την χρήση των συμβόλων της βιβλιοθήκης pybfd. Η διαδικασία αυτή γίνεται κατά την αρχικοποίηση ενός αντικειμένου της κλάσης Carrion (βλ 4.5 Ανάλυση πηγαίου κώδικα). Άλλη μία ομοιότητα που παρουσιάζεται μεταξύ των αρχιτεκτονικών είναι η εύρεση του μεγέθους μίας συνάρτησης**.** Λόγο της ομοιότητας αυτής και της ανεξαρτησίας της υλοποίησης, η παρακάτω συνάρτηση βρίσκεται στο αρχείο carrion_base.py.

findFunctionSizeByAddress(bfd, bytestream, address)**:** Δέχεται ως ορίσματα ένα binary file descriptor (bfd), ένα byte stream (bytestream) και μία διεύθυνση σε αριθμητική μορφή (address) και επιστρέφει το μέγεθος της συνάρτησης που βρίσκεται στην διεύθυνση address, εάν βρεθεί, διαφορετικά επιστρέφει την τιμή 0. Η εύρεση του μεγέθους της συνάρτησης γίνεται μέσα από τις πληροφορίες που περιέχονται στον τομέα .dynsym. Γίνεται χρήση του bfd για την εύρεση του file offset, του μεγέθους των εγγραφών του τομέα .dynsym και του endian της βιβλιοθήκης και χρήση του bytestream για την ανάγνωση των εγγραφών του τομέα .dynsym. Ελέγχεται το value της κάθε εγγραφής, το οποίο αντιπροσωπεύει την διεύθυνση στην οποία ξεκινάει η υλοποίηση της συνάρτησης του συμβόλου. Αν το value αυτό είναι ίδιο με την διεύθυνση address που δόθηκε, τότε γίνεται ανάγνωση του μεγέθους του συμβόλου, το οποίο και επιστρέφεται σε αριθμητική μορφή.

Η αναζήτηση πληροφοριών συμβόλων εξωτερικών συναρτήσεων, διαφέρει ανά αρχιτεκτονική, κάτι το οποίο απαιτεί και διαφορετική υλοποίηση ανά περίπτωση. Οι πληροφορίες αυτές εξάγονται από τους τομείς της βιβλιοθήκης προς έλεγχο. Οι τομείς που μας ενδιαφέρουν είναι οι εξής:

- *.plt (arm/aarch64/i386/x64):* Περιέχει εντολές αρχικοποίησης των εξωτερικών συμβόλων και εντολές μετάβασης της ροής του προγράμματος στην υλοποίηση του συμβόλου που κλήθηκε. Κατά την κλήση ενός εξωτερικού συμβόλου, καλείται πρώτα η αντίστοιχη διεύθυνση στον τομέα .plt, από όπου καθορίζεται η επόμενη κλήση.
- *.rel.plt (arm /i386):* Αποτελεί έναν πίνακα με εγγραφές, όπου κάθε εγγραφή περιέχει την διεύθυνση του τομέα .got για αρχιτεκτονική arm ή του τομέα .got.plt για αρχιτεκτονική i386

ενός συμβόλου και τον αύξον αριθμό της εγγραφής του ίδιου συμβόλου στον τομέα .dynsym, για αρχιτεκτονικές 32 bits.

- *.rela.plt (aarch64 /x64):* Αποτελεί έναν πίνακα με εγγραφές, όπου κάθε εγγραφή περιέχει την διεύθυνση του τομέα .got για αρχιτεκτονική aarch64 ή του τομέα .got.plt για αρχιτεκτονική x64 ενός συμβόλου και τον αύξον αριθμό της εγγραφής του ίδιου συμβόλου στον τομέα .dynsym, για αρχιτεκτονικές 64 bits.
- *.dynsym (arm/aarch64/i386/x64):* Αποτελεί έναν πίνακα με εγγραφές, όπου κάθε εγγραφή περιέχει διάφορες πληροφορίες για ένα σύμβολο. Χρήσιμες πληροφορίες του τομέα .dynsym είναι το offset από την αρχή του τομέα .dynstr, όπου βρίσκεται το όνομα του συμβόλου.
- *.dynstr (arm/aarch64/i386/x64):* Περιέχει σε μορφή αλφαριθμητικών τιμών, τα ονόματα των συμβόλων και των βιβλιοθηκών τους.
- *.gnu_version (arm/aarch64/i386/x64):* Αποτελεί έναν πίνακα με εγγραφές μεγέθους 2 bytes, όπου κάθε εγγραφή αντιστοιχίζεται με τις εγγραφές του τομέα .dynsym, με βάση τον αύξον αριθμό της κάθε εγγραφής και περιέχει μια τιμή του τομέα .gnu_version_r
- *.gnu_version_r (arm/aarch64/i386/x64):* Αποτελεί έναν πίνακα με εγγραφές, οι οποίες περιέχουν μία τιμή, ξεχωριστή για κάθε εγγραφή και ένα offset από την αρχή του τομέα .dynstr, όπου βρίσκεται το όνομα της βιβλιοθήκης που αντιστοιχεί στην ξεχωριστή τιμή.

Κατά συνέπεια προκύπτει διαφορετική υλοποίηση της ίδιας συνάρτησης στις κλάσεις Carrion_arm, Carrion_aarch64, Carrion_i386 και Carrion_64 στα αρχεία carrion_arm, carrion_aarch64, carrion_i386 και carrion_64 αντίστοιχα:

***scan_function_for_external(bfd, bytestream, fsym)*:** Δέχεται ως ορίσματα ένα binary file descriptor (bfd) της βιβλιοθήκης pybfd, το αντίστοιχο byte stream (bytestream) του bfd και ένα αντικείμενο της κλάσης FunctionSymbol (fsym), το οποίο περιέχει ήδη όλες τις πληροφορίες μίας συνάρτησης που υλοποιείται στο αρχείο του bfd, εκτός από τα εξωτερικά σύμβολα που καλούνται μέσα σε αυτή και τις πληροφορίες για το stack. Επιστρέφει το αντικείμενο της κλάσης FunctionSymbol fsym, συμπληρωμένο με όλα τα εξωτερικά σύμβολα, τις πληροφορίες αυτών και τις διευθύνσεις όπου καλούνται μέσα στη συνάρτηση, καθώς και τις πληροφορίες του stack (συνολικό μέγεθος, ανάλυση του stack σε μεγέθη εγγραφών).

Αλγόριθμος: Όλοι οι έλεγχοι και η εξόρυξη των απαραίτητων πληροφοριών που γίνονται σε αυτή τη συνάρτηση, γίνονται στο επίπεδο assembly της εκάστοτε αρχιτεκτονικής. Υπάρχουν αρκετές διαφορές μεταξύ των αρχιτεκτονικών, οι οποίες καθιστούσαν αδύνατη την υλοποίηση ενός ενιαίου αλγορίθμου κοινό για όλες τις αρχιτεκτονικές, με την χρήση ενδιάμεσης αναπαράστασης, οπότε η συνάρτηση αυτή υλοποιήθηκε διαφορετικά για κάθε αρχιτεκτονική. Όλες οι απαραίτητες πληροφορίες των τομέων της βιβλιοθήκης προς ανάλυση βρίσκονται από το αντικείμενο *bfd* (μέγεθος τομέα, εικονική διεύθυνση μνήμης της αρχής του τομέα, file offset της αρχής του τομέα).

4.3.1 Αλγόριθμος αρχιτεκτονικής arm32 συνάρτησης scan_function_for_external()

Η πρώτη εντολή αφαίρεσης (*sub)* που βρίσκεται από την αρχή της συνάρτησης, χρησιμοποιείται για την δέσμευση χώρου στο stack, οπότε και μας δίνει το μέγεθος της στοίβας.

Οι εντολές αποθήκευσης (*str*) γράφουν στη στοίβα κάποια μεταβλητή, είτε εσωτερική είτε εξωτερική. Βρίσκοντας όλα τα σημεία της στοίβας όπου έχει γίνει εγγραφή μεταβλητών, μπορούμε να υπολογίσουμε το μέγεθος της κάθε μεταβλητής.

Η εντολή διακλάδωσης με σύνδεσμο (*bl*) χρησιμοποιείται για την κλήση εξωτερικών (και όχι μόνο) συμβόλων. Αν η διεύθυνση της εντολής διακλάδωσης με σύνδεσμο ανήκει στον τομέα .plt, τότε πρόκειται για εξωτερικό σύμβολο.

Η πρώτη εντολή που βρίσκεται στην διεύθυνση του τομέα .plt της εντολής διακλάδωσης, είναι εντολή πρόσθεσης. Αποθηκεύεται στον καταχωρητή ip (r12 - Intra-Procedure-call scratch register) ο

program counter (pc) αυξημένος κατά 12. Ουσιαστικά αποθηκεύεται η διεύθυνση της τρίτης εν συνεχεία εντολής του τομέα .plt από το σημείο που κλήθηκε.

Η αμέσως επόμενη (δεύτερη) εντολή στον τομέα .plt, είναι επίσης εντολή πρόσθεσης, κατά την οποία προστίθεται μία σταθερά στην ήδη υπάρχων τιμή του καταχωρητή ip.

Η τρίτη και τελευταία εντολή του τομέα .plt που αντιστοιχεί στο σύμβολο που κλήθηκε, είναι εντολή φόρτωσης με offset (*ldr*). Φορτώνεται το περιεχόμενο του καταχωρητή ip με ένα offset στον program counter (pc), μεταφέροντας έτσι την ροή του προγράμματος στο σημείο όπου δείχνει πλέον ο pc, το οποίο αντιστοιχεί στην εγγραφή του τομέα .got, που αντιστοιχεί στο σύμβολο προς ανάλυση. Αναλυτικά, η διεύθυνση του τομέα .got, προκύπτει ως το άθροισμα της διεύθυνσης της εντολής *ldr* (η οποία είναι +8 από την διεύθυνση που χρησιμοποιήθηκε στην εντολή *bl*), της σταθερής τιμής που προστίθεται στην δεύτερη εντολή και του offset που χρησιμοποιήθηκε στην εντολή *ldr*. (Ο τομέας .got αρχικοποιείται σε όλες του τις εγγραφές με την διεύθυνση της πρώτης εντολής του τομέα .plt. Κατά την πρώτη κλήση ενός συμβόλου, ο τομέας .got καλεί τις πρώτες εντολές του τομέα .plt, οι οποίες είναι υπεύθυνες για την εύρεση της υλοποίησης του συμβόλου και για την ενημέρωση της αντίστοιχης εγγραφής στον τομέα .got. Για τον λόγο αυτό, στα πλαίσια της στατικής ανάλυσης δεν θα ασχοληθούμε περαιτέρω με το περιεχόμενο του τομέα .got).

Οι εγγραφές του τομέα .rel.plt αντιστοιχούν σε 8 bytes η κάθε μία, με τα 4 πρώτα bytes να περιέχουν την διεύθυνση του συμβόλου στον τομέα .got και τα 4 υπόλοιπα περιέχουν τον αύξον αριθμό εγγραφής του τομέα .dynsym και του τομέα .gnu.version που αντιστοιχούν στον σύμβολο.

Οι εγγραφές του τομέα .dynsym είναι μεγέθους 16 bytes. Τα πρώτα 4 bytes αποτελούν το offset από την αρχή του τομέα .dynstr που αντιστοιχεί στο όνομα του συμβόλου.

Οι εγγραφές του τομέα .gnu.version είναι μεγέθους 2 bytes, όπου η κάθε εγγραφή αντιστοιχεί σε μία εγγραφή του τομέα .dynsym, συσχετισμένες κατά αύξον αριθμό και περιέχει την τιμή του πεδίου value μίας εγγραφής του τομέα .gnu.version_r.

Οι εγγραφές του τομέα .gnu.version_r είναι μεγέθους 16 bytes, μεταξύ του έκτου(6) και όγδοου(8) byte βρίσκεται η τιμή του πεδίου value της εγγραφής και μεταξύ του όγδοου(8) και δωδέκατου(12) byte το offset από την αρχή του τομέα .dynstr που αντιστοιχεί στο όνομα της βιβλιοθήκης.

Όλες οι απαραίτητες πληροφορίες του συμβόλου βρίσκονται στους παραπάνω τομείς, αρκεί να γίνει η σωστή αντιστοίχιση των εγγραφών που ανήκουν στο ίδιο σύμβολο.

4.3.2 Αλγόριθμος αρχιτεκτονικής aarch64 συνάρτησης scan_function_for_external()

Η πρώτη εντολή αποθήκευσης ζευγαριού καταχωρητών (*stp)* που βρίσκεται από την αρχή της συνάρτησης, χρησιμοποιείται για την δέσμευση χώρου στο stack, οπότε και μας δίνει το μέγεθος της στοίβας.

Οι εντολές αποθήκευσης (*str*) γράφουν στη στοίβα κάποια μεταβλητή, είτε εσωτερική είτε εξωτερική. Βρίσκοντας όλα τα σημεία της στοίβας όπου έχει γίνει εγγραφή μεταβλητών, μπορούμε να υπολογίσουμε το μέγεθος της κάθε μεταβλητής.

Η εντολή διακλάδωσης με σύνδεσμο (*bl*) χρησιμοποιείται για την κλήση εξωτερικών (και όχι μόνο) συμβόλων. Αν η διεύθυνση της εντολής διακλάδωσης με σύνδεσμο ανήκει στον τομέα .plt, τότε πρόκειται για εξωτερικό σύμβολο.

Η εντολή που βρίσκεται στην διεύθυνση του τομέα .plt της εντολής διακλάδωσης, είναι η εντολή *adrp*. Η εντολή *adrp* υπολογίζει μια διεύθυνση σχετική με τον program counter (pc) και την αποθηκεύει στον καταχωρητή που δίνεται. Στην προκειμένη περίπτωση, η διεύθυνση αυτή, είναι η εικονική διεύθυνση της αρχής του τομέα .got.plt.

Η αμέσως επόμενη εντολή στον τομέα .plt, είναι εντολή φόρτωσης *ldr*. Φορτώνεται η τιμή που βρίσκεται στην διεύθυνση του καταχωρητή που αποθηκεύτηκε το αποτέλεσμα της προηγούμενης

εντολής (ο οποίος περιέχει τη διεύθυνση της αρχής του τομέα .got.plt), με κάποια σταθερά offset. Το offset αυτό είναι της εγγραφής του εξωτερικού συμβόλου στο τομέα .got.plt από την αρχή του.

Προσθέτοντας την σταθερά που βρέθηκε στην εντολή *adrp* (εικονική διεύθυνση της αρχής του τομέα .got.plt) και του offset της εντολής *ldr,* βρίσκεται η εικονική διεύθυνση της εγγραφής του τομέα .got.plt που αντιστοιχεί στο εξωτερικό σύμβολο (Ο τομέας .got.plt αρχικοποιείται σε όλες του τις εγγραφές με την διεύθυνση της πρώτης εντολής του τομέα .plt. Κατά την πρώτη κλήση ενός συμβόλου, ο τομέας .got.plt καλεί τις πρώτες εντολές του τομέα .plt, οι οποίες είναι υπεύθυνες για την εύρεση της υλοποίησης του συμβόλου και για την ενημέρωση της αντίστοιχης εγγραφής στον τομέα .got.plt. Για τον λόγο αυτό, στα πλαίσια της στατικής ανάλυσης δεν θα ασχοληθούμε περαιτέρω με το περιεχόμενο του τομέα .got.plt).

Οι εγγραφές του τομέα .rela.plt αντιστοιχούν σε 24 bytes η κάθε μία, με τα 4 πρώτα bytes να περιέχουν την διεύθυνση του συμβόλου στον τομέα .got.plt και το δωδέκατο (12) byte έως και το δέκατο έκτο (16) byte περιέχουν τον αύξον αριθμό εγγραφής του τομέα .dynsym και του τομέα .gnu.version που αντιστοιχούν στον σύμβολο.

Οι εγγραφές του τομέα .dynsym είναι μεγέθους 24 bytes. Τα πρώτα 4 bytes αποτελούν το offset από την αρχή του τομέα .dynstr που αντιστοιχεί στο όνομα του συμβόλου.

Οι εγγραφές του τομέα .gnu.version είναι μεγέθους 2 bytes, όπου η κάθε εγγραφή αντιστοιχεί σε μία εγγραφή του τομέα .dynsym, συσχετισμένες κατά αύξον αριθμό και περιέχει την τιμή του πεδίου value μίας εγγραφής του τομέα .gnu.version_r.

Οι εγγραφές του τομέα .gnu.version_r είναι μεγέθους 16 bytes, μεταξύ του έκτου(6) και όγδοου(8) byte βρίσκεται η τιμή του πεδίου value της εγγραφής και μεταξύ του όγδοου(8) και δωδέκατου(12) byte το offset από την αρχή του τομέα .dynstr που αντιστοιχεί στο όνομα της βιβλιοθήκης.

Όλες οι απαραίτητες πληροφορίες του συμβόλου βρίσκονται στους παραπάνω τομείς, αρκεί να γίνει η σωστή αντιστοίχιση των εγγραφών που ανήκουν στο ίδιο σύμβολο.

4.3.3 Αλγόριθμος αρχιτεκτονικής intel x86 συνάρτησης scan_function_for_external()

Η πρώτη εντολή αφαίρεσης (*sub)* που βρίσκεται από την αρχή της συνάρτησης, χρησιμοποιείται για την δέσμευση χώρου στο stack.

Οι εντολές μεταφοράς δεδομένων (*mov)* γράφουν στη μνήμη που δείχνει το άθροισμα του πρώτου καταχωρητή και του offset της εντολής, την τιμή που περιέχει ο δεύτερος καταχωρητής της εντολής. Αν ο πρώτος καταχωρητής είναι ο *ebp* (Stack Base Pointer), σημαίνει πως γίνεται αποθήκευση στο stack. γράφουν στη στοίβα κάποια μεταβλητή, είτε εσωτερική είτε εξωτερική. Βρίσκοντας όλα τα σημεία της στοίβας όπου έχει γίνει εγγραφή μεταβλητών, μπορούμε να υπολογίσουμε το μέγεθος της κάθε μεταβλητής.

Οι εντολές κλήσης (*call)* χρησιμοποιούνται για την κλήση εσωτερικών ή εξωτερικών συναρτήσεων. Αν η διεύθυνση της εντολής ανήκει στον τομέα .plt, τότε πρόκειται για εξωτερικό σύμβολο.

Η πρώτη εντολή που βρίσκεται στην διεύθυνση του τομέα .plt της εντολής κλήσης συνάρτησης, είναι εντολή μετάβασης ροής του προγράμματος *jmp*, η οποία είναι μεγέθους 6 bytes. Η διεύθυνση της αμέσως επόμενης εντολής του τομέα .plt είναι η διεύθυνση της εντολής κλήσης συνάρτησης αυξημένη κατά το μέγεθος της εντολής μετάβασης ροής του προγράμματος *jmp* (+6).

Οι εγγραφές του τομέα .got.plt αντιστοιχούν σε 4 bytes η κάθε μία, τα οποία περιέχουν την διεύθυνση της δεύτερης εντολής ενός συμβόλου του τομέα .plt (διεύθυνση που καλείται στην εντολή κλήσης (*call*) + 6).

Οι εγγραφές του τομέα .rel.plt αντιστοιχούν σε 8 bytes η κάθε μία, με τα 4 πρώτα bytes να περιέχουν την διεύθυνση του συμβόλου στον τομέα .got και τα 4 υπόλοιπα περιέχουν τον αύξον αριθμό εγγραφής του τομέα .dynsym και του τομέα .gnu.version που αντιστοιχούν στον σύμβολο.

Οι εγγραφές του τομέα .dynsym είναι μεγέθους 16 bytes. Τα πρώτα 4 bytes αποτελούν το offset από την αρχή του τομέα .dynstr που αντιστοιχεί στο όνομα του συμβόλου.

Οι εγγραφές του τομέα .gnu.version είναι μεγέθους 2 bytes, όπου η κάθε εγγραφή αντιστοιχεί σε μία εγγραφή του τομέα .dynsym, συσχετισμένες κατά αύξον αριθμό και περιέχει την τιμή του πεδίου value μίας εγγραφής του τομέα .gnu.version_r.

Οι εγγραφές του τομέα .gnu.version_r είναι μεγέθους 16 bytes, μεταξύ του έκτου(6) και όγδοου(8) byte βρίσκεται η τιμή του πεδίου value της εγγραφής και μεταξύ του όγδοου(8) και δωδέκατου(12) byte το offset από την αρχή του τομέα .dynstr που αντιστοιχεί στο όνομα της βιβλιοθήκης.

Όλες οι απαραίτητες πληροφορίες του συμβόλου βρίσκονται στους παραπάνω τομείς, αρκεί να γίνει η σωστή αντιστοίχιση των εγγραφών που ανήκουν στο ίδιο σύμβολο.

4.3.4 Αλγόριθμος αρχιτεκτονικής intel x64 συνάρτησης scan_function_for_external()

Η πρώτη εντολή αφαίρεσης (*sub)* που βρίσκεται από την αρχή της συνάρτησης, χρησιμοποιείται για την δέσμευση χώρου στο stack.

Οι εντολές μεταφοράς δεδομένων (*mov)* γράφουν στη μνήμη που δείχνει το άθροισμα του πρώτου καταχωρητή και του offset της εντολής, την τιμή που περιέχει ο δεύτερος καταχωρητής της εντολής. Αν ο πρώτος καταχωρητής είναι ο *rbp* (Stack Base Pointer), σημαίνει πως γίνεται αποθήκευση στο stack. γράφουν στη στοίβα κάποια μεταβλητή, είτε εσωτερική είτε εξωτερική. Βρίσκοντας όλα τα σημεία της στοίβας όπου έχει γίνει εγγραφή μεταβλητών, μπορούμε να υπολογίσουμε το μέγεθος της κάθε μεταβλητής.

Οι εντολές κλήσης (*call)* χρησιμοποιούνται για την κλήση εσωτερικών ή εξωτερικών συναρτήσεων. Αν η διεύθυνση της εντολής ανήκει στον τομέα .plt, τότε πρόκειται για εξωτερικό σύμβολο.

Η πρώτη εντολή που βρίσκεται στην διεύθυνση του τομέα .plt της εντολής κλήσης συνάρτησης, είναι εντολή μετάβασης ροής του προγράμματος *jmp*, η οποία είναι μεγέθους 6 bytes. Η διεύθυνση της αμέσως επόμενης εντολής του τομέα .plt είναι η διεύθυνση της εντολής κλήσης συνάρτησης αυξημένη κατά το μέγεθος της εντολής μετάβασης ροής του προγράμματος *jmp* (+6).

Οι εγγραφές του τομέα .got.plt αντιστοιχούν σε 8 bytes η κάθε μία, με τα 4 πρώτα bytes να περιέχουν την διεύθυνση της δεύτερης εντολής ενός συμβόλου του τομέα .plt.

Οι εγγραφές του τομέα .rela.plt αντιστοιχούν σε 24 bytes η κάθε μία, με τα 4 πρώτα bytes να περιέχουν την διεύθυνση του συμβόλου στον τομέα .got.plt και το δωδέκατο (12) byte έως και το δέκατο έκτο (16) byte να περιέχουν τον αύξον αριθμό εγγραφής του τομέα .dynsym και του τομέα .gnu.version που αντιστοιχούν στον σύμβολο.

Οι εγγραφές του τομέα .dynsym είναι μεγέθους 24 bytes. Τα πρώτα 4 bytes αποτελούν το offset από την αρχή του τομέα .dynstr που αντιστοιχεί στο όνομα του συμβόλου.

Οι εγγραφές του τομέα .gnu.version είναι μεγέθους 2 bytes, όπου η κάθε εγγραφή αντιστοιχεί σε μία εγγραφή του τομέα .dynsym, συσχετισμένες κατά αύξον αριθμό και περιέχει την τιμή του πεδίου value μίας εγγραφής του τομέα .gnu.version_r.

Οι εγγραφές του τομέα .gnu.version_r είναι μεγέθους 16 bytes, μεταξύ του έκτου(6) και όγδοου(8) byte βρίσκεται η τιμή του πεδίου value της εγγραφής και μεταξύ του όγδοου(8) και δωδέκατου(12) byte το offset από την αρχή του τομέα .dynstr που αντιστοιχεί στο όνομα της βιβλιοθήκης.

Όλες οι απαραίτητες πληροφορίες του συμβόλου βρίσκονται στους παραπάνω τομείς, αρκεί να γίνει η σωστή αντιστοίχιση των εγγραφών που ανήκουν στο ίδιο σύμβολο.

4.4 Απαλοιφή Αρχιτεκτονικών

Στα πλαίσια της υλοποίησης δημιουργήθηκαν διαφορετικά αρχεία για κάθε αρχιτεκτονική (carrion_arm.py, carrion_aarch64.py, carrion_i386.py, carrion_64.py), τα οποία απαρτίζονται από το ίδιο interface, όσον αφορά τις ονομασίες κλάσεων και συναρτήσεων, τις λειτουργίες αυτών, των ορισμάτων και των αποτελεσμάτων επιστροφής, με στόχο την απαλοιφή των διαφορών μεταξύ των αρχιτεκτονικών. Διαφοροποιούνται ως προς τον αλγόριθμο που ακολουθείται σε κάθε περίπτωση για την υλοποίηση της εκάστοτε λειτουργίας.

class Carrion*_arm32/*Carrion*_aarch64/*Carrion*_i386/*Carrion*_64*:

***__init__()*:** Αρχικοποίηση του αντικειμένου της βιβλιοθήκης archinfo [31], το οποίο περιέχει τις πληροφορίες της αντίστοιχης αρχιτεκτονικής. Τέτοιες πληροφορίες, όπως είναι τα ονόματα των καταχωρητών και οι δείκτες που αντιστοιχούν σε αυτούς, χρησιμοποιούνται από τη γλώσσα ενδιάμεσης αναπαράστασης.

***archinfo()*:** Επιστρέφει το αντικείμενο της βιβλιοθήκης archinfo [31] που αρχικοποιήθηκε στην συνάρτηση __init__().

***get_n_argument_offset(n)*:** Επιστρέφει τον δείκτη του καταχωρητή που χρησιμοποιείτε ως *n* όρισμα σε μία κλήση, σύμφωνα με την βιβλιοθήκη archinfo [31]. Δέχεται ως όρισμα τον αριθμό n.

***frame_pointer()*:** Επιστρέφει ένα dictionary με όλα τα πιθανά ονόματα του καταχωρητή που χρησιμοποιείται ως frame pointer ως key και με τους αντίστοιχους δείκτες που περιέχονται στην βιβλιοθήκη archinfo ως το αντίστοιχο value.

***stack_pointer()*:** Επιστρέφει ένα dictionary με όλα τα πιθανά ονόματα του καταχωρητή που χρησιμοποιείται ως stack pointer ως key και με τους αντίστοιχους δείκτες που περιέχονται στην βιβλιοθήκη archinfo ως το αντίστοιχο value.

***get_cmpl_flag()*:** Επιστρέφει την τιμή που έχει ο καταχωρητής cc_op της ενδιάμεσης αναπαράστασης, όταν πρόκειται για πράξη σύγκρισης.

***find_n_argument_fp_offset(bfd, bytestream, irsb, stmt_addr, block_num, n)*:** Δέχεται ως ορίσματα ένα binary file descriptor (bfd) της βιβλιοθήκης pybfd, το αντίστοιχο byte stream (bytestream) του bfd, μία λίστα από simple blocks της ενδιάμεσης αναπαράστασης pyvex (irsb), όπου περιέχουν την ενδιάμεση αναπαράσταση της υλοποίησης μίας συνάρτησης, ο αύξον αριθμός μέσα στο block της εντολής που βρίσκεται η κλήση εξωτερικής συνάρτησης (stmt_addr), ο αύξον αριθμός του block στη λίστα *irsb* που βρίσκεται η κλήση (block_num) και ο αύξον αριθμός (n) του ορίσματος της κλήσης. Επιστρέφει το offset σε μορφή ακεραίου από τον frame pointer (fp) από το οποίο μεταφέρθηκε η τιμή του στο *n* όρισμα, -1 σε περίπτωση μη εύρεσης του.

***scan_function_for_external(bfd, bytestream, fsym)*:** βλ. 4.3 Ανάλυση Συμβόλων και Συναρτήσεων

4.4.1 Αλγόριθμος αρχιτεκτονικών arm32, aarch64, intel x64 συνάρτησης find_n_argument_fp_offset()

Οι αρχιτεκτονικές arm32, aarch64 και intel x64 χρησιμοποιούν συγκεκριμένους καταχωρητές για την μετάβαση ορισμάτων κατά την κλήση συναρτήσεων. Κάθε τέτοιος καταχωρητής είναι δεσμευμένος για συγκεκριμένη θέση ορίσματος. Για παράδειγμα, στην αρχιτεκτονική arm32, για οποιαδήποτε κλήση συνάρτησης με ορίσματα, το πρώτο όρισμα θα αποθηκευτεί στον καταχωρητή r0, το δεύτερο στον καταχωρητή r1, το τρίτο στον r2 κλπ. Οι ίδιοι καταχωρητές θα χρησιμοποιηθούν σε κάθε περίπτωση κλήσης συνάρτησης με ορίσματα. Η εγγραφή των καταχωρητών αυτών στην γλώσσα ενδιάμεσης αναπαράστασης γίνεται με την εντολή Put.

Για κάθε εντολή Put που βρίσκεται, ελέγχεται ο καταχωρητής στον οποίο γίνεται εγγραφή. Γίνεται χρήση της συνάρτησης *get_n_argument_offset(),* η οποία επιστρέφει τον δείκτη που αντιστοιχεί στον καταχωρητή που χρησιμοποιείται για το *n* όρισμα. Αν αυτός ο δείκτης είναι ίδιος με το offset της εντολής Put, αυτό σημαίνει ότι γράφει στον καταχωρητή που αντιστοιχεί στο όρισμα *n.*

Ελέγχονται η τιμή που γράφτηκε στον καταχωρητή.

Αν η πηγή αποτελείται από μία μεταβλητή, τότε γίνεται χρήση της συνάρτησης value_traceback(), η οποία επιστρέφει την τιμή που έχει πάρει η μεταβλητή αυτή.

Αν η πηγή αποτελείται από άθροισμα δύο μεταβλητών, τότε η δεύτερη αποτελεί το offset της πρώτης μεταβλητής.

Γίνεται έλεγχος για τον εντοπισμό της τιμής του offset.

Ελέγχεται αν η πρώτη μεταβλητή είναι ο καταχωρητής frame pointer.

Αν είναι, τότε βρέθηκε το offset, αλλιώς συνεχίζεται η αναζήτηση στην επόμενη εντολή Put.

Αν δεν βρεθεί το offset, επιστρέφεται η τιμή -1.

Στην αρχιτεκτονική intel x64, γίνεται η χρήση τεχνικής υπερχείλισης με την εντολή Add64. Προστίθεται ένα big_offset, αρκετά μεγάλο ώστε να προκαλέσει υπερχείλιση κατά την διάρκεια της πρόσθεσης, όπου το αποτέλεσμα της οποίας είναι ο ζητούμενος δείκτης. Η εύρεση του πραγματικού offset από τον αρχικό δείκτη, προκύπτει από την μαθηματική πράξη: offset = 0xffffffffffffffff - big_offset + 1

4.4.2 Αλγόριθμος αρχιτεκτονικών intel x86 συνάρτησης find_n_argument_fp_offset()

Η αρχιτεκτονική intel x86, χρησιμοποιεί το stack για την μεταφορά των ορισμάτων, γράφοντάς τα σε αυτό ακριβώς πριν την κλήση της εξωτερικής συνάρτησης. Υπολογίζει και γράφει πρώτα το τελευταίο όρισμα και τέλος το πρώτο, ξεκινώντας την εγγραφή σε χαμηλό μέρος του stack, φτάνοντας στο πάνω μέρος του. Η εγγραφή στο stack στην γλώσσα ενδιάμεσης αναπαράστασης γίνεται με την χρήση της εντολής Store. Αυτό διαφοροποιεί τον αλγόριθμο ως προς τον τρόπο που εντοπίζονται τα ορίσματα. Ο αλγόριθμος πλέον διαμορφώνεται ως εξής:

Αναζητείται η εγγραφή του n ορίσματος. Για την εύρεση της εγγραφής του n ορίσματος, αρκεί να βρεθεί η n εντολή store, αναζητώντας ανάποδα από το σημείο κλήσης. Αυτό συμβαίνει, γιατί καθώς ελέγχουμε τις εντολές ενδιάμεσης αναπαράστασης προς τα πίσω από το σημείο που έγινε η κλήση, το πρώτο όρισμα που συναντάμε να γράφεται στο stack είναι και το πρώτο όρισμα της συνάρτησης, αντίστοιχα το δεύτερο κλπ. (αφού κατά την κανονική ροή του προγράμματος, πρώτα γράφεται το τελευταίο όρισμα στο stack, και τελευταίο το πρώτο όρισμα). Αν για παράδειγμα αναζητούμε το δεύτερο όρισμα μίας κλήσης (n = 2), στην πρώτη εντολή Store στο stack που θα συναντήσουμε ελέγχοντας τις εντολές προς τα πίσω, θα είναι το πρώτο όρισμα. Η εντολή αυτή δεν θα αναλυθεί περεταίρω. Στην επόμενη εντολή Store στο stack που θα συναντήσουμε, θα πρέπει να αναλυθεί, μιας και αναφέρεται στο ζητούμενο όρισμα (δεύτερο).

Ελέγχεται η τιμή που γράφτηκε στο stack.

Αν η πηγή αποτελείται από άθροισμα δύο μεταβλητών, τότε η δεύτερη αποτελεί το offset της πρώτης μεταβλητής.

Γίνεται έλεγχος για τον εντοπισμό της τιμής του offset.

Αν η πρώτη μεταβλητή είναι ο καταχωρητής frame pointer, τότε βρέθηκε το offset.

Αν δεν βρεθεί το offset, επιστρέφεται η τιμή -1.

Επίσης, γίνεται η χρήση τεχνικής υπερχείλισης με την εντολή Add32. Προστίθεται ένα big_offset, αρκετά μεγάλο ώστε να προκαλέσει υπερχείλιση κατά την διάρκεια της πρόσθεσης, όπου το αποτέλεσμα της οποίας είναι ο ζητούμενος δείκτης. Η εύρεση του πραγματικού offset από τον αρχικό δείκτη, προκύπτει από την μαθηματική πράξη: offset = 0xffffffff - big_offset + 1

4.5 Ανάλυση κώδικα

Η ανάλυση του πηγαίου κώδικα γίνεται με την χρήση της ενδιάμεσης αναπαράστασης pyvex. Κάποιες από τις συναρτήσεις που υλοποιούν απλές και βασικές λειτουργίες, δεν εξαρτώνται από άλλα αρχεία του εργαλείου και βρίσκονται στο αρχείο carrion_base.py. Οι συναρτήσεις αυτές θα μπορούσαν να

χρησιμοποιηθούν και εξωτερικά του εργαλείου, κάνοντας χρήση μόνο το συγκεκριμένο αρχείο σε άλλη εφαρμογή. Αναλυτικά:

***getInt(bytestream, start, end, step=1)*:** Δέχεται ως ορίσματα ένα byte stream (bytestream), τον αύξον αριθμό του πρώτου (start) και του τελευταίου (end) byte ενός αριθμού που εμπεριέχεται μέσα στο byte stream και προορατικά το βήμα (step) με το οποίο θα γίνει η μετατροπή (αρχικοποίηση στο 1). Επιστρέφει τον ακέραιο αριθμό που βρίσκεται στις θέσεις byte start έως end του bytestream.

***hasNumbers(inputString)*:** Δέχεται ως όρισμα ένα string (inputString) και επιστρέφει boolean True σε περίπτωση που το inputString περιέχει αριθμούς, αλλιώς επιστρέφει boolean False.

***load_value_from_offset(bytestream, offset)*:** Δέχεται ως ορίσματα ένα byte stream (bytestream) και ένα offset από την αρχή του byte stream και επιστρέφει την τιμή που αντιστοιχεί σε Dword (double word = 32bits) από το σημείο του offset με την χρήση της συνάρτησης getInt(), σε αλφαριθμητική μορφή δεκαεξαδικής αξίας. Το offset μπορεί να είναι αλφαριθμητικού ή αριθμητικού τύπου μεταβλητή.

***load_value_from_address(bfd, bytestream, address)*:** Δέχεται ως ορίσματα ένα binary file descriptor (bfd), ένα byte stream (bytestream) και μία εικονική διεύθυνση (address) και επιστρέφει την τιμή που αντιστοιχεί σε Dword (double word = 32bits) από το σημείο της εικονικής διεύθυνσης address. Η εικονική διεύθυνση address μπορεί να είναι αλφαριθμητικού ή αριθμητικού τύπου μεταβλητή. Η μετατροπή της εικονικής διεύθυνσης σε offset του byte stream γίνεται με την βοήθεια του bfd. Αρχικά, βρίσκεται σε ποιον τομέα του bfd ανήκει η εικονική διεύθυνση address. Έπειτα, υπολογίζεται η διαφορά της εικονικής διεύθυνσης address με την εικονική διεύθυνση της αρχής του τομέα στην οποία ανήκει η διεύθυνση. Τέλος, δίνοντας ως παραμέτρους το bytestream και το άθροισμα του file_offset του τομέα και της διαφοράς που υπολογίστηκε προηγουμένως (bfd.sections[section_name].file_offset + (address - bfd.sections[section_name].vma)) στην συνάρτησης *load_value_from_offset(),* βρίσκεται η τιμή της Dword που αντιστοιχεί στην εικονική διεύθυνση address.

***fetch_target_register(stmt)*:** Δέχεται ως όρισμα μία εντολή ενδιάμεσης αναπαράστασης (stmt) της βιβλιοθήκης pyvex και επιστρέφει τον καταχωρητή στον οποίο θα διαπραχθεί μία ενέργεια (προορισμός της ενέργειας). Τέτοιες ενέργειες μπορεί να είναι μία πράξη πρόσθεσης (add), αφαίρεσης (sub), αποθήκευσης (store), φόρτωσης (load), ανάθεσης τιμής, κλπ. Για παράδειγμα στην εντολή *STle(t8) = t2,* θα επιστραφεί ο καταχωρητής t8. Εάν η εντολή δεν περιέχει πράξεις ανάθεσης τιμής, επιστρέφει την τιμή -1.

***fetch_source_register(stmt)*:** Δέχεται ως όρισμα μία εντολή ενδιάμεσης αναπαράστασης (stmt) της βιβλιοθήκης pyvex και επιστρέφει τον καταχωρητή από τον οποίο προέρχεται η τιμή που θα ανατεθεί (πηγή της ενέργειας). Τέτοιες ενέργειες μπορεί να είναι μία πράξη αποθήκευσης (store), φόρτωσης (load), ανάθεσης τιμής, κλπ. Σε περιπτώσεις πράξεων, όπως μία πρόσθεσης (add), αφαίρεσης (sub), κλπ. που η τιμή προέρχεται από τον συνδυασμό δύο ή περισσοτέρων καταχωρητών, θα επιστραφεί ένα tuple με τους καταχωρητές που χρησιμοποιήθηκαν. Για παράδειγμα στην εντολή *STle(t8) = t2,* θα επιστραφεί ο καταχωρητής t2, ενώ στην εντολή *t4 = Add32(t2,t3),* θα επιστραφεί το tuple [t2, t3]. Εάν η εντολή δεν περιέχει πράξεις ανάθεσης τιμής, επιστρέφει την τιμή -1.

***value_traceback(bfd, bytestream, irsb, stmt_addr, block_num, reg)*:** Δέχεται ως ορίσματα ένα binary file descriptor (bfd) της βιβλιοθήκης pybfd, το αντίστοιχο byte stream (bytestream) του bfd, μία λίστα από simple blocks της ενδιάμεσης αναπαράστασης pyvex (irsb), όπου περιέχουν την ενδιάμεση αναπαράσταση της υλοποίησης μίας συνάρτησης, ο αύξον αριθμός μέσα στο block της εντολής που βρίσκεται η κλήση εξωτερικής συνάρτησης (stmt_addr), ο αύξον αριθμός του block στη λίστα *irsb* που βρίσκεται η κλήση (block_num) και ο καταχωρητής (reg) του οποίου αναζητάτε η τιμή. Επιστρέφει την τιμή που έχει ο καταχωρητής reg στην εντολή που βρίσκεται στη θέση stmt_addr του block block_num. Αν δεν βρεθεί σταθερά τιμή, επιστρέφεται η τιμή -1.

Στο αρχείο carrion.py, υλοποιείται η βασική κλάση του εργαλείου (Carrion), η οποία περικλείει όλες απαραίτητες πληροφορίες και συναρτήσεις σχετικά με το αρχείο. Αναλυτικά:

class Carrion: Περιέχει όλες τις πληροφορίες της βιβλιοθήκης προς ανάλυση, καθώς και κάποιες συναρτήσεις εύρεσης ορισμένων από αυτών. Συγκεκριμένα:

- ένα binary file descriptor της βιβλιοθήκης pybfd (*bfd*),

- το αντίστοιχο byte stream του bfd (*bytestream*) ,
- ένα αντικείμενο της αντίστοιχης κλάσης (Carrion_arm, Carrion_aarch64, Carrion_i386, Carrion_64) της αρχιτεκτονικής της βιβλιοθήκης προς ανάλυση (*arch*) και
- μία λίστα με τα σύμβολα των συναρτήσεων και πληροφορίες αυτών, που υλοποιούνται μέσα στην βιβλιοθήκη, το κάθε ένα με την μορφή αντικειμένου της κλάσης FunctionSymbol (function_symbols).

__init__(bfd, bytestream): Δέχεται ως ορίσματα ένα binary file descriptor (bfd) και το αντίστοιχο byte stream (bytestream) του bfd της βιβλιοθήκης προς ανάλυση. Με την χρήση των πληροφοριών του *bfd* επιλέγεται η αντίστοιχη κλάση κλάσης (Carrion_arm, Carrion_aarch64, Carrion_i386, Carrion_64) του πεδίου *arch.* Ελέγχονται τα σύμβολα της βιβλιοθήκης και για όποια από αυτά υλοποιούνται μέσα στην ίδια την βιβλιοθήκη, δημιουργεί ένα αντικείμενο της κλάσης FunctionSymbol και σε συνδιασμό με την συνάρτηση scan_function_for_external του αντίστοιχου αντικειμένου *arch,* βρίσκονται όλες οι σχετικές με το σύμβολο πληροφορίες που είναι απαραίτητες για την ανάλυση της βιβλιοθήκης. Τα αντικείμενα της κλάσης FunctionSymbol, αποθηκεύονται στην λίστα function_symbols.

get_n_argument_offset(n)**:** Επιστρέφει τον δείκτη του καταχωρητή που χρησιμοποιείτε ως *n* όρισμα σε μία κλήση, χρησιμοποιώντας την αντίστοιχη συνάρτηση του αντικειμένου *arch*. Δέχεται ως όρισμα τον αριθμό n.

frame_pointer()**:** Επιστρέφει ένα dictionary με όλα τα πιθανά ονόματα του καταχωρητή που χρησιμοποιείται ως frame pointer ως key και με τους αντίστοιχους δείκτες που περιέχονται στην βιβλιοθήκη archinfo ως το αντίστοιχο value, , χρησιμοποιώντας την αντίστοιχη συνάρτηση του αντικειμένου *arch*.

stack_pointer()**:** Επιστρέφει ένα dictionary με όλα τα πιθανά ονόματα του καταχωρητή που χρησιμοποιείται ως stack pointer ως key και με τους αντίστοιχους δείκτες που περιέχονται στην βιβλιοθήκη archinfo ως το αντίστοιχο value, χρησιμοποιώντας την αντίστοιχη συνάρτηση του αντικειμένου *arch*.

is_frame_pointer(offset): Δέχεται ως όρισμα έναν δείκτη (offset) και επιστρέφει *boolean True* αν αντιστοιχεί στον δείκτη του frame pointer στην βιβλιοθήκη archinfo, αλλιώς επιστρέφει *boolean False*.

is_stack_pointer(offset): Δέχεται ως όρισμα έναν δείκτη (offset) και επιστρέφει *boolean True* αν αντιστοιχεί στον δείκτη του stack pointer στην βιβλιοθήκη archinfo, αλλιώς επιστρέφει *boolean False*.

get_cc_op_offset(): Επιστρέφει τον δείκτη του καταχωρητή *cc_op* στην βιβλιοθήκη archinfo.

get_cc_dep1_offset(): Επιστρέφει τον δείκτη του καταχωρητή *cc_dep1* στην βιβλιοθήκη archinfo.

get_cc_dep2_offset(): Επιστρέφει τον δείκτη του καταχωρητή *cc_dep2* στην βιβλιοθήκη archinfo.

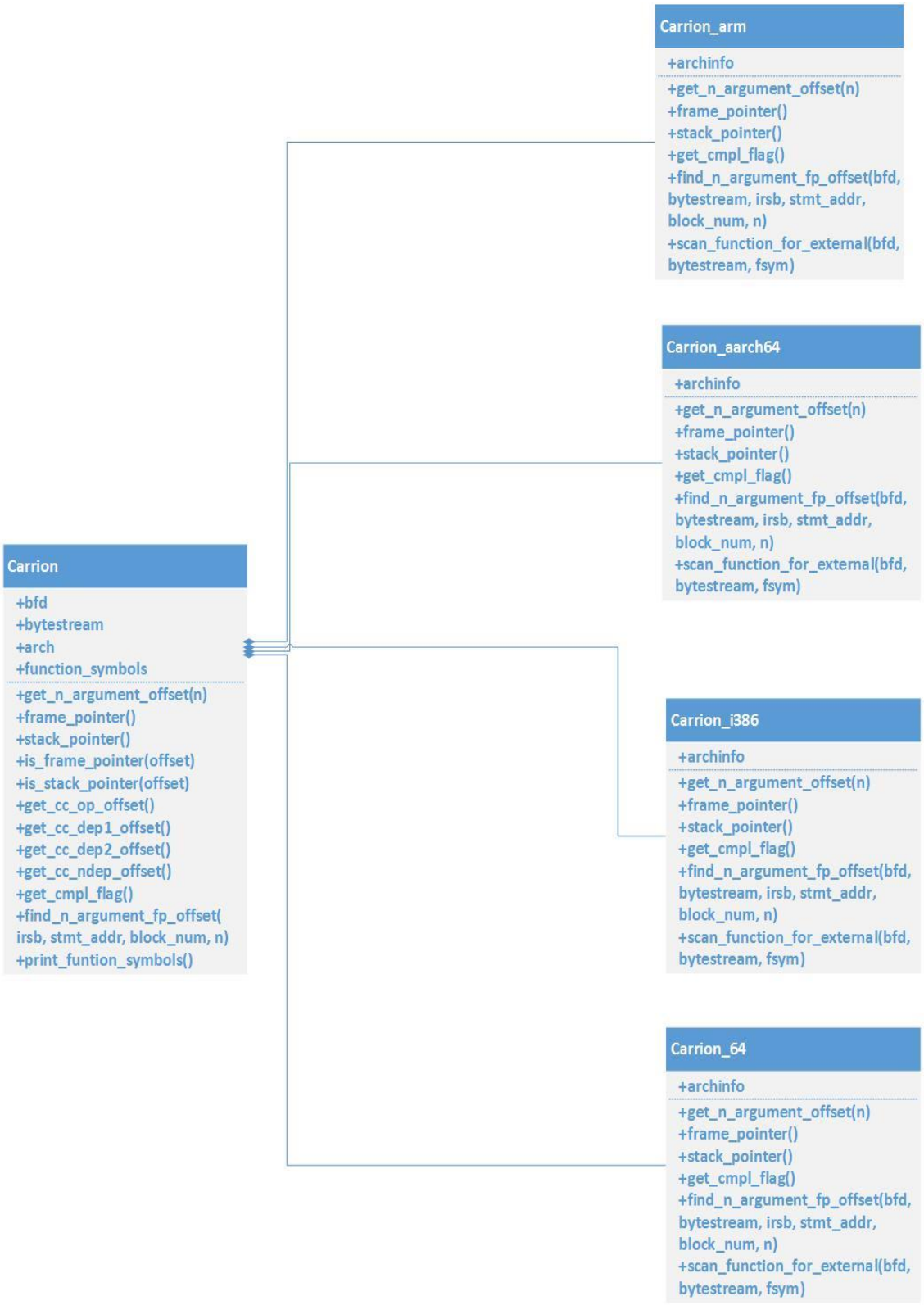
get_cc_ndep_offset(): Επιστρέφει τον δείκτη του καταχωρητή *cc_ndep* στην βιβλιοθήκη archinfo.

get_cmpl_flag(): Επιστρέφει την τιμή που έχει ο καταχωρητής cc_op της ενδιάμεσης αναπαράστασης, όταν πρόκειται για πράξη σύγκρισης, χρησιμοποιώντας την αντίστοιχη συνάρτηση του αντικειμένου *arch*.

find_n_argument_fp_offset(irsb, stmt_addr, block_num, n): Δέχεται ως ορίσματα μία λίστα από simple blocks της ενδιάμεσης αναπαράστασης pyvex (irsb), όπου περιέχουν την ενδιάμεση αναπαράσταση της υλοποίησης μίας συνάρτησης, ο αύξον αριθμός μέσα στο block της εντολής που βρίσκεται η κλήση εξωτερικής συνάρτησης (stmt_addr), ο αύξον αριθμός του block στη λίστα *irsb* που βρίσκεται η κλήση (block_num) και ο αύξον αριθμός (n) του ορίσματος της κλήσης. Χρησιμοποιεί την αντίστοιχη συνάρτηση του αντικειμένου *arch* και επιστρέφει το offset σε μορφή ακεραίου από τον frame pointer (fp) από το οποίο μεταφέρθηκε η τιμή του στο *n* όρισμα, -1 σε περίπτωση μη εύρεσης του.

print_function_symbols(): Εκτυπώνει όλα τα σύμβολα των συναρτήσεων που βρέθηκαν να υλοποιούνται εντός της βιβλιοθήκης προς ανάλυση καθώς και τις πληροφορίες αυτών. Συγκεκριμένα εκτυπώνει:

- το όνομα της συνάρτησης,
- τη διεύθυνση της συνάρτησης,
- τον τομέα υλοποίησης της συνάρτησης,
- το μέγεθος της συνάρτησης,
- το μέγεθος του stack της συνάρτησης,
- όλες τις εξωτερικές συναρτήσεις άλλων βιβλιοθηκών που βρέθηκαν να καλούνται μέσα στην συνάρτηση (όνομα εξωτερικής συνάρτησης και βιβλιοθήκη υλοποίησης), με όλες τις διευθύνσεις όπου έγινε η κλήση (εντός συνάρτησης) και
- ανάλυση του stack, με offset της κάθε μεταβλητής του από την αρχή του και το μέγεθος της



Εικόνα 4 Κλάσεις απαλοιφής αρχιτεκτονικών

Τέλος, στο αρχείο carrion.py, υλοποιούνται οι πιο σύνθετες συναρτήσεις, οι οποίες εξαρτώνται από συναρτήσεις όλων των προηγούμενων αρχείων που έχουν αναφερθεί:

traceback(carrion, irsb, stmt_addr, block_num, reg): Δέχεται ως ορίσματα ένα αντικείμενο της κλάσσης Carrion (*carrion*), μία λίστα από simple blocks της ενδιάμεσης αναπαράστασης pyvex (irsb), όπου περιέχουν την ενδιάμεση αναπαράσταση της υλοποίησης μίας συνάρτησης, ο αύξον αριθμός μέσα στο block της εντολής που βρίσκεται η κλήση εξωτερικής συνάρτησης (stmt_addr), ο αύξον αριθμός του block στη λίστα *irsb* που βρίσκεται η κλήση (block_num) και ο καταχωρητής (*reg)* του οποίου αναζητάτε η τιμή. Επιστρέφει την τιμή του καταχωρητή *reg* σε μορφή ακεραίου ή τον ίδιο τον καταχωρητή *reg* σε περίπτωση μη εύρεσης τιμής.

trace_rodata(carrion, irsb, stmt_addr, block_num, reg): Δέχεται ως ορίσματα ένα αντικείμενο της κλάσσης Carrion (*carrion*), μία λίστα από simple blocks της ενδιάμεσης αναπαράστασης pyvex (irsb), όπου περιέχουν την ενδιάμεση αναπαράσταση της υλοποίησης μίας συνάρτησης, ο αύξον αριθμός μέσα στο block της εντολής που βρίσκεται η κλήση εξωτερικής συνάρτησης (stmt_addr), ο αύξον αριθμός του block στη λίστα *irsb* που βρίσκεται η κλήση (block_num) και μία μεταβλητή (reg*),* η οποία μπορεί να είναι ένας καταχωρητής ή μία διεύθυνση προς έλεγχο. λεγχθεί για το αν η τιμή του προέρχεται από τον τομέα .rodata ή έχει σταθερή τιμή . Επιστρέφει boolean True αν η τιμή του καταχωρητή *reg* προέρχεται από τον τομέα .rodata, αν η τιμή αυτή είναι σταθερά, ή αν η μεταβλητή reg περιέχει διεύθυνση του τομέα .rodata, αλλιώς boolean False.

get_fp_offset(carrion, irsb, stmt_addr, block_num, reg): Δέχεται ως ορίσματα ένα αντικείμενο της κλάσσης Carrion (*carrion*), μία λίστα από simple blocks της ενδιάμεσης αναπαράστασης pyvex (irsb), όπου περιέχουν την ενδιάμεση αναπαράσταση της υλοποίησης μίας συνάρτησης, ο αύξον αριθμός μέσα στο block της εντολής που βρίσκεται η κλήση εξωτερικής συνάρτησης (stmt_addr), ο αύξον αριθμός του block στη λίστα *irsb* που βρίσκεται η κλήση (block_num) και ο καταχωρητής (*reg),* όπου αναζητείται το offset από τον frame pointer από όπου προήλθε η τιμή του. Επιστρέφει το offset από τον frame pointer του καταχωρητή *reg* σε μορφή ακεραίου ή τον ίδιο τον καταχωρητή *reg* σε περίπτωση μη εύρεσης τιμής του offset.

is_external_argument_by_fp_offset(carrion, irsb, offset): Δέχεται ως ορίσματα ένα αντικείμενο της κλάσσης Carrion (carrion), μία λίστα από simple blocks της ενδιάμεσης αναπαράστασης pyvex (irsb) και ένα offset από τον frame pointer μιας μεταβλητής του stack (offset) και επιστρέφει boolean True όταν η μεταβλητή αυτή έχει πάρει την τιμή της από εξωτερικό όρισμα και όταν το offset που δόθηκε είναι -1 ή boolean False αν είναι σταθερά.

is_static_string(carrion, irsb, offset): Δέχεται ως ορίσματα ένα αντικείμενο της κλάσσης Carrion (carrion), μία λίστα από simple blocks της ενδιάμεσης αναπαράστασης pyvex (irsb) και ένα offset από τον frame pointer μιας μεταβλητής αλφαριθμητικού τύπου του stack (offset) και επιστρέφει boolean True όταν η μεταβλητή αυτή έχει σταθερά τιμή ή έχει πάρει την τιμή της από τον τομέα .rodata ή boolean False αν προέρχεται από εξωτερικό όρισμα.

check_results(carrion, irsb, stmt_addr, block_num): Δέχεται ως ορίσματα ένα αντικείμενο της κλάσσης Carrion (*carrion*), μία λίστα από simple blocks της ενδιάμεσης αναπαράστασης pyvex (irsb), όπου περιέχουν την ενδιάμεση αναπαράσταση της υλοποίησης μίας συνάρτησης, ο αύξον αριθμός μέσα στο block της εντολής που βρίσκεται η κλήση εξωτερικής συνάρτησης (stmt_addr) και ο αύξον αριθμός του block στη λίστα *irsb* που βρίσκεται η κλήση (block_num). Επιστρέφει boolean True αν γίνεται έλεγχος των αποτελεσμάτων που ενδεχομένος να επέστρεψε μία συνάρτηση που κλήθηκε στο σημείο stmt_addr ή boolean False εάν τα αποτελέσματα δεν ελέγχονται.

print_results(results, fsym, ex_func, ex_func_addr): Δέχεται ως ορίσματα μία λίστα αποτελεσμάτων μίας συνάρτησης ελέγχου σεναρίου (results), ένα αντικείμενο της κλάσης FunctionSymbol (fsym) της εσωτερικής συνάρτησης που αναλύθηκε, το όνομα της εξωτερικής συνάρτησης που ελέγχθηκε (ex_func) και την διεύθυνση αυτής (ex_func_addr) και εκτυπώνει τα ονόματα και τις διευθύνσεις των συναρτήσεων που ελέγχθηκαν, πρώτα τις προειδοποιήσεις και μετά τις ευπάθειες που βρίσκονται μέσα στη λίστα αποτελεσμάτων.

4.5.1 Αλγόριθμος συνάρτησης value_traceback()

Αναζητούνται οι εντολές στις οποίες επηρεάζεται η τιμή του καταχωρητή reg. Οι εντολές αυτές χωρίζονται σε τρεις περιπτώσεις.

Πρώτη περίπτωση είναι η εντολή να είναι τύπου Load.

Τότε, γίνεται ξανά η χρήση της ίδιας της συνάρτησης value_traceback(), αναζητώντας αυτή τη φορά, τη διεύθυνση από την οποία θα γίνει φόρτωση τιμής (load). Αφού βρεθεί η τιμή φόρτωσης, με την χρήση της συνάρτησης *load_value_from_address()* βρίσκεται η τιμή του καταχωρητή reg.

Δεύτερη περίπτωση είναι η εντολή να είναι τύπου μαθηματικής πράξης, στην οποία θα έχουμε τουλάχιστον δύο καταχωρητές/σταθερές από όπου προέρχεται η τιμή.

Πάλι, γίνεται ξανά η χρήση της ίδιας της συνάρτησης value_traceback(), αναζητώντας την αξία όλων των καταχωρητών που χρησιμοποιούνται στην εντολή μαθηματικής πράξης.

Σε περιπτώσεις όπου κάποιος καταχωρητής δεν προέρχεται από κάποια σταθερά τιμή, αλλά πρόκειται για καταχωρητή όπου παίρνει την τιμή του από κάποιο εξωτερικό όρισμα ή είναι καταχωρητής stack/frame pointer, η συνάρτηση τερματίζει και επιστρέφεται η τιμή -1, γιατί ο αρχικός καταχωρητής reg προς αναζήτηση, εμπεριέχει είτε κάποιον stack/frame pointer μετακινημένο κατά κάποια τιμή, η οποία δεν μπορεί να υπολογιστεί, είτε κάποιο από εξωτερικό όρισμα, του οποίου η τιμή είναι και πάλι άγνωστη.

Αν όλοι οι καταχωρητές προέλευσης έχουν σταθερές τιμές, τότε υπολογίζεται και επιστρέφεται το αποτέλεσμα της πράξης.

Τρίτη και τελευταία περίπτωση, είναι η εντολές ανάθεσης τιμής. Σε αυτές τις περιπτώσεις ανατήθετε στον αρχικό καταχωρητή reg, ο καταχωρητής προέλευσης, δηλαδή ο καταχωρητής που προκύπτει από την χρήση της συνάρτησης *fetch_source_register().* Συνεχίζοντας την αναζήτηση, πλέον ψάχνουμε για την τιμή του καταχωρητή προέλευσης, η οποία τιμή ανατήθετε στον αρχικό καταχωρητή.

Η αναζήτηση συνεχίζεται έως ότου φτάσει στην αρχή της συνάρτησης.

4.5.2 Αλγόριθμος συνάρτησης traceback()

Αναζητούνται οι εντολές στις οποίες επηρεάζεται η τιμή του καταχωρητή reg. Οι εντολές αυτές χωρίζονται σε τρεις περιπτώσεις.

Πρώτη περίπτωση είναι η εντολή να είναι τύπου Load.

Τότε, γίνεται ξανά η χρήση της ίδιας της συνάρτησης traceback(), αναζητώντας αυτή τη φορά, τη διεύθυνση από την οποία θα γίνει φόρτωση τιμής (load). Αφού βρεθεί η τιμή φόρτωσης, με την χρήση της συνάρτησης *load_value_from_address()* βρίσκεται η τιμή του καταχωρητή reg.

Δεύτερη περίπτωση είναι η εντολή να είναι τύπου μαθηματικής πράξης, στην οποία θα έχουμε τουλάχιστον δύο καταχωρητές/σταθερές από όπου προέρχεται η τιμή.

Πάλι, γίνεται ξανά η χρήση της ίδιας της συνάρτησης value_traceback(), αναζητώντας την αξία όλων των καταχωρητών που χρησιμοποιούνται στην εντολή μαθηματικής πράξης.

Σε περιπτώσεις όπου κάποιος καταχωρητής δεν περιέχει κάποια σταθερά τιμή, αν είναι καταχωρητής stack/frame pointer, επιστρέφεται ο καταχωρητής αυτός. Αν πρόκειται για καταχωρητή όπου παίρνει την τιμή του από κάποιο εξωτερικό όρισμα, ο καταχωρητής αυτός παίρνει την τιμή 0.

Υπολογίζεται και επιστρέφεται το αποτέλεσμα της πράξης.

Τρίτη και τελευταία περίπτωση, είναι η εντολές ανάθεσης τιμής. Σε αυτές τις περιπτώσεις ανατήθετε στον αρχικό καταχωρητή reg, ο καταχωρητής προέλευσης, δηλαδή ο καταχωρητής που προκύπτει από την χρήση της συνάρτησης *fetch_source_register().* Συνεχίζοντας την αναζήτηση, πλέον ψάχνουμε για την τιμή του καταχωρητή προέλευσης, η οποία τιμή ανατήθετε στον αρχικό καταχωρητή.

Η αναζήτηση συνεχίζεται έως ότου φτάσει στην αρχή της συνάρτησης.

4.5.3 Αλγόριθμος συνάρτησης trace_rodata()

Ελέγχεται η ύπαρξη του τομέα .rodata. Αν δεν βρεθεί ο τομέας, επιστρέφεται η τιμή False.

Αν η μεταβλητή reg είναι ακαίρεος και η τιμή της ανήκει ως διεύθυνση εντός του τομέα .rodata, επιστρέφεται True.

Αναζητούνται οι εντολές στις οποίες επηρεάζεται η τιμή του καταχωρητή reg.

Αν η εντολή είναι τύπου Load, γίνεται η χρήση της συνάρτησης traceback(), αναζητώντας την διεύθυνση από την οποία θα γίνει φόρτωση τιμής (load). Αφού βρεθεί η διεύθυνση φόρτωσης, επιστρέφεται True αν ανήκει ως διεύθυνση εντός του τομέα .rodata.

Αν η εντολή είναι ανάθεσης τιμής, ανατήθετε στον αρχικό καταχωρητή reg, ο καταχωρητής προέλευσης, δηλαδή ο καταχωρητής που προκύπτει από την χρήση της συνάρτησης *fetch_source_register().* Συνεχίζοντας την αναζήτηση, πλέον ψάχνουμε για την τιμή του καταχωρητή προέλευσης, η οποία τιμή ανατήθετε στον αρχικό καταχωρητή.

Η αναζήτηση συνεχίζεται έως ότου φτάσει στην αρχή της συνάρτησης.

Αν στο τέλος της αναζήτησης ο καταχωρητής περιέχει κάποια σταθερά τιμή, επιστρέφεται True.

Εάν η συνάρτηση δεν έχει επιστρέψει κάποια τιμή μέχρι το τέλος της αναζήτησης, αυτό σημαίνει πως δεν βρέθηκε περίπτωση ανάθεσης σταθερής τιμής στην μεταβλητή reg. Άρα επιστρέφεται False.

4.5.4 Αλγόριθμος συνάρτησης get_fp_offset()

Αν ο κατχωρητής reg είναι ο frame pointer ή δεν υπάρχουν άλλες εντολές προς αναζήτηση, επιστρέφεται ο καταχωρητής reg. Η επιστροφή του ίδιου του καταχωρητή σε περίπτωση που είναι ο frame pointer ή σε περίπτωση που δεν υπάρχουν άλλες εντολές προς αναζήτηση, εξυπηρετεί την αναδρομή που χρησιμοποιείται σε επόμενα βήματα.

Αναζητούνται οι εντολές στις οποίες επηρεάζεται η τιμή του καταχωρητή reg.

Αν η εντολή είναι ανάθεσης τιμής και αν ο καταχωρητής προέλευσης είναι ο frame pointer, τότε επιστρέφεται ο frame pointer, αλλιώς ανατήθετε στον αρχικό καταχωρητή reg, ο καταχωρητής προέλευσης, που προκύπτει από την χρήση της συνάρτησης fetch_source_register(). Συνεχίζοντας την αναζήτηση, πλέον ψάχνουμε για την τιμή του καταχωρητή προέλευσης, η οποία τιμή ανατήθετε στον αρχικό καταχωρητή.

Αν η εντολή να είναι τύπου μαθηματικής πράξης με δύο καταχωρητές/σταθερές από όπου προέρχεται η τιμή, τότε χρησιμοποιείται η ίδια συνάρτηση get_fp_offset(), για την εύρεση των τιμών των καταχωρητών αυτών.

Αν ένας από τους δύο καταχωρητές είναι ο frame pointer, τότε ο δεύτερος είναι το ζητούμενο offset.

Αν η μαθηματική πράξη είναι Add32 ή Add64, κατά τις οποίες προστίθεται ένα big_offset, αρκετά μεγάλο ώστε να προκαλέσει υπερχείλιση κατά την διάρκεια της πρόσθεσης, όπου το αποτέλεσμα της οποίας είναι ο ζητούμενος δείκτης, τότε το πραγματικό offset από τον αρχικό δείκτη, προκύπτει από την μαθηματική πράξη: offset = 0xffffffff - big_offset + 1 για Add32 και offset = 0xffffffffffffffff - big_offset + 1 για Add64, το οποίο και επιστρέφεται σε μορφή ακεραίου

Αν κανένας από τους καταχωρητές δεν είναι ο frame pointer, ανατίθεται η τιμή 0 σε όποιον καταχωρητή είναι ο stack pointer και επιστρέφεται το αποτέλεσμα της πράξης.

Η αναζήτηση συνεχίζεται έως ότου φτάσει στην αρχή της συνάρτησης.

Αν έχει βρεθεί τιμή, η τιμή αυτή επιστρέφεται, αλλιώς επιστρέφεται ο τελευταίος καταχωρητής που βρέθηκε κατά την αναζήτηση ανάθεσης τιμών στον αρχικό καταχωρητή.

4.5.5 Αλγόριθμος συνάρτησης is_external_argument_by_fp_offset()

Αναζητούνται οι εντολές ανάθεσης τιμής ενδιάμεσης αναπαράστασης από την αρχή της συνάρτησης, όπου ο καταχωρητής προέλευσης είναι ο frame pointer. Στην ενδιάμεση αναπαράσταση pyvex πολλές λειτουργίες δεν γίνονται απευθείας με τους αρχικούς καταχωρητές (στην συγκεκριμένη περίπτωση με τον frame pointer), αλλά γίνεται με ανάθεση τιμών σε προσωρινούς καταχωρητές.

Αν βρεθεί ο frame pointer, ελέγχονται οι επόμενες εντολές, αναζητώντας μαθηματική πράξη στην οποία ο πρώτος καταχωρητής είναι ο προσωρινός καταχωρητής (ή ο ίδιος ο frame pointer) που του έχει ανατεθεί ο frame pointer και βρέθηκε στο προηγούμενο βήμα.

Η εντολή μαθηματικής πράξης με πρώτο καταχωρητή τον frame pointer (ή κάποιον προσωρινό καταχωρητή που περιέχει τον frame pointer), χρησιμοποιείται για την δημιουργία δείκτη της μεταβλητής του stack που βρίσκεται στο offset της πράξης αυτής. Το offset βρίσκεται με την χρήση της συνάρτησης traceback().

Με την χρήση της εντολής get_fp_offset() βρίσκεται το ενδεχόμενο offset του προσωρινού καταχωρητή από τον frame pointer (συνήθως 0).

Ελέγχεται αν η πράξη είναι Add32 ή Add64. Στις πράξεις αυτές προστίθεται ένα big_offset, αρκετά μεγάλο ώστε να προκαλέσει υπερχείλιση κατά την διάρκεια της πρόσθεσης, όπου το αποτέλεσμα της οποίας είναι ο ζητούμενος δείκτης, τότε το πραγματικό offset από τον αρχικό δείκτη, προκύπτει από την μαθηματική πράξη: offset = 0xffffffff - big_offset + 1 για πράξη Add32 και offset = 0xffffffffffffffff - big_offset + 1 για πράξη Add64. Το τελικό offset από τον frame pointer προκύπτει με την αφαίρεση του ενδεχόμενου offset του προσωρινού καταχωρητή από τον frame pointer (offset = 0xffffffff - temp_offset - big_offset + 1 για πράξη Add32 και offset = 0xffffffffffffffff - temp_offset - big_offset + 1 για πράξη Add64).

Αν βρεθεί εντολή ανάθεσης τιμής με καταχωρητή προέλευσης τον προσωρινό καταχωρητή, τότε η αναζήτηση συνεχίζεται ψάχνοντας πλέον για τον καταχωρητή προορισμού της εντολής αυτής.

Αν το τελικό offset από τον frame pointer που βρέθηκε αντιστοιχεί στο offset που δόθηκε ως όρισμα, αναζητούνται εντολές ανάθεσης τιμής από το σημείο της πράξης που δημιουργεί τον δείκτη της μεταβλητής του stack με το offset αυτό.

Αν βρεθεί εντολή ανάθεσης τιμής με καταχωρητή προέλευσης τον δείκτη της μεταβλητής του stack με το offset του προηγούμενου βήματος, η αναζήτηση συνεχίζεται ψάχνοντας πλέον τον καταχωρητή προορισμού της πράξης αυτής.

Αναζητούνται εντολές αποθήκευσης STle. Οι εντολές αποθήκευσης STle αποθηκεύουν την τιμή του καταχωρητή προέλευσης στο σημείο που δείχνει ο καταχωρητής προορισμού (ο καταχωρητής προορισμού χρησιμοποιείται ως δείκτης σε κάποιο σημείο της μνήμης, στην προκειμένη περίπτωση σε θέση μνήμης του stack).

Αν βρεθεί εντολή αποθήκευσης STle με καταχωρητή προορισμού τον δείκτη της μεταβλητής, ελέγχετε η τιμή που θα αποθηκευτεί με την χρήση της συνάρτησης traceback() με όρισμα τον καταχωρητή προέλευσης.

Αν η συνάρτηση traceback() επιστρέψει ακέραιο, σημαίνει πως ο καταχωρητής προέλευσης περιέχει κάποια σταθερή τιμή, η οποία δεν προέρχεται από εξωτερικό όρισμα, οπότε και επιστρέφεται False. Στην περίπτωση που επιστραφεί αλφαριθμητική τιμή, αυτή θα είναι το όνομα κάποιου καταχωρητή (διαφορετικό ανά αρχιτεκτονική) που χρησιμοποιείται για την ανάγνωση εξωτερικών ορισμάτων. Αυτό σημαίνει πως η τιμή που γράφεται στο stack με την χρήση της εντολής STle προέρχεται από εξωτερικό όρισμα, άρα επιστρέφεται True.

Αν το τελικό offset δεν αντιστοιχεί στο offset που δόθηκε ως όρισμα, συνεχίζεται η αναζήτηση με τον ίδιο τρόπο έως ότου βρεθεί το κατάλληλο offset. Αν η αναζήτηση φτάσει στο τέλος της συνάρτησης και δεν έχει βρεθεί το ζητούμενο offset, τότε δεν έχει γραφτεί κάτι από εξωτερικά ορίσματα στο offset του stack που δόθηκε ως όρισμα, άρα επιστρέφεται False.

4.5.6 Αλγόριθμος συνάρτησης is_static_string()

Αναζητούνται οι εντολές αποθήκευσης. Αν το offset από τον frame pointer του καταχωρητή προορισμού της εντολής αποθήκευσης είναι ίδιο με το offset που δόθηκε ως όρισμα, σημαίνει πως γράφεται κάτι στην μεταβλητή του stack που ζητήθηκε.

Με την χρήση της συνάρτησης trace_rodata() γίνεται έλεγχος της προέλευσης της τιμής που γράφεται στην μεταβλητή του stack που ζητήθηκε.

Επιστρέφεται True στην περίπτωση που η συνάρτηση trace_rodata() επιστρέψει True.

Εάν δεν βρεθεί καμία εντολή αποθήκευσης στην μεταβλητή του stack που ζητήθηκε όπου να αποθηκεύει σταθερά τιμή, επιστρέφεται False.

4.5.7 Αλγόριθμος συνάρτησης check_results()

Αναζητείται η εντολή αποθήκευσης STle μέσα στις επόμενες 5 εντολές από το σημείο κλήσης stmt_addr, με την οποία αποθηκεύονται τα αποτελέσματα μιας συνάρτησης στο stack. Αν δεν βρεθεί, σημαίνει πως τα αποτελέσματα αυτά δεν αποθηκεύονται κάπου στο stack, άρα δεν γίνεται και κάποιος αντίστοιχος έλεγχος, οπότε και επιστρέφεται False.

Υπολογίζεται το offset από τον frame pointer του καταχωρητή προορισμού της εντολής αποθήκευσης. Με αυτό τον τρόπο αναγνωρίζεται το σημείο στο stack στο οποίο έγινε εγγραφή των αποτελεσμάτων.

Ο έλεγχος των αποτελεσμάτων θα γίνει με κάποια πράξη σύγκρισης. Η πράξη σύγκρισης στην γλώσσα ενδιάμεσης αναπαράστασης pyvex υλοποιείται με την χρήση τριών εντολών τύπου Put.

Η πρώτη εντολή Put γράφει στον καταχωρητή cc_op μία σταθερά τιμή, διαφορετική ανά αρχιτεκτονική.

Η δεύτερη και η τρίτη εντολή Put γράφουν στους καταχωρητές cc_dep1 και cc_dep2 αντίστοιχα, τις δύο τιμές προς σύγκριση.

Οπότε, αν ο καταχωρητής cc_op δέχεται την τιμή σύγκρισης και κάποιος από τους καταχωρητές cc_dep1 ή cc_dep2 δέχεται την τιμή του από το stack όπου γράφτηκαν τα αποπτελέσματα της συνάρτησης, τότε τα αποτελέσματα αυτά ελέγχονται. Άρα επιστρέφεται True.

Αν δεν βρεθεί καμία διαδικασία σύγκρισης επιστρέφεται False.

4.6 Βοηθητικές Συναρτήσεις

Στα πλαίσια της ανάλυσης αρχείων βιβλιοθηκών σε δυαδική μορφή και της υλοποίησης του εργαλείου εύρεσης ευπαθειών, δημιουργήθηκαν κάποιες βοηθητικές συναρτήσεις για περεταίρω λεπτομερή ανάλυση. Οι συναρτήσεις αυτές υλοποιούνται στο αρχείο carrion_info.py και στόχος τους είναι η διευκόλυνση της κατανόησης της δομής των αρχείων προς ανάλυση και η εξαγωγή πληροφοριών από αυτά. Πιο συγκεκριμένα:

***getInt(bytestream, start, end, step=1)*:** Δέχεται ως ορίσματα ένα byte stream (bytestream), τον αύξον αριθμό του πρώτου (start) και του τελευταίου (end) byte ενός αριθμού που εμπεριέχεται μέσα στο byte stream και προορατικά το βήμα (step) με το οποίο θα γίνει η μετατροπή (αρχικοποίηση στο 1). Επιστρέφει τον ακέραιο αριθμό που βρίσκεται στις θέσεις byte start έως end του bytestream.

***hasNumbers(inputString)*:** Δέχεται ως όρισμα ένα string (inputString) και επιστρέφει boolean True σε περίπτωση που το inputString περιέχει αριθμούς, αλλιώς επιστρέφει boolean False.

***dumpFileSymbols(bfd)*:** Δέχεται ως όρισμα ένα binary file descriptor (bfd) και εκτυπώνει όλα τα σύμβολα που εμπεριέχονται σε αυτό.

***disassembleSection(bfd, section_name)*:** Δέχεται ως ορίσματα ένα binary file descriptor (bfd) και ένα όνομα τομέα (section_name) και εκτυπώνει τις εντολές assembly που προκύπτουν από το περιεχόμενο του τομέα με όνομα section_name στο bfd αρχείο.

***disassembleSections(bfd)*:** Δέχεται ως όρισμα ένα binary file descriptor (bfd) και εκτυπώνει τις εντολές assembly που προκύπτουν από το περιεχόμενο όσων τομέων εμπεριέχουν κώδικα στο bfd αρχείο.

***dumpSectionInfosFromContent(section)*:** Δέχεται ως όρισμα ένα τομέα (section) της βιβλιοθήκης bfd και εκτυπώνει το όνομα, τον αύξον αριθμό και το μέγεθος του τομέα.

***dumpSectionInfos(bfd, section_name)*:** Δέχεται ως ορίσματα ένα binary file descriptor (bfd) και ένα όνομα τομέα (section_name) και εκτυπώνει το όνομα, τον αύξον αριθμό και το μέγεθος του τομέα.

***dumpSectionsInfos(bfd)*:** Δέχεται ως όρισμα ένα binary file descriptor (bfd) και εκτυπώνει το όνομα, τον αύξον αριθμό και το μέγεθος όλων των τομέων του bfd με την χρήση του dumpSectionInfosFromContent().

***printDWord(bfd, bytestream, offset, size, entries_num)*:** Δέχεται ως ορίσματα ένα binary file descriptor (bfd), ένα byte stream (bytestream), ένα offset από την αρχή του byte stream (αντιπροσωπεύει

το σημείο του byte stream από το οποίο ξεκινάει η εκτύπωση), το συνολικό μέγεθος των bytes προς εκτύπωση από το offset και μετά και έναν αριθμό (entries_num) κομματιών διαχώρισης της εκτύπωσης για περιπτώσεις ομαδοποιήσεων[1]. Εκτυπώνει DWords (double word = 32bits) σε δεκαεξαδική μορφή byte από το offset του byte stream, για μέγεθος size, ομαδοποιημένα σε entries_num ομάδες.

***printRawSection(bfd, bytestream, section_name)*:** Δέχεται ως ορίσματα ένα binary file descriptor (bfd), το αντίστοιχο byte stream (bytestream) του bfd και ένα όνομα τομέα. Εκτυπώνει DWords (double word = 32bits) σε δεκαεξαδική μορφή byte του τομέα section_name, χωρισμένα σε ομάδες με βάση το με βάση το μέγεθος εγγραφής που περιέχουν οι πληροφορίες του τομέα (bfd.sections[section_name].entry_size).

***printRawSectionByEntries(bfd, bytestream, section_name, entries_num)*:** Δέχεται ως ορίσματα ένα binary file descriptor (bfd), το αντίστοιχο byte stream (bytestream) του bfd, ένα όνομα τομέα (section_name) και έναν αριθμό (entries_num) κομματιών διαχώρισης της εκτύπωσης για περιπτώσεις ομαδοποιήσεων. Εκτυπώνει DWords (double word = 32bits) σε δεκαεξαδική μορφή byte του τομέα section_name, ομαδοποιημένα σε entries_num ομάδες.

printRawSectionByEntrySize(bfd, bytestream, section_name, entry_size): Δέχεται ως ορίσματα ένα binary file descriptor (bfd), το αντίστοιχο byte stream (bytestream) του bfd, ένα όνομα τομέα (section_name) και το πλήθος των DWords της κάθε εγγραφής (entry_size). Εκτυπώνει DWords (double word = 32bits) σε δεκαεξαδική μορφή byte του τομέα section_name, ομαδοποιημένα σε ομάδες μεγέθους entry_size.

***printStringsFromSection(bfd, bytestream, section_name)*:** Δέχεται ως ορίσματα ένα binary file descriptor (bfd), το αντίστοιχο byte stream (bytestream) του bfd και ένα όνομα τομέα (section_name). Εκτυπώνει τα bytes του τομέα σε μορφή αλφαριθμητικής τιμής (string), διαχωρισμένα ανά NUL terminator (\0).

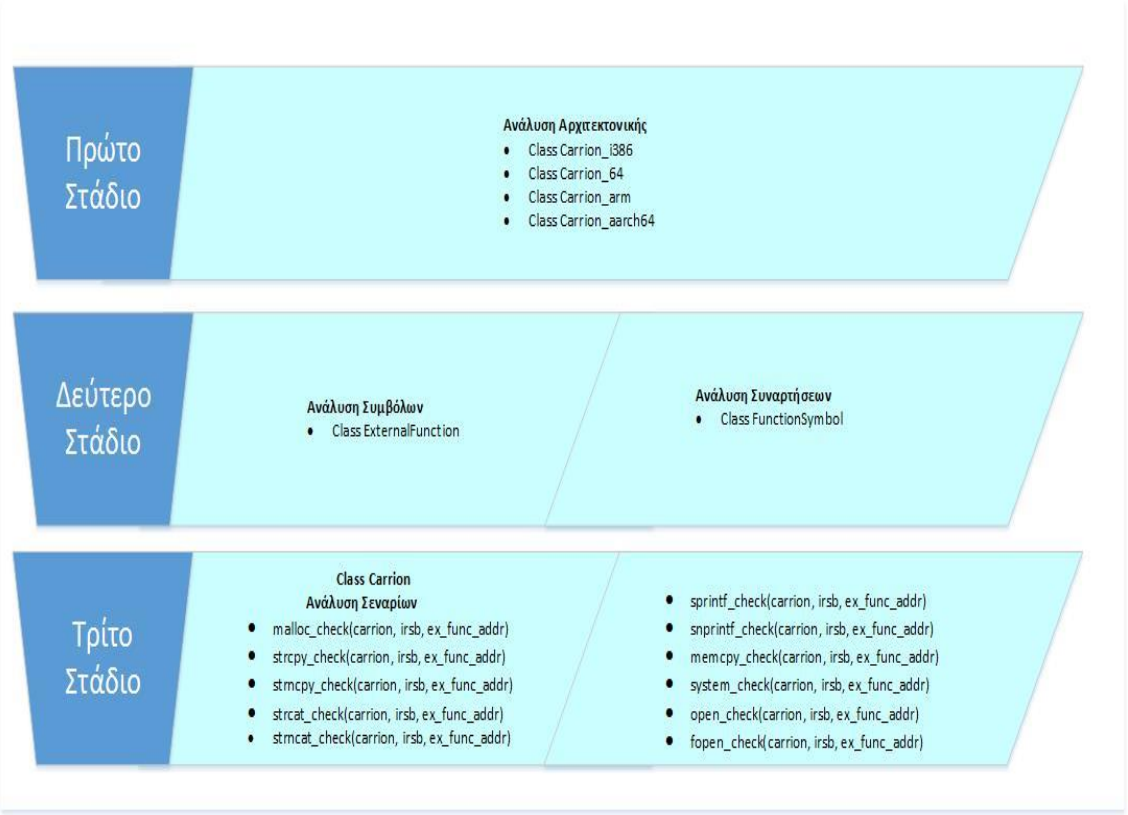
***printStringsFromAddress(bytestream, offset, size)*:** Δέχεται ως ορίσματα ένα byte stream (bytestream), ένα offset από την αρχή του byte stream (αντιπροσωπεύει το σημείο του byte stream από το οποίο ξεκινάει η εκτύπωση) και το συνολικό μέγεθος των bytes προς εκτύπωση από το offset και μετά. Εκτυπώνει τα bytes από το offset του byte stream, για μέγεθος size, σε μορφή αλφαριθμητικής τιμής (string), διαχωρισμένα ανά NUL terminator (\0).

4.7 Σενάρια

Ο έλεγχος των σεναρίων υλοποιείται με μία ξεχωριστή συνάρτηση ανά σενάριο, με σκοπό την εύκολη αναβάθμιση του κάθε σεναρίου. Όλες οι συναρτήσεις υλοποίησης των σεναρίων δέχονται ως ορίσματα ένα αντικείμενο της κλάσσης Carrion (*carrion*), μία λίστα από simple blocks της ενδιάμεσης αναπαράστασης pyvex (irsb), όπου περιέχουν την ενδιάμεση αναπαράσταση της υλοποίησης μίας συνάρτησης και την διεύθυνση της συνάρτησης (ex_func_addr) και επιστρέφουν μία λίστα με μηνύματα προειδοποιήσεων ή εύρεσης ευπαθειών. Για την υλοποίηση των ελέγχων, έτσι όπως περιγράφονται στο κεφάλαιο 3.6, γίνεται συνδυαστική χρήση των συναρτήσεων *find_n_argument_fp_offset()* για την εύρεση του offset από τον frame pointer του *n* ορίσματος της συνάρτησης προς έλεγχο, *is_external_argument_by_fp_offset()* για έλεγχο αν στο offset που βρέθηκε με την συνάρτηση *find_n_argument_fp_offset()* περιέχεται τιμή εξωτερικής μεταβλητής, *is_static_string()* για έλεγχο αν στο offset που βρέθηκε με την συνάρτηση *find_n_argument_fp_offset()* περιέχεται σταθερή αλφαριθμητική τιμή και *check_results()* για τον αν γίνεται έλεγχος των αποτελεσμάτων που επέστρεψε μία συνάρτηση. Οι συναρτήσεις που προκύπτουν είναι οι εξής:

- ***malloc_check(carrion, irsb, ex_func_addr)***
- ***strcpy_check(carrion, irsb, ex_func_addr)***
- ***strncpy_check(carrion, irsb, ex_func_addr)***
- ***strcat_check(carrion, irsb, ex_func_addr)***
- ***strncat_check(carrion, irsb, ex_func_addr)***
- ***sprintf_check(carrion, irsb, ex_func_addr)***
- ***snprintf_check(carrion, irsb, ex_func_addr)***

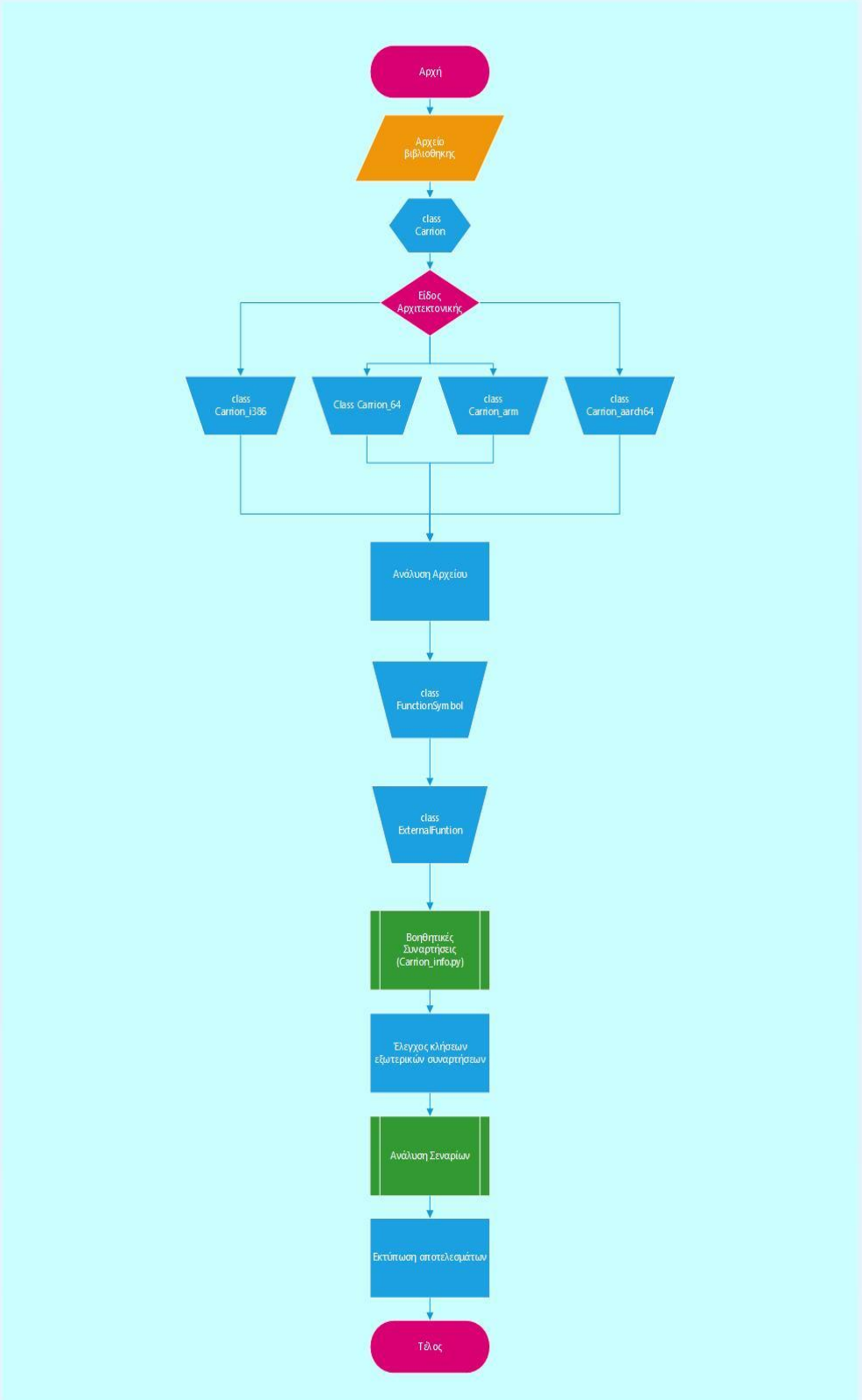
- ***memcpy_check(carrion, irsb, ex_func_addr)***
- ***system_check(carrion, irsb, ex_func_addr)***
- ***open_check(carrion, irsb, ex_func_addr)***
- ***fopen_check(carrion, irsb, ex_func_addr)***



Εικόνα 5 Υλοποίηση σταδίων ανάλυσης

4.8 Κύρια Ροή

Η κύρια ροή του εργαλείου βρίσκεται στο αρχείο carrion.py. Αρχικά, δημιουργεί ένα αντικείμενο της κλάσης pybfd με βάση το αρχείο προς ανάλυση και ένα byte stream αυτού. Με την χρήση αυτών των δύο καλείται οποιαδήποτε βοηθητική συνάρτηση εξόρυξης πληροφοριών του αρχείου carrion_info.py (βλ. 4.6 Βοηθητικές Συναρτήσεις), καθώς επίσης, δημιουργείται αντικείμενο της κύριας κλάσης Carrion (βλ. 4.5 Ανάλυση πηγαίου κώδικα). Ελέγχοντας την λίστα με τα σύμβολα του αντικειμένου της κλάσης Carrion, σε περίπτωση εύρεσης εξωτερικής συνάρτησης που προέρχεται από την βιβλιοθήκη libc και για την οποία υπάρχει συνάρτηση ελέγχου σεναρίων, τότε καλείται η αντίστοιχη συνάρτηση και εκτυπώνονται τα αποτελέσματα της.


Αρχή
Αρχείο βιβλιοθηκης
class Carrion
Είδος Αρχιτεκτονικής
class Carrion_i386
Class Carrion_64
class Carrion_arm
class Carrion_aarch64
Ανάλυση Αρχείου
class FunctionSymbol
class ExternalFuntion
Βοηθητικές Συναρτήσεις (Carrion_info.py)
Έλεγχος κλήσεων εξωτερικών συναρτήσεων
Ανάλυση Σεναρίων
Εκτύπωση αποτελεσμάτων
Τέλος

5 Εγχειρίδιο Χρήστη

5.1 Εξαρτήσεις εξωτερικών βιβλιοθηκών

Το εργαλείο αναπτύχθηκε σε περιβάλλον Ubuntu 14.04.4 LTS, σε γλώσσα Python 2.7.6 [32]. Επιπλέον, εγκαταστάθηκαν τα πακέτα binutils-dev και python-dev. Χρησιμοποιήθηκαν οι βιβλιοθήκες PyBFD και PyVEX, όπου η λειτουργία τους θα αναλυθεί παρακάτω. Για την χρήση της βιβλιοθήκης PyBFD απαιτείται η εγκατάσταση του πακέτου binutils-multiarch.

5.1.1 Βιβλιοθήκη PyBFD

Η βιβλιοθήκη PyBFD είναι μια διασύνδεση της βιβλιοθήκης GNU Binary File Descriptor (BFD) στην γλώσσα Python. Αποτελεί ένα περιτύλιγμα γύρω από τις λειτουργίες χαμηλού επιπέδου που παρέχουν τα GNU binutils libopcodes και libbfd. Επιτρέπει στο χρήστη να χειριστεί όλες τις υποστηριζόμενες αρχιτεκτονικές και μορφές αρχείων που υποστηρίζουν τα binutils εργαλεία.

Η βιβλιοθήκη GNU Binary File Descriptor (BFD) αποτελεί τον βασικό μηχανισμό χειρισμού ποικίλων μορφών δυαδικών αρχείων και αρχιτεκτονικών. Το εργαλείο χρησιμοποιεί την βιβλιοθήκη PyBFD για την εύρεση βασικών πληροφοριών του αρχείου απαραίτητες για την διαδικασία της ανάλυσης.

5.1.2 Βιβλιοθήκη PyVEX

Η βιβλιοθήκη PyVEX είναι μια διασύνδεση της βιβλιοθήκης VEX στην γλώσσα Python. Η βιβλιοθήκη VEX αποτελεί την υλοποίηση μιας ενδιάμεσης αναπαράστασης γλωσσών χαμηλού επιπέδου μεταξύ διαφορετικών αρχιτεκτονικών. Με την χρήση μιας ενδιάμεσης αναπαράστασης, απαλείφονται οι διαφορές μεταξύ των αρχιτεκτονικών, επιτρέποντας μία ενιαία ανάλυση για όλες τις αρχιτεκτονικές.

Ο έλεγχος σεναρίων του εργαλείου αποτελεί ανάλυση της ενδιάμεσης αναπαράστασης που προκύπτει από το εκάστοτε αρχείο.

5.2 Εγκατάσταση

Σε περιβάλλον Ubuntu 14.04.4 LTS (ή μεταγενέστερο), εγκαθιστούμε πρώτα τα απαραίτητα πακέτα binutils-dev και python-dev.

```
$ sudo apt-get install binutils-dev
$ sudo apt-get install python-dev
```

Έπειτα, εγκαθιστούμε το πακέτο binutils-multiarch, το οποίο απαιτείται για την εγκατάσταση της βιβλιοθήκης PyBFD.

```
$ sudo apt-get install binutils-multiarch
```

Για την εγκατάσταση της βιβλιοθήκης PyBFD προτείνονται διάφοροι τρόποι με την χρήση διαφορετικών εργαλείων. Συνιστάται η χρήση του εργαλείου git, όπου και δοκιμάστηκε με επιτυχία. Η εγκατάστασή του γίνεται με την παρακάτω εντολή:

```
$ sudo apt-get install git
```

Η εγκατάσταση της βιβλιοθήκης PyBFD έγινε με της εξής εντολές:

```
$ git clone https://github.com/Groundworkstech/pybfd.git
$ cd pybfd
$ python ./setup.py install
```

Τέλος, για την εγκατάσταση της βιβλιοθήκης PyVEX χρησιμοποιήθηκε το εργαλείο pip.

```
$ sudo apt-get install python-pip
$ pip install pyvex
```

Λόγο της χρήσης του εργαλείου pip, ενδέχεται να παρουσιαστεί κάποιο πρόβλημα κατά την φόρτωση δυναμικής βιβλιοθήκης.

```
ImportError: ERROR: fail to load the dynamic library.
```

Η λύση του προβλήματος είναι η αντιγραφή του αρχείου της βιβλιοθήκης libcapstone.so στον σωστό φάκελο, με την παρακάτω εντολή:

```
$ cp /usr/local/lib/python2.7/dist-packages/usr/lib/python2.7/dist-packages/capstone/libcapstone.so
/usr/local/lib/python2.7/dist-packages/capstone/
```

5.3 Ορίσματα χρήσης

Το εργαλείο που αναπτύχθηκε, πέρα από την ανάλυση σεναρίων αρχείων βιβλιοθηκών, περιέχει και μερικές λειτουργίες στατικής ανάλυσης. Για απλή ανάλυση σεναρίων το μόνο που χρειάζεται ως όρισμα, είναι το αρχείο της βιβλιοθήκης προς ανάλυση:

```
$ python carrion.py [library_file]
```

5.3.1 Λειτουργίες ανάλυσης

Όπως αναφέρθηκε, το εργαλείο διαθέτει κάποιες λειτουργίες στατικής ανάλυσης. Με την παράμετρο *-h* (ή *--help*) εκτυπώνονται όλες οι παραμέτροι και οι λειτουργίες τους:

```
$    python carrion.py -h
usage: carrion.py [-h] [-f] [-d DISAS_SECTION] [--disas_all] [-i SECTION_INFO]
                  [--all_sections_info] [-b BYTES] [-e ENTRIES]
                  [-s ENTRY_SIZE] [-S SECTION_SIZE] [-c STRINGS] [-E] [-v]
                  filename

positional arguments:
  filename                file to be analyzed

optional arguments:
  -h, --help              show this help message and exit
  -f, --file_symbols      dump file symbols
  -d DISAS_SECTION, --disas_section DISAS_SECTION
                          disassemble section(s) seperated with commas(,)
  --disas_all             disassemble all sections
  -i SECTION_INFO, --section_info SECTION_INFO
                          print infos of section(s) seperated with commas(,)
  --all_sections_info     print infos of all sections
  -b BYTES, --bytes BYTES
                          print raw bytes of section(s)/address(es) seperated
                          with commas(,)
  -e ENTRIES, --entries ENTRIES
                          number of entries of section(s) seperated with
                          commas(,) - for raw bytes of a section
```

-s ENTRY_SIZE, --entry_size ENTRY_SIZE	
	number of DWords(32bits) of entry of section(s)
	seperated with commas(,) - for raw bytes of a section
-S SECTION_SIZE, --section_size SECTION_SIZE	
	size of section(s) seperated with commas(,) in bytes –
	for print strings/raw bytes from address of a section
-c STRINGS, --strings STRINGS	
	print strings of section(s)/address(es) seperated with
	commas(,)
-E, --export_functions	
	print functions and their addresses found in the file
-v, --vex_ir	vex ir translate

Παράμετροι:

- ***-h, --help:*** Εκτυπώνει το κείμενο επεξήγησης των παραμέτρων.
- ***-f, --file_symbols:*** Εκτυπώνει τα σύμβολα του αρχείου (διεύθυνση, όνομα τομέα, όνομα συμβόλου, αξία συμβόλου).
- ***-d DISAS_SECTION, --disas_section DISAS_SECTION:*** Εκτυπώνει σε μορφή assembly τον/τους τομέα/τομείς που δόθηκαν (*DISAS_SECTION*). Περισσότεροι από ένας τομέας μπορούν να δοθούν ως όρισμα, οι οποίοι διαχωρίζονται με κόμμα (,).
- ***--disas_all:*** Εκτυπώνει σε μορφή assembly όλους τους τομείς.
- ***-i SECTION_INFO, --section_info SECTION_INFO:*** Εκτυπώνει τις πληροφορίες του/των τομέα/τομέων που δόθηκαν (*SECTION_INFO*). Περισσότεροι από ένας τομέας μπορούν να δοθούν ως όρισμα, οι οποίοι διαχωρίζονται με κόμμα (,).
- ***--all_sections_info:*** Εκτυπώνει τις πληροφορίες όλων των τομέων.
- ***-b BYTES, --bytes BYTES:*** Εκτυπώνει τα bytes του/των τομέα/τομέων ή της/των διεύθυνσης/διευθύνσεων που δόθηκαν (*BYTES*). Περισσότεροι από ένας τομέας ή μία διεύθυνση μπορούν να δοθούν ως όρισμα, ακόμα και ανάμικτα, αρκεί να διαχωρίζονται με κόμμα (,).
- ***-e ENTRIES, --entries ENTRIES:*** Λειτουργεί συνδυαστικά με την παράμετρο *-b/--bytes.* Αποτελεί τον αριθμό (*ENTRIES)* ομάδων όπου θα διαχωριστεί η εκτύπωση της παραμέτρου *-b/--bytes.* Εάν η παράμετρος *-b/--bytes* έχει περισσότερα από ένα ορίσματα, μπορούν να δοθούν τα αντίστοιχα μεγέθη ομάδων (χρησιμοποιώντας την αντίστοιχη σειρά), τα οποία διαχωρίζονται με κόμμα (,).
- ***-s ENTRY_SIZE, --entry_size ENTRY_SIZE:*** Λειτουργεί συνδυαστικά με την παράμετρο *-b/--bytes.* Αποτελεί τον αριθμό (*ENTRY_SIZE)* DWords(32bits) όπου αποτελείται μία ομάδα της εκτύπωση της παραμέτρου *-b/--bytes.* Εάν η παράμετρος *-b/--bytes* έχει περισσότερα από ένα ορίσματα, μπορούν να δοθούν τα αντίστοιχα μεγέθη ομάδων (χρησιμοποιώντας την αντίστοιχη σειρά), τα οποία διαχωρίζονται με κόμμα (,).
- ***-S SECTION_SIZE, --section_size SECTION_SIZE:*** Λειτουργεί συνδυαστικά με την παράμετρο *-b/--bytes* ή με την παράμετρο *-c/--strings.* Αποτελεί το μέγεθος ή τα μεγέθη (*SECTION_SIZE)* του/των τομέα/τομέων σε bytes. Εάν η παράμετρος *-b/--bytes (*ή η παράμετρος *-c/--strings)* έχει

περισσότερα από ένα ορίσματα, μπορούν να δοθούν τα αντίστοιχα μεγέθη (χρησιμοποιώντας την αντίστοιχη σειρά), τα οποία διαχωρίζονται με κόμμα (,).

- ***-c STRINGS, --strings STRINGS:*** Εκτυπώνει τον τομέα ή την διεύθυνση (*STRINGS*) σε αλφαριθμητική μορφή. Σε περιπτώσεις διευθύνσεων συνιστάται η χρήση της παραμέτρου *-S/--section_size*. Περισσότεροι από ένας τομέας ή μία διεύθυνση μπορούν να δοθούν ως όρισμα, ακόμα και ανάμικτα, αρκεί να διαχωρίζονται με κόμμα (,).
- ***-E, --export_functions:*** Εκτυπώνει τα ονόματα των συναρτήσεων που βρέθηκαν μέσα στο αρχείο, την διεύθυνση όπου βρίσκονται, το όνομα του τομέα που ανήκουν, το μέγεθος της συνάρτησης, το μέγεθος του stack, τα σημεία όπου γίνεται κλήση εξωτερικών συναρτήσεων (όνομα εξωτερικής συνάρτησης, όνομα βιβλιοθήκης όπου ανήκει και διεύθυνση όπου γίνεται η κλήση), καθώς επίσης και μία ανάλυση του stack.
- ***-v, --vex_ir:*** Εκτυπώνει την ενδιάμεση αναπαράσταση όλων των συναρτήσεων που βρέθηκαν μέσα στο αρχείο.

5.4 Παραδείγματα χρήσης

Στα πλαίσια ελέγχου του εργαλείου δημιουργήθηκαν δοκιμαστικές βιβλιοθήκες, στις αρχιτεκτονικές arm, aarch64, i386 και x64, οι οποίες περιείχαν περιπτώσεις σεναρίων. Για τα επόμενα παραδείγματα θα χρησιμοποιηθεί μία βιβλιοθήκη με δύο συναρτήσεις, μία που υλοποιεί την περίπτωση ευπάθειας ασφαλείας του σεναρίου της συνάρτησης malloc() της libc και μία όπου χρησιμοποιείται η ίδια συνάρτηση, χωρίς ευπάθεια ασφαλείας. Δίνοντας μόνο ως όρισμα την βιβλιοθήκη στο εργαλείο, θα παραχθεί το παρακάτω output:

```
$   python carrion.py libctestaarch64.so.1.0

[+] Creating BFD instance...
[+] File format    : Linker/assembler/compiler output.
[+] Architecture   : ARM AArch64 (87)
[+] BFD target name : elf64-littleaarch64
[+] Entry point    : 0x6B0
[+] Sections       : 22



ctest_malloc_vulnerable(0x804): malloc@GLIBC_2.17(0x814)

        Warning:          First argument of malloc is external

        Possible Vulnerability:   Returned results are not being checked
```

Χρησιμοποιώντας την παράμετρο -E εκτυπώνονται όλες οι πληροφορίες των συναρτήσεων που βρίσκονται στο αρχείο:

```
$    python carrion.py libctestaarch64.so.1.0 -E

[+] Creating BFD instance...
[+] File format     : Linker/assembler/compiler output.
[+] Architecture    : ARM AArch64 (87)
[+] BFD target name : elf64-littleaarch64
[+] Entry point     : 0x6B0
[+] Sections        : 22

     Name           Address             Section          Size       Stack Size

[+]ctest_malloc_vulnerable       00000804             .text 40        48:

         free@GLIBC_2.17 at:

                    00000820

        malloc@GLIBC_2.17 at:

                    00000814

           Stack:

                    Offset   Size

                    0:         0
```

```
[+]    _fini        0000082c            .fini  0        0:
          Stack:
                   Offset   Size

[+]ctest_malloc 000007c8            .text 60        32:
        free@GLIBC_2.17 at:
                   000007f8
       malloc@GLIBC_2.17 at:
                   000007dc
          Stack:
                   Offset   Size
                   0:       0

[+]    _init        00000638            .init  0        0:
          Stack:
                   Offset   Size

ctest_malloc_vulnerable(0x804): malloc@GLIBC_2.17(0x814)
          Warning:          First argument of malloc is external
          Possible Vulnerability:   Returned results are not being checked
```

Με την παράμετρο -d εκτυπώνεται η assembly ενός τομέα. Στο παρακάτω παράδειγμα εκτυπώνεται ο τομέας .plt.

```
$   python carrion.py libctestaarch64.so.1.0 -d .plt

[+] Creating BFD instance...
[+] File format    : Linker/assembler/compiler output.
[+] Architecture   : ARM AArch64 (87)
[+] BFD target name : elf64-littleaarch64
[+] Entry point    : 0x6B0
[+] Sections       : 22
```

```
Disassembly of section .plt at 00000650
   650 (4)        stp     x16, x30, [sp,#-16]!
   654 (4)        adrp    x16, 0x0000000000010000
   658 (4)        ldr     x17, [x16,#4088]
   65c (4)        add     x16, x16, #0xff8
   660 (4)        br      x17
   664 (4)        nop
   668 (4)        nop
   66c (4)        nop
   670 (4)        adrp    x16, 0x0000000000011000
   674 (4)        ldr     x17, [x16]
   678 (4)        add     x16, x16, #0x0
   67c (4)        br      x17
   680 (4)        adrp    x16, 0x0000000000011000
   684 (4)        ldr     x17, [x16,#8]
   688 (4)        add     x16, x16, #0x8
   68c (4)        br      x17
   690 (4)        adrp    x16, 0x0000000000011000
   694 (4)        ldr     x17, [x16,#16]
   698 (4)        add     x16, x16, #0x10
   69c (4)        br      x17
   6a0 (4)        adrp    x16, 0x0000000000011000
   6a4 (4)        ldr     x17, [x16,#24]
   6a8 (4)        add     x16, x16, #0x18
   6ac (4)        br      x17


ctest_malloc_vulnerable(0x804): malloc@GLIBC_2.17(0x814)
         Warning         First argument of malloc is external
         Vulnerability   Returned results are not being checked
```

Οι εγγραφές του τομέα .rela.plt στην αρχιτεκτονική aarch64, όπου είναι η βιβλιοθήκη του παραδείγματος είναι μεγέθους 24 bytes (6 DWords = 6 x 32bits). Χρησιμοποιώντας τις παραμέτρους -b και -s βλέπουμε τις εγγραφές του τομέα.

```
$    python carrion.py libctestaarch64.so.1.0 -b .rela.plt -s 6
[+] Creating BFD instance...
[+] File format     : Linker/assembler/compiler output.
[+] Architecture    : ARM AArch64 (87)
[+] BFD target name : elf64-littleaarch64
[+] Entry point     : 0x6B0
[+] Sections        : 22

Raw bytes of section .rela.plt at 000005d8


Next Entry 0

000005d8(+00000000): 00011000
000005dc(+00000004): 00000000
000005e0(+00000008): 00000402
000005e4(+0000000c): 00000004
000005e8(+00000010): 00000000
000005ec(+00000014): 00000000

Next Entry 1

000005f0(+00000018): 00011008
000005f4(+0000001c): 00000000
000005f8(+00000020): 00000402
000005fc(+00000024): 00000005
00000600(+00000028): 00000000
00000604(+0000002c): 00000000
```

```
Next Entry 2

00000608(+00000030): 00011010
0000060c(+00000034): 00000000
00000610(+00000038): 00000402
00000614(+0000003c): 00000006
00000618(+00000040): 00000000
0000061c(+00000044): 00000000

Next Entry 3

00000620(+00000048): 00011018
00000624(+0000004c): 00000000
00000628(+00000050): 00000402
0000062c(+00000054): 00000007
00000630(+00000058): 00000000
00000634(+0000005c): 00000000

ctest_malloc_vulnerable(0x804): malloc@GLIBC_2.17(0x814)
        Warning:        First argument of malloc is external
        Possible Vulnerability:   Returned results are not being checked
```

Τέλος, για τον έλεγχο των αλφαριθμητικών τιμών, είτε αυτές αποτελούν κάποιο όνομα συνάρτησης, τομέα κλπ., είτε κάποια σταθερά, και αναζητείται η τιμή ή η διεύθυνση στην οποία υπάρχει, γίνεται χρήση της παραμέτρου -c και -S συνδυαστικά. Για το παρακάτω παράδειγμα χρησιμοποιήθηκε η διεύθυνση και το μέγεθος του τομέα .strtab.

```
$    python carrion.py libctestaarch64.so.1.0 -c 0x1f28 -S 741
[+] Creating BFD instance...
[+] File format     : Linker/assembler/compiler output.
[+] Architecture    : ARM AArch64 (87)
[+] BFD target name : elf64-littleaarch64
[+] Entry point     : 0x6B0
[+] Sections        : 22


Strings at 00001f28


00001f28(+00000000):

00001f29(+00000001):  /usr/lib/gcc-cross/aarch64-linux-gnu/4.8/../../../../aarch64-linux-
gnu/lib/../lib/crti.o

00001f82(+0000005a):  $x

00001f85(+0000005d):  call_weak_fn

00001f92(+0000006a):  /usr/lib/gcc-cross/aarch64-linux-gnu/4.8/../../../../aarch64-linux-
gnu/lib/../lib/crtn.o

00001feb(+000000c3):  crtstuff.c

00001ff6(+000000ce):   __JCR_LIST__

00002003(+000000db): deregister_tm_clones

00002018(+000000f0):  register_tm_clones

0000202b(+00000103): $d

0000202e(+00000106): __do_global_dtors_aux

00002044(+0000011c):  completed.7422

00002053(+0000012b): __do_global_dtors_aux_fini_array_entry

0000207a(+00000152):  frame_dummy

00002086(+0000015e): __frame_dummy_init_array_entry

000020a5(+0000017d): ctest.c

000020ad(+00000185): __FRAME_END__

000020bb(+00000193): __JCR_END__

000020c7(+0000019f):   __dso_handle

000020d4(+000001ac):  _DYNAMIC
```

```
000020dd(+000001b5): __TMC_END__
000020e9(+000001c1): _GLOBAL_OFFSET_TABLE_
000020ff(+000001d7): _ITM_deregisterTMCloneTable
0000211b(+000001f3): __bss_start__
00002129(+00000201): __cxa_finalize@@GLIBC_2.17
00002144(+0000021c): _bss_end__
0000214f(+00000227): _edata
00002156(+0000022e): _fini
0000215c(+00000234): __bss_end__
00002168(+00000240): malloc@@GLIBC_2.17
0000217b(+00000253): __gmon_start__
0000218a(+00000262): _end
0000218f(+00000267): free@@GLIBC_2.17
000021a0(+00000278): __end__
000021a8(+00000280): ctest_malloc
000021b5(+0000028d): __bss_start
000021c1(+00000299): ctest_malloc_vulnerable
000021d9(+000002b1): _Jv_RegisterClasses
000021ed(+000002c5): _ITM_registerTMCloneTable
00002207(+000002df): _init


ctest_malloc_vulnerable(0x804): malloc@GLIBC_2.17(0x814)
        Warning:        First argument of malloc is external
        Possible Vulnerability:   Returned results are not being checked
```

6 Πειραματική αξιολόγηση

6.1 Εισαγωγή

Το εργαλείο ελέγχου ευπαθειών δοκιμάστηκε σε περιβάλον Ubuntu 14.04.4 64-bit, 2GB ram, στις βιβλιοθήκες λειτουργικού συστήματος Ubuntu 14.04.4 που βρέθηκαν στους φακέλους /lib, /lib64, /lib32 και /usr (συγκεκριμένα στους υποφακέλους usr/lib, /usr/lib32, /usr/arm-linux-gnueabi/lib, /usr/x86_64-linux-gnu/arm-linux-gnueabi/lib, /usr/x86_64-linux-gnu/aarch64-linux-gnu/lib, /usr/arm-linux-gnueabihf/lib/sf, /usr/arm-linux-gnueabi/lib, /usr/aarch64-linux-gnu/lib). Επίσης, δοκιμάστηκε στις βιβλιοθήκες λειτουργικού συστήματος Android 7.1.1 που βρέθηκαν στους φακέλους /fake-libs, /fake-libs64 /lib (και στους υποφακέλους /lib/hw, /lib/drm, /lib/soundfx), /lib64 (και στους υποφακέλους /lib64/drm, /lib64/hw, /lib64/soundfx) και /vendor (και στους υποφακέλους /vendor/app/ims/lib/arm64, /vendor/lib64, /vendor/lib64/egl, /vendor/lib64/hw, /vendor/lib64/mediadrm, /vendor/lib, /vendor/lib/drm, /vendor/lib/egl, /vendor/lib/hw, /vendor/lib/mediadrm). Λόγω του όγκου των ευρημάτων, θα αναφερθούν ορισμένα από αυτά, κυρίως αυτά που θεωρήθηκαν σημαντικά ως ευρήματα, καθώς και κάποιες τυχαίες επιλογές για την κάλυψη όλων των περιπτώσεων. Βασικό κριτήριο επιλογής ήταν η δυνατότητα επαλήθευσης των αποτελεσμάτων, ελέγχοντας τον πηγαίο κώδικα, όπου αυτός ήταν διαθέσιμος (λογισμικό ανοιχτού κώδικα).

6.2 Ευρήματα περιπτώσεων ευπαθειών ασφαλείας κατά την χρήση της συνάρτησης system()

Ευπάθειες ασφαλείας στη χρήση της συνάρτησης system() της βιβλιοθήκης GLIBC βρέθηκαν στις εξής βιβλιοθήκες:

- liblirc_client.so.0.2.1 (intel x64): Στην συνάρτηση lirc_readconfig() (http://www.lirc.org/api-docs/html/lirc__client_8c_source.html γραμμή 1445) γίνεται η χρήση της συνάρτησης system() (γραμμή 1495) με όρισμα την αλφαριθμητική μεταβλητή “command”, η οποία ορίζεται στη γραμμή 1493, όπου χρησιμοποιείται το εξωτερικό όρισμα “config”, χωρίς έλεγχο, με αποτέλεσμα η χρήση της συνάρτησης lirc_readconfig() να είναι ευπαθές σε επιθέσεις command injection.

```
liblirc_client.so.0.2.1 (/usr/lib/liblirc_client.so.0.2.1)


[+] Creating BFD instance...
[+] File format    : Linker/assembler/compiler output.
[+] Architecture   : Intel 386 (9)
[+] BFD target name : elf64-x86-64
[+] Entry point    : 0x1440
[+] Sections       : 24


lirc_readconfig(0x4200): system@GLIBC_2.2.5(0x4419)
        Possible Vulnerability:   First argument of system is external
```

- libicalss.so.1.0.0 (intel x64): Στην συνάρτηση icalfileset_commit() (https://github.com/libical/libical/blob/master/src/libicalss/icalfileset.c#L339, γραμμή 339)

γίνεται η χρήση της συνάρτησης system() (γραμμή 371) με όρισμα την αλφαριθμητική μεταβλητή “tmp”, η οποία ορίζεται στις γραμμές 364 ή 367, με την χρήση του εξωτερικού ορίσματος “fset”, χωρίς έλεγχο, με αποτέλεσμα η χρήση της συνάρτησης icalfileset_commit() να είναι ευπαθές σε επιθέσεις command injection.

```
libicalss.so.1.0.0 (/usr/lib/libicalss.so.1.0.0)

[+] Creating BFD instance...
[+] File format     : Linker/assembler/compiler output.
[+] Architecture    : Intel 386 (9)
[+] BFD target name : elf64-x86-64
[+] Entry point     : 0x7440
[+] Sections        : 25

icalfileset_commit(0xa9e0): system@GLIBC_2.2.5(0xabda)
        Possible Vulnerability:    First argument of system is external
```

6.3 Ευρήματα περιπτώσεων ευπαθειών ασφαλείας κατά την χρήση της συνάρτησης malloc()

Ευπάθειες ασφαλείας στη χρήση της συνάρτησης malloc() της βιβλιοθήκης LIBC βρέθηκαν στις εξής βιβλιοθήκες:

- libstagefright_enc_common.so(arm x64): Στην συνάρτηση cmnMemAlloc() (https://android.googlesource.com/platform/frameworks/av/+/e2e838afcf03e603a41a0455846eaf9614537c16/media/libstagefright/codecs/amrwbenc/src/cmnMemory.c#31 γραμμή 31) γίνεται η χρήση της συνάρτησης malloc() (γραμμή 36) με το εξωτερικό όρισμα “pMemInfo”, το οποίο δεν ελέγχεται, καθώς επίσης δεν ελέγχεται και η τιμή επιστροφής της συνάρτησης malloc(). Η χρήση της συνάρτησης cmnMemAlloc() ενδέχεται να δημιουργήσει δείκτη θέσης μνήμης με την τιμή 0, στην εξωτερική μεταβλητή “pMemInfo”.

```
libstagefright_enc_common.so (/android-7.1.1-system/lib64/libstagefright_enc_common.so)

[+] Creating BFD instance...
[+] File format     : Linker/assembler/compiler output.
[+] Architecture    : ARM AArch64 (87)
[+] BFD target name : elf64-littleaarch64
[+] Entry point     : 0x0
[+] Sections        : 21

cmnMemAlloc(0x820): malloc@LIBC(0x840)
        Warning:          First argument of malloc is external
        Possible Vulnerability:    Returned results are not being checked
```

6.4 Ευρήματα περιπτώσεων ευπαθειών ασφαλείας κατά την χρήση της συνάρτησης strcpy()

Ευπάθειες ασφαλείας στη χρήση της συνάρτησης strcpy() της βιβλιοθήκης GLIBC βρέθηκαν στις εξής βιβλιοθήκες:

- liblangtag.so.1.2.0 (intel x64): Στην συνάρτηση lt_string_new() (https://searchcode.com/file/45587323/liblangtag/lt-string.c γραμμή 80) γίνεται η χρήση της συνάρτησης strcpy() (γραμμή 93) με πρώτο όρισμα την εσωτερική μεταβλητή της συνάρτησης “retval” που ορίζεται στην γραμμή 82 και δεύτερο όρισμα την εξωτερική μεταβλητή “string”. Η χρήση της συνάρτησης lt_string_new() είναι ευάλωτη σε ευπάθειες ασφαλείας buffer overflow και format string bugs.

```
liblangtag.so.1.2.0 (/usr/lib/liblangtag.so.1.2.0)

[+] Creating BFD instance...
[+] File format    : Linker/assembler/compiler output.
[+] Architecture   : Intel 386 (9)
[+] BFD target name : elf64-x86-64
[+] Entry point    : 0x6F50
[+] Sections       : 25

lt_string_new(0xef90): strcpy@GLIBC_2.2.5(0xefdf)
        Warning:        Second argument of strcpy is external
        Possible Vulnerability:   First argument of strcpy is internal, while second argument is external
```

6.5 Ευρήματα περιπτώσεων ευπαθειών ασφαλείας κατά την χρήση της συνάρτησης strncpy()

Ευπάθειες ασφαλείας στη χρήση της συνάρτησης strncpy() της βιβλιοθήκης GLIBC βρέθηκαν στις εξής βιβλιοθήκες:

- libext2_e2p.so (arm x64): Στην συνάρτηση list_super2() (https://android.googlesource.com/platform/external/e2fsprogs/+/froyo-release/lib/e2p/ls.c#176 γραμμή 176) γίνεται η χρήση της συνάρτησης strncpy() δύο φορές (γραμμές 188 και 194), με πρώτο όρισμα την εσωτερική μεταβλητή της συνάρτησης “buff”, που ορίζεται στη γραμμή 179 και δεύτερο όρισμα την εξωτερική μεταβλητή “sb”. Η χρήση της συνάρτησης list_super2() είναι ευάλωτη σε ευπάθειες ασφαλείας format string bugs.

```
libext2_e2p.so (/android-7.1.1-system/lib64/libext2_e2p.so)


[+] Creating BFD instance...
[+] File format    : Linker/assembler/compiler output.
[+] Architecture   : ARM AArch64 (87)
[+] BFD target name : elf64-littleaarch64
[+] Entry point    : 0x0
[+] Sections       : 24


list_super2(0x2d78): strncpy@LIBC(0x2dfc)
        Warning:        Second argument of strncpy is external
        Possible Vulnerability:   First argument of strncpy is internal, while second argument is external


list_super2(0x2d78): strncpy@LIBC(0x2e60)
        Warning:        Second argument of strncpy is external
        Possible Vulnerability:   First argument of strncpy is internal, while second argument is external
```

6.6 Ευρήματα περιπτώσεων ευπαθειών ασφαλείας κατά την χρήση της συνάρτησης strcat()

Ευπάθειες ασφαλείας στη χρήση της συνάρτησης strcat() της βιβλιοθήκης GLIBC βρέθηκαν στις εξής βιβλιοθήκες:

- libkpathsea.so.6.1.1 (intel x64): Στην συνάρτηση concat() (http://www.luatex.org/svn/tags/beta-0.31.1/src/texk/kpathsea/concat.c γραμμή 26) γίνεται η χρήση της συνάρτησης strcat() (γραμμή 32) με πρώτο όρισμα την εσωτερική μεταβλητή της συνάρτησης “answer”, που ορίζεται στη γραμμή 30 και δεύτερο όρισμα την εξωτερική μεταβλητή “s2”. Στην συνάρτηση concat3() (http://www.luatex.org/svn/tags/beta-0.30.2/src/texk/kpathsea/concat3.c γραμμή 21) γίνεται η χρήση της συνάρτησης strcat() (γραμμή 27) με πρώτο όρισμα την εσωτερική μεταβλητή της συνάρτησης “answer”, που ορίζεται στη γραμμή 24 και δεύτερο όρισμα την εξωτερική μεταβλητή “s2”. Οι χρήσεις των συναρτήσεων concat() και concat3() είναι ευάλωτες σε ευπάθειες ασφαλείας format string bugs.

```
libkpathsea.so.6.1.1 (/usr/lib/libkpathsea.so.6.1.1)

[+] Creating BFD instance...
[+] File format     : Linker/assembler/compiler output.
[+] Architecture    : Intel 386 (9)
[+] BFD target name : elf64-x86-64
[+] Entry point     : 0x41E0
[+] Sections        : 25

concat(0x8700): strcat@GLIBC_2.2.5(0x8742)
        Warning:          Second argument of strcat is external
        Possible Vulnerability:   First argument of strcat is internal, while second argument is external

concat3(0x8760): strcat@GLIBC_2.2.5(0x87d2)
        Warning:          Second argument of strcat is external
        Possible Vulnerability:   First argument of strcat is internal, while second argument is external
```

6.7 Ευρήματα περιπτώσεων ευπαθειών ασφαλείας κατά την χρήση της συνάρτησης strncat()

Δεν βρέθηκαν περιπτώσεις ευπάθειας ασφαλείας στη χρήση της συνάρτησης strncat() που να μπορούσαν να επαληθευθούν στα ολοκληρωμένα αποτελέσματα των βιβλιοθηκών που ερευνήθηκαν. Η εξέταση των περιπτώσεων χρήσης της strncat() παρέμεινε στις δοκιμαστικές βιβλιοθήκες που δημιουργήθηκαν κατά την ανάπτυξη του εργαλείου.

```
libctestx32.so
[+] Creating BFD instance...
[+] File format    : Linker/assembler/compiler output.
[+] Architecture   : Intel 386 (9)
[+] BFD target name : elf32-i386
[+] Entry point    : 0xBF0
[+] Sections       : 24

ctest_strncat_vulnerable(0x1317): strncat@GLIBC_2.0(0x134f)
        Warning:        Second argument of strncat is external
        Possible Vulnerability:   First argument of strncat is internal, while second argument is external
```

6.8 Ευρήματα περιπτώσεων ευπαθειών ασφαλείας κατά την χρήση της συνάρτησης sprintf()

Ευπάθειες ασφαλείας στη χρήση της συνάρτησης sprintf() της βιβλιοθήκης GLIBC βρέθηκαν στις εξής βιβλιοθήκες:

- libnss_dns-2.19.so (arm x64): Στη συνάρτηση _nss_dns_gethostbyaddr2_r() (https://code.woboq.org/userspace/glibc/resolv/nss_dns/dns-host.c.html γραμμή 409) γίνεται η χρήση της συνάρτησης sprintf() (γραμμή 495) με πρώτο όρισμα την εσωτερική μεταβλητή της συνάρτησης " qbuf", που ορίζεται στη γραμμή 430, και δεύτερο όρισμα την εσωτερική μεταβλητή "uaddr", η οποία ορίζεται στη γραμμή 416 και παίρνει την τιμή της από την εξωτερική μεταβλητή "addr", χωρίς έλεγχο, επίσης στην γραμμή 416. Η χρήση της συνάρτησης _nss_dns_gethostbyaddr2_r() είναι ευάλωτη σε ευπάθειες ασφαλείας buffer overflow.

```
libnss_dns-2.19.so (/usr/aarch64-linux-gnu/lib/libnss_dns-2.19.so)


[+] Creating BFD instance...
[+] File format    : Linker/assembler/compiler output.
[+] Architecture   : ARM AArch64 (87)
[+] BFD target name : elf64-littleaarch64
[+] Entry point    : 0x1060
[+] Sections       : 26


_nss_dns_gethostbyaddr2_r(0x2998): sprintf@GLIBC_2.17(0x2e4c)
        Warning:        Second argument of sprintf is external
```

6.9 Ευρήματα περιπτώσεων ευπαθειών ασφαλείας κατά την χρήση της συνάρτησης snprintf()

Ευπάθειες ασφαλείας στη χρήση της συνάρτησης snprintf() της βιβλιοθήκης GLIBC βρέθηκαν στις εξής βιβλιοθήκες:

- libnss_nisplus-2.19.so (arm x32): Στη συνάρτηση _nss_nisplus_getrpcbyname_r() (https://searchcode.com/file/34681580/Sources/selp_glibc-2.5.90-19.0.46/nis/nss_nisplus/nisplus-rpc.c γραμμή 292) γίνεται η χρήση της συνάρτησης snprintf() (γραμμές 317 και 333) με τρίτο όρισμα την εξωτερική μεταβλητή " name", χωρίς έλεγχο, και πρώτο όρισμα την εσωτερική μεταβλητή "buff", που ορίζεται στη γραμμή 312. Η χρήση της συνάρτησης _nss_nisplus_getrpcbyname_r() είναι ευάλωτη σε ευπάθειες ασφαλείας format string bugs.

```
libnss_nisplus-2.19.so (/lib32/libnss_nisplus-2.19.so)


[+] Creating BFD instance...

[+] File format     : Linker/assembler/compiler output.

[+] Architecture    : Intel 386 (9)

[+] BFD target name : elf32-i386

[+] Entry point     : 0x1A20

[+] Sections        : 28


_nss_nisplus_getrpcbyname_r(0x6860): snprintf@GLIBC_2.0(0x694d)

        Warning:        Third argument of snprintf is external

        Possible Vulnerability:   Returned results are not being checked


_nss_nisplus_getrpcbyname_r(0x6860): snprintf@GLIBC_2.0(0x6a86)

        Warning:        Third argument of snprintf is external

        Possible Vulnerability:   Returned results are not being checked
```

6.10 Ευρήματα περιπτώσεων ευπαθειών ασφαλείας κατά την χρήση της συνάρτησης memcpy()

Ευπάθειες ασφαλείας στη χρήση της συνάρτησης memcpy() της βιβλιοθήκης LIBC βρέθηκαν στις εξής βιβλιοθήκες:

- libtinyalsa.so (arm x64): Στην συνάρτηση mixer_ctl_set_array() (https://android.googlesource.com/platform/external/tinyalsa/+/master/mixer.c#458 γραμμή 458), γίνεται η χρήση της συνάρτησης memcpy() (γραμμές 501 και 522) με δεύτερο όρισμα εξωτερική "array", τρίτο όρισμα την μεταβλητή "count" μεταβλητή και πρώτο όρισμα την εσωτερική μεταβλητή "tlv" στη γραμμή 501, που ορίζεται στις γραμμές 492 και 496, και την εσωτερική μεταβλητή "dest" στη γραμμή 522, που ορίζεται στις γραμμές 462, 486, 509 και 515.

Η χρήση της συνάρτησης mixer_ctl_set_array() είναι ευάλωτη σε ευπάθειες ασφαλείας buffer overflow.

```
libtinyalsa.so (/android-7.1.1-system/lib64/libtinyalsa.so)


[+] Creating BFD instance...
[+] File format    : Linker/assembler/compiler output.
[+] Architecture   : ARM AArch64 (87)
[+] BFD target name : elf64-littleaarch64
[+] Entry point    : 0x0
[+] Sections       : 23


mixer_ctl_set_array(0x286c): memcpy@LIBC(0x2928)
        Warning:        Second argument of mempcy is external
        Possible Vulnerability:   First argument of memcpy is internal, while second argument is external


mixer_ctl_set_array(0x286c): memcpy@LIBC(0x29cc)
        Warning:        Second argument of mempcy is external
```

6.11 Ευρήματα περιπτώσεων ευπαθειών ασφαλείας κατά την χρήση της συνάρτησης open()

Ευπάθειες ασφαλείας στη χρήση της συνάρτησης open() της βιβλιοθήκης GLIBC βρέθηκαν στις εξής βιβλιοθήκες:

- libext2_e2p.so (arm x64): Στη συνάρτηση fsetversion() (https://android.googlesource.com/platform/external/e2fsprogs/+/refs/heads/lollipop-mr1-release/lib/e2p/fsetversion.c#39 γραμμή 39) γίνεται η χρήση της συνάρτησης open() (γραμμή 45) με πρώτο όρισμα την εξωτερική μεταβλητή “name”. Στη συνάρτηση fgetversion() (https://android.googlesource.com/platform/external/e2fsprogs/+/refs/heads/lollipop-mr1-release/lib/e2p/fgetversion.c#39 γραμμή 39)) γίνεται η χρήση της συνάρτησης open() (γραμμή 45) με πρώτο όρισμα την εξωτερική μεταβλητή “name”. Στη συνάρτηση fgetflags() (https://android.googlesource.com/platform/external/e2fsprogs/+/refs/heads/lollipop-mr1-release/lib/e2p/fgetflags.c#43 γραμμή 43) γίνεται η χρήση της συνάρτησης open() (γραμμή 75) με πρώτο όρισμα την εξωτερική μεταβλητή “name”. Στη συνάρτηση fsetflags() (https://android.googlesource.com/platform/external/e2fsprogs/+/refs/heads/lollipop-mr1-release/lib/e2p/fsetflags.c#52 γραμμή 52) γίνεται η χρήση της συνάρτησης open() (γραμμή 81) με πρώτο όρισμα την εξωτερική μεταβλητή “name”. Οι χρήσεις των συναρτήσεων fsetversion(), fgetversion(), fgetflags() και fsetflags() μπορεί να ανοίξουν κακόβουλο αρχείο ή κάποιο κρίσιμο αρχείο συστήματος.

```
libext2_e2p.so (/android-7.1.1-system/lib64/libext2_e2p.so)

[+] Creating BFD instance...
[+] File format    : Linker/assembler/compiler output.
[+] Architecture   : ARM AArch64 (87)
[+] BFD target name : elf64-littleaarch64
[+] Entry point    : 0x0
[+] Sections       : 24

fsetversion(0x2938): open@LIBC(0x296c)
        Possible Vulnerability:   First argument of open is external

fgetversion(0x2874): open@LIBC(0x28a8)
        Possible Vulnerability:   First argument of open is external

fgetflags(0x2664): open@LIBC(0x26c8)
        Possible Vulnerability:   First argument of open is external

fsetflags(0x276c): open@LIBC(0x27cc)
        Possible Vulnerability:   First argument of open is external
```

6.12 Ευρήματα περιπτώσεων ευπαθειών ασφαλείας κατά την χρήση της συνάρτησης fopen()

Ευπάθειες ασφαλείας στη χρήση της συνάρτησης fopen() της βιβλιοθήκης GLIBC βρέθηκαν στις εξής βιβλιοθήκες:

- libwpa_client.so (arm x64): Στη συνάρτηση os_readfile() (https://android.googlesource.com/platform/external/wpa_supplicant_8/+/l-preview/src/utils/os_unix.c#373 γραμμή 373, γίνεται η χρήση της συνάρτησης fopen() (γραμμή 379) με όρισμα την εξωτερική μεταβλητή “name”. Στη συνάρτηση os_file_exists() (https://android.googlesource.com/platform/external/wpa_supplicant_8/+/l-preview/src/utils/os_unix.c#411 γραμμή 411, γίνεται η χρήση της συνάρτησης fopen() (γραμμή 413) με όρισμα την εξωτερική μεταβλητή “name”. Στη συνάρτηση os_daemonize() (https://android.googlesource.com/platform/external/wpa_supplicant_8/+/l-preview/src/utils/os_unix.c#194 γραμμή 194) γίνεται η χρήση της συνάρτησης fopen() (γραμμή 205) με όρισμα την εξωτερική μεταβλητή “pid_file”. Οι χρήσεις των συναρτήσεων os_readfile(), os_file_exists() και os_daemonize() μπορεί να ανοίξουν κακόβουλο αρχείο ή κάποιο κρίσιμο αρχείο συστήματος.

```
libwpa_client.so (/android-7.1.1-system/lib64/libwpa_client.so)

[+] Creating BFD instance...
[+] File format     : Linker/assembler/compiler output.
[+] Architecture    : ARM AArch64 (87)
[+] BFD target name : elf64-littleaarch64
[+] Entry point     : 0x0
[+] Sections        : 23

os_readfile(0x2ea0): fopen@LIBC(0x2ebc)
        Possible Vulnerability:    First argument of fopen is external

os_file_exists(0x2f70): fopen@LIBC(0x2f80)
        Possible Vulnerability:    First argument of fopen is external

os_daemonize(0x2aa8): fopen@LIBC(0x2ad8)
        Possible Vulnerability:    First argument of fopen is external
```

6.13 Παρατηρήσεις

Κατά την διεξαγωγή της πειραματικής αξιολόγησης έγιναν αρκετές αλλαγές στο εργαλείο εύρεσης ευπαθειών ασφαλείας, με σκοπό την βελτιστοποίηση των αποτελεσμάτων και την μείωση των εσφαλμένων αυτών. Όπως παρατηρήθηκε, μία περίπτωση κώδικα γλώσσας υψηλού επιπέδου μπορεί να εκφραστεί με διαφορετικούς τρόπους σε γλώσσες χαμηλού επιπέδου (assembly) , όπως για παράδειγμα η πράξη της αφαίρεσης σε κάποιες περιπτώσεις υλοποιούταν με την επιτηδευμένη χρήση υπερχείλισης σε πράξη πρόσθεσης, και κατά συνέπεια στη γλώσσα ενδιάμεσης αναπαράστασης, κάτι που οφείλεται στην βελτιστοποίηση των μεταγλωττιστών. Αυτό με τη σειρά του επηρέασε την ορθότητα των αποτελεσμάτων. Βασικό πρόβλημα αποτέλεσε η αναζήτηση τιμών των διαφόρων μεταβλητών της γλώσσας ενδιάμεσης αναπαράστασης, καθώς και η προέλευση αυτών – εσωτερικές ή εξωτερικές μεταβλητές. Επιπλέον, ένα εξίσου σημαντικό πρόβλημα ήταν η εμφάνιση εντολών της γλώσσας ενδιάμεσης αναπαράστασης, που δεν είχαν εμφανιστεί σε αντίστοιχες περιπτώσεις κώδικα κατά την υλοποίηση του εργαλείου. Παρατηρήθηκαν σφάλματα κατά την μεταγλώττιση κάποιων αρχείων στην γλώσσα ενδιάμεσης αναπαράστασης στις αρχιτεκτονικές arm x32 και arm x64, τα οποία οφείλονται στην βιβλιοθήκη pyvex. Τέλος, παρατηρήθηκε πρόβλημα κατανάλωσης μνήμης, σε μεγάλα αρχεία που δοκιμάστηκαν σε περιβάλλον εικονικού μηχανήματος με περιορισμένη μνήμη. Σε περιβάλλον Ubuntu 14.04.4 64-bit, 2 GB ram, ελέχθηκαν αρχεία μεγέθους έως και 710KB, σε χρόνο έως και 1 λεπτό.

7 Μελλοντικές προσθήκες

Το εργαλείο αναπτύχθηκε σε πρώτο στάδιο σύμφωνα με απλές βιβλιοθήκες που δημιουργήθηκαν κατά την υλοποίησή του, για την παρατήρηση των διαφορών των εκάστοτε assembly σε ίδιες περιπτώσεις, καθώς επίσης και της μεταφοράς αυτών στην γλώσσα ενδιάμεσης αναπαράστασης. Εν συνεχεία στην πειραματική αξιολόγηση διερευνήθηκαν και καλύφθηκαν όσες περιπτώσεις βρέθηκαν κατά την διάρκεια της.

Το ενδεχόμενο επιστροφής λάθους αποτελέσματος οφείλεται στις διαφορετικές περιπτώσεις υλοποίησης ίδιων υποσεναρίων. Η ανάθεση τιμών με οποιονδήποτε τρόπο, επηρεάζει τα υποσενάρια που διερευνούν την προέλευση (εσωτερική, εξωτερική) ή την τιμή μιας μεταβλητής. Αυτό κατά συνέπεια επηρεάζει τα αποτελέσματα των σεναρίων ευπαθειών ασφαλείας. Υπολογίζεται ότι υπάρχουν και άλλες περιπτώσεις υλοποίησης, οι οποίες δεν έχουν διερευνηθεί. Περαιτέρω πειραματική αξιολόγηση θα βελτίωνε τα αποτελέσματα, τόσο της επαλήθευσης υποσεναρίων, όσο και των περιπτώσεων ευπαθειών ασφαλείας.

Μία άλλη περίπτωση βελτιστοποίησης του εργαλείου είναι η αποφυγή φόρτωσης όλου του αρχείου της βιβλιοθήκης στην μνήμη. Η αντικατάσταση αυτής της περίπτωσης με την ανάγνωση συγκεκριμένου τμήματος του αρχείου της βιβλιοθήκης, όπου αυτό χρειάζεται, και την εκάστοτε μεταφορά του στην γλώσσα ενδιάμεσης αναπαράστασης, για την εξέταση σεναρίων, θα μείωνε την χρήση της μνήμης. Στην συγκεκριμένη περίπτωση ορισμένες από τις λειτουργίες αυτές ενδέχεται να επιφέρουν βελτίωση εάν χρησιμοποιηθεί διαφορετική γλώσσα υλοποίησης του εργαλείου, επιλέγοντας κάποια γλώσσα η οποία μεταγλωττίζεται πρώτα σε γλώσσα μηχανής και ύστερα εκτελείται και όχι γλώσσα που εκτελείται βήμα-βήμα, όπως η γλώσσα python που επιλέχθηκε για την υλοποίηση του εργαλείου.

Επίσης, μία επιπλέον βελτίωση στην κατανάλωση της μνήμης, είναι η μείωση χρήσης αναδρομικών συναρτήσεων, όπου μπορεί να επιτευχθεί με διαφορετική υλοποίηση κάποιων από αυτών. Σε μεγάλα αρχεία βιβλιοθηκών που εξετάζονται σε σύστημα χαμηλής μνήμης, όπως για παράδειγμα ένα εικονικό μηχάνημα, θα βελτίωνε το πρόβλημα κατανάλωσης μνήμης που παρατηρήθηκε κατά την πειραματική αξιολόγηση.

Η ενημέρωση του εργαλείου, όπου απαιτείται, ώστε αυτό να είναι συμβατό με την τελευταία έκδοση της βιβλιοθήκης pyvex θα λύσει το πρόβλημα σφαλμάτων που παρουσιάστηκαν κατά την μεταγλώττιση στην ενδιάμεση αναπαράσταση, όπως αναφέρθηκε στις παρατηρήσεις της πειραματικής αξιολόγησης.

Το εργαλείο σχεδιάστηκε έτσι ώστε να διευκολύνει όσων το δυνατό περισσότερο την μελλοντική υλοποίηση άλλων σεναρίων ευπαθειών ασφαλείας. Μελλοντική έρευνα θα μπορούσε να προσθέσει και άλλα σενάρια στο πεδίο έρευνας του εργαλείου, με ελάχιστη υλοποίηση. Επίσης, είναι δυνατή η ανάπτυξη ανάγνωσης αρχείου που θα δημιουργείται από χρήστες, το οποίο θα περιέχει τα υποσεναρία που επιλέχθηκαν από τον χρήστη, καθώς και την διασύνδεση αυτών, με σκοπό την δημιουργία σεναρίων ευπαθειών ασφαλείας.

8 Συμπεράσματα

Λόγο της ραγδαίας ανάπτυξης υλικού, και κατά συνέπεια του λογισμικού, θεωρήθηκε χρήσιμη η ανάπτυξη εργαλείου αυτοματοποιημένου ελέγχου εύρεσης ευπαθειών ασφαλείας σε αρχεία βιβλιοθηκών διαφορετικών μεταξύ τους αρχιτεκτονικών, που ενδέχεται να χρησιμοποιηθούν κατά την ανάπτυξη εφαρμογών, ακόμα και όταν ο πηγαίος κώδικας αυτών δεν είναι διαθέσιμος. Βασική ανάγκη κρίθηκε η επικάλυψη των διαφορών μεταξύ των αρχιτεκτονικών, ώστε να υπάρχουν ενιαία σενάρια ελέγχου.

Μια διαδικασία εύρεσης ευπαθειών ασφαλείας είναι ανάλυση συμπεριφοράς προγραμμάτων. Η διαδικασία της ανάλυσης μπορεί να πραγματοποιηθεί με την στατική ανάλυση, την δυναμική ανάλυση και την συμβολική εκτέλεση. Κάθε περίπτωση από τις παραπάνω έχει διαφορετικό στόχο και εξετάζει διαφορετικές περιπτώσεις. Στη στατική ανάλυση αναλύεται ο πηγαίος κώδικας ενός προγράμματος, χωρίς αυτό να εκτελεστεί. Στη δυναμική ανάλυση, εξετάζεται η συμπεριφορά του προγράμματος κατά

την εκτέλεση του σε έναν πραγματικό ή εικονικό περιβάλλον με αρκετές εισόδους ελέγχου. Στη συμβολική εκτέλεση αναλύεται ένα πρόγραμμα με σκοπό τον καθορισμό των παραμέτρων που προκαλούν την εκτέλεση των εκάστοτε μερών του προγράμματος και η εκτέλεση βασίζεται στις διαφορετικές ροές του προγράμματος. Για την ανάλυση διαφορετικών αρχιτεκτονικών σε ίδια σενάρια ευπαθειών ασφαλείας, μπορεί να γίνει χρήση μιας γλώσσας ενδιάμεσης αναπαράστασης, αφαιρώντας έτσι, αρκετές διαφορές που υπάρχουν μεταξύ των αρχιτεκτονικών και επιτρέποντας μια ενιαία κοινή ανάλυση για όλες τις αρχιτεκτονικές.

Το εργαλείο αναπτύχθηκε σε γλώσσα python και παρέχει ελέγχους για περιπτώσεις ευπαθειών ασφαλείας σε βιβλιοθήκες αρχιτεκτονικών intel x86, intel x64, arm x32 και arm x64, που εμφανίζονται υπό συνθήκες κατά την χρήση των παρακάτω συναρτήσεων της βασικής βιβλιοθήκης libc: fopen(), open, malloc(), memcpy(), snprintf, sprintf(), strcat (), strcpy(), strncat(), strncpy() και system(). Για την κάλυψη των περιπτώσεων αυτών ελέγχονται η τιμή και η προέλευση μιας μεταβλητής, εσωτερική ή εξωτερική, με διάφορους τρόπους, αναλόγως την περίπτωση που εξετάζεται, εξετάζεται αν υπάρχει έλεγχος του αποτελέσματος που επιστρέφεται, καθώς και συνδυασμός των παραπάνω. Η υλοποίηση του εργαλείου διαχωρίστηκε σε διαφορετικά επίπεδα. Αναπτύχθηκαν διαφορετικά αρχεία για κάθε αρχιτεκτονική, στα οποία υλοποιούνται ίδια υποσενάρια, καλύπτοντας τις διαφορές που έχει η εκάστοτε αρχιτεκτονική και δεν ήταν δυνατό να καλυφθούν με την χρήση της γλώσσας ενδιάμεσης αναπαράστασης. Αναπτύχθηκαν, επίσης, αρχεία τα οποία περιλαμβάνουν βασικές λειτουργίες, τόσο για την ανάλυση του αρχείου, όσο και για τον έλεγχο περιπτώσεων ευπαθειών ασφαλείας, που είναι κοινές για όλες τις αρχιτεκτονικές. Επίσης, παρέχονται και βοηθητικές συναρτήσεις, οι οποίες προσφέρουν πληροφορίες σχετικές με την ανάλυση του αρχείου. Τέλος, αναπτύχθηκε ο έλεγχος των ευπαθειών ασφαλείας, βασισμένος στη γλώσσα ενδιάμεσης αναπαράστασης pyvex που χρησιμοποιήθηκε. Κατά την εκτέλεση του εργαλείου, αρχικά γίνεται ανάλυση του αρχείου για την συλλογή των απαραίτητων πληροφοριών και των πληροφοριών που ζητήθηκαν, και έπειτα γίνεται η χρήση αυτών για τον έλεγχο των σεναρίων ευπαθειών ασφαλείας.

Στα πλαίσια της διπλωματικής εργασίας αναπτύχθηκε το εργαλείο carrion, το οποίο παρέχει τους εξής ελέγχους σε αρχεία βιβλιοθηκών intel x86, intel x64, arm x32 και arm x64 αρχιτεκτονικών :

- **fopen(), open():** Αν το όνομα του αρχείου (πρώτο όρισμα) προέρχεται από εξωτερική μεταβλητή και αν ελέγχεται η τιμή επιστροφής της συνάρτησης
- **malloc():** Αν η τιμή της μεταβλητής που δίνεται ως όρισμα προέρχεται από εξωτερική τιμή (προειδοποίηση) και αν ελέγχεται η τιμή επιστροφής της συνάρτησης
- **memcpy():** Προειδοποίηση αν το δεύτερο όρισμα προέρχεται από εξωτερική μεταβλητή και έλεγχος αν το πρώτο όρισμα είναι εσωτερική μεταβλητή της στοίβας, ενώ η τιμή του δεύτερου ορίσματος προέρχεται από εξωτερική μεταβλητή.
- **snprintf():** Προειδοποίηση αν το τρίτο όρισμα προέρχεται από εξωτερική μεταβλητή και αν ελέγχεται η τιμή επιστροφής της συνάρτησης.
- **sprintf():** Προειδοποίηση αν το δεύτερο όρισμα προέρχεται από εξωτερική μεταβλητή.
- **strcat (), strcpy(), strncat(), strncpy():** Προειδοποίηση αν το δεύτερο όρισμα προέρχεται από εξωτερική μεταβλητή και έλεγχος αν το πρώτο όρισμα είναι εσωτερική μεταβλητή της στοίβας, ενώ η τιμή του δεύτερου ορίσματος προέρχεται από εξωτερική μεταβλητή.
- **system():** Έλεγχος αν το όρισμα προέρχεται από εξωτερική μεταβλητή.

Επίσης, το εργαλείο παρέχει εκτύπωση των συμβόλων του αρχείου, αναλυτική λίστα των συναρτήσεων και των διευθύνσεων αυτών που βρέθηκαν στο αρχείο, εκτύπωση ενδιάμεσης αναπαράστασης, disassemble, πληροφορίες τομέων, bytes του αρχείου σε δεκαεξαδική μορφή, εκτύπωση σημείων του αρχείου με την μορφή αλφαριθμητικών μεταβλητών, όπου η διεύθυνση ή ο τομέας και το μέγεθος προς ανάλυση της εκάστοτε συνάρτησης ελέγχονται από τον χρήστη με αντίστοιχα ορίσματα.

Κατά την διάρκεια της πειραματικής αξιολόγησης, το εργαλείο δοκιμάστηκε και βελτιώθηκε σε πραγματικές βιβλιοθήκες όλων των αρχιτεκτονικών που σχεδιάστηκε να ερευνά. Ως κριτήρια αποτελεσμάτων και βελτίωσης του εργαλείου θεωρήθηκαν όσες βιβλιοθήκες αποτελούσαν λογισμικό ανοιχτού κώδικα. Βελτιώθηκε η αναζήτηση τιμών μεταβλητών, καθώς και η αναγνώριση προέλευσης αυτών (εσωτερική ή εξωτερική). Σε αρχεία βιβλιοθηκών μεγάλου όγκου παρατηρήθηκε πρόβλημα στην

κατανάλωση μνήμης και στην ταχύτητα έρευνας τους. Επίσης, παρατηρήθηκε πρόβλημα στην μεταγλώττιση ορισμένων εντολών στη γλώσσα ενδιάμεσης αναπαράστασης, σε κάποια αρχεία αρχιτεκτονικής arm x32 και arm x64.

Μελλοντική έρευνα θα μπορούσε να προσθέσει επιπλέον ελέγχους, υποσενάρια και περιπτώσεις ευπαθειών ασφαλείας στο πεδίο έρευνας του εργαλείου, καθώς και να βελτιώσει τα ήδη υπάρχων αποτελέσματα. Η ανάπτυξη ανάγνωσης αρχείου που περιέχει υποσεναρία και την διασύνδεση αυτών που απαρτίζουν ένα σενάριο ασφαλείας είναι εύκολο να υλοποιηθεί, λόγο του διαχωρισμού των υποσεναρίων που έχει ήδη γίνει κατά την ανάπτυξη του εργαλείου και θα διευκόλυνε την κατασκευή σεναρίων ευπαθειών ασφαλείας.

Το εργαλείο εύρεσης ευπαθειών ασφαλείας καλύπτει επιτυχώς τα σενάρια που ερευνήθηκαν, όπως επιβεβαιώθηκε κατά την πειραματική αξιολόγηση, καθώς επίσης, εξυπηρετεί και σκοπούς ανάλυσης αρχείων βιβλιοθηκών με την χρήση των βοηθητικών συναρτήσεων. Η χρήση του εργαλείου συμβάλει στον έλεγχο ασφαλείας βιβλιοθηκών που πρόκειται να χρησιμοποιηθούν για την ανάπτυξη εφαρμογών, δίχως την ανάγκη του πηγαίου κώδικα των βιβλιοθηκών αυτών, μειώνοντας έτσι τις πιθανότητες ευπαθειών ασφαλείας που ενδέχεται να προκύψουν λόγο χρήσης εξωτερικών βιβλιοθηκών στις εφαρμογές αυτές.

Παράρτημα

carrion_base.py

```
#!/usr/bin/env python

from pybfd.bfd import *
from pybfd.opcodes import *
from pybfd.section import *
from pybfd.symbol import *

import pyvex
import archinfo
import binascii
import ctypes

from io import BytesIO
from sys import argv
from sys import getsizeof

class ExternalFunction:

    def __init__(self, name, library):
        self.name = name
        self.library = library
        self.addresses = []

    def name(self):
        return self._name

    def library(self):
        return self._library

    def addresses(self):
        return self._addresses

class FunctionSymbol:

    def __init__(self, name, address, section_name, size=0,
stack_size=0):
        self.name = name
        self.address = address
        self.section_name = section_name
        self.size = size
        self.stack_size = stack_size
        self.external_functions = {}
        self.stack = {}

    def name(self):
        return self._name

    def address(self):
        return self._address

    def section_name(self):
        return self._section_name
```

```
    def size(self):
        return self._size

    def stack_size(self):
        return self._stack_size

    def external_functions(self):
        return self._external_functions

    def stack(self):
        return self._stack

def getInt(bytestream, start, end, step=1):
    return int(binascii.hexlify(bytearray(bytestream[y] for y in
range(start, end, step))), 16)

def hasNumbers(inputString):
    return all(char.isdigit() for char in inputString)

def isHexString(inputString):
    return any(char.isdigit() for char in inputString)

def findFunctionSizeByAddress(bfd, bytestream, address):

    dynsym = bfd.sections['.dynsym']

    if "elf32" in bfd.target:
        for entry in range(0, dynsym.size, 16):

            if bfd.little_endian:
                value = getInt(bytestream, dynsym.file_offset+entry+7,
dynsym.file_offset+entry+3, -1)
            elif bfd.big_endian:
                value = getInt(bytestream, dynsym.file_offset+entry+4,
dynsym.file_offset+entry+8)
            if value == address or value == (address +1):
                if bfd.little_endian:
                    size = getInt(bytestream,
dynsym.file_offset+entry+11, dynsym.file_offset+entry+7, -1)
                elif bfd.big_endian:
                    size = getInt(bytestream,
dynsym.file_offset+entry+8, dynsym.file_offset+entry+12)
                return size
    elif "elf64" in bfd.target:
        for entry in range(0, dynsym.size, 24):

            if bfd.little_endian:
                value = getInt(bytestream,
dynsym.file_offset+entry+15, dynsym.file_offset+entry+7, -1)
            elif bfd.big_endian:
                value = getInt(bytestream, dynsym.file_offset+entry+8,
dynsym.file_offset+entry+16)

            if value == address:
                if bfd.little_endian:
                    size = getInt(bytestream,
dynsym.file_offset+entry+19, dynsym.file_offset+entry+15, -1)
                elif bfd.big_endian:
```

```
                    size = getInt(bytestream,
dynsym.file_offset+entry+16, dynsym.file_offset+entry+20)

                return size
    return 0

def load_value_from_offset(bytestream, offset):
    if not isinstance(offset, int):
        offset = int(offset, 16)
    return '%x' %getInt(bytestream, offset+3, offset-1, -1)

def load_value_from_address(bfd, bytestream, address):
    if not isinstance(address, int):
        address = int(address, 16)

    for section_name in bfd.sections:
        if address >= bfd.sections[section_name].vma and address <=
bfd.sections[section_name].vma + bfd.sections[section_name].size:
            return load_value_from_offset(bytestream,
bfd.sections[section_name].file_offset + (address -
bfd.sections[section_name].vma))
    return "0x0"

def fetch_target_register(stmt): #left register
    if isinstance(stmt, pyvex.IRStmt.IMark) or isinstance(stmt,
pyvex.IRStmt.NoOp) or isinstance(stmt, pyvex.IRStmt.AbiHint) or
isinstance(stmt, pyvex.IRStmt.MBE):
        return '-1'
    elif isinstance(stmt, pyvex.IRStmt.Put):
        return stmt.arch.translate_register_name(stmt.offset)
    elif isinstance(stmt, pyvex.IRStmt.PutI):
        return stmt.descr
    elif isinstance(stmt, pyvex.IRStmt.WrTmp) or isinstance(stmt,
pyvex.IRStmt.Dirty):
        return 't%s' % stmt.tmp
    elif isinstance(stmt, pyvex.IRStmt.Store) or isinstance(stmt,
pyvex.IRStmt.StoreG):
        return stmt.addr.__str__()
    elif isinstance(stmt, pyvex.IRStmt.CAS):
        return  't(%s,%s)' %(stmt.oldLo, stmt.oldHi)
    elif isinstance(stmt, pyvex.IRStmt.LLSC):
        return 't%d' % stmt.result
    elif isinstance(stmt, pyvex.IRStmt.Exit):
        return 't%s' % stmt.arch.translate_register_name(stmt.offsIP)
    elif isinstance(stmt, pyvex.IRStmt.LoadG):
        return 't%d' % stmt.dst

def fetch_source_register(stmt): #right register
    if isinstance(stmt, pyvex.IRStmt.IMark) or isinstance(stmt,
pyvex.IRStmt.NoOp) or isinstance(stmt, pyvex.IRStmt.AbiHint) or
isinstance(stmt, pyvex.IRStmt.MBE):
        return '-1'
    elif isinstance(stmt, pyvex.IRStmt.Put) or isinstance(stmt,
pyvex.IRStmt.PutI) or isinstance(stmt, pyvex.IRStmt.Store) or
isinstance(stmt, pyvex.IRStmt.StoreG):
        return stmt.data.__str__()
    elif isinstance(stmt, pyvex.IRStmt.WrTmp):
```

```
        if isinstance(stmt.data, pyvex.IRExpr.Get) or
isinstance(stmt.data, pyvex.IRExpr.Get):
            return
stmt.data.arch.translate_register_name(stmt.data.offset)
        elif isinstance(stmt.data, pyvex.IRExpr.Load):
            return stmt.data.addr.__str__()
        elif isinstance(stmt.data, pyvex.IRExpr.Binop):
            return [stmt.data.args[0].__str__(),
stmt.data.args[1].__str__()]
        elif isinstance(stmt.data, pyvex.IRExpr.Unop):
            return stmt.data.args[0].__str__()
        else:
            return stmt.data.__str__()
    elif isinstance(stmt, pyvex.IRStmt.CAS):
        return '(%s,%s)' % (stmt.dataLo, stmt.dataHi)
    elif isinstance(stmt, pyvex.IRStmt.LLSC):
        if stmt.storedata is None:
            return stmt.addr.__str__()
        else:
            return stmt.storedata
    elif isinstance(stmt, pyvex.IRStmt.Dirty):
         return ''.join(str(a) for a in stmt.args)
    elif isinstance(stmt, pyvex.IRStmt.Exit):
        return hex(stmt.dst.value)
    elif isinstance(stmt, pyvex.IRStmt.LoadG):
        return [stmt.addr, stmt.alt]

def value_traceback(bfd, bytestream, irsb, stmt_addr, block_num, reg):
    #return the value of reg
    if (stmt_addr == 0 and block_num == 0) or (isinstance(reg, int)):
        return reg

    if isinstance(reg, str):
        if reg.startswith('0x'):
            return int(reg, 16)
    for j in range(block_num, -1, -1):
        for i in range(stmt_addr, -1, -1):
            stmt = irsb[j].statements[i]
            if reg == fetch_target_register(stmt):
                if isinstance(stmt, pyvex.IRStmt.WrTmp):
                    if isinstance(stmt.data, pyvex.IRExpr.Load):
                        load_address = value_traceback(bfd,
bytestream, irsb, i, j, fetch_source_register(stmt))
                        if load_address == -1:
                            return -1
                        reg = int(load_value_from_address(bfd,
bytestream, load_address),16)
                        if reg == 0:
                            return -1
                    elif isinstance(stmt.data, pyvex.IRExpr.Binop):
                        source = fetch_source_register(stmt)
                        source[0] = value_traceback(bfd, bytestream,
irsb, i-1, j, source[0])
                        source[1] = value_traceback(bfd, bytestream,
irsb, i-1, j, source[1])
                        if isinstance(source[0], str):
                            if not source[0].startswith('0x'):
                                return -1
```

```
                            else:
                                source[0] = int(source[0], 16)
                        if isinstance(source[1], str):
                            if not source[1].startswith('0x'):
                                return -1
                            else:
                                source[1] = int(source[1], 16)

                        if source[0] == -1 or source[1] == -1:
                            return -1

                        if 'Add' in stmt.data.op:
                            return source[0] + source[1]
                        elif 'Sub' in stmt.data.op:
                            return source[0] - source[1]
                        elif 'Shl' in stmt.data.op:
                            return source[0] << source[1]
                        #TODO more ops
                    else:
                        reg = fetch_source_register(stmt)
                else:
                    reg = fetch_source_register(stmt)
            if j > 0:
                stmt_addr = len(irsb[j-1].statements) -1
        if isinstance(reg, str):
            if reg.startswith('0x'):
                reg = int(reg, 16)
        if isinstance(reg, int):
            return reg
        else:
            return -1
```

carrion_info.py

```
#!/usr/bin/env python

from pybfd.bfd import *
from pybfd.opcodes import *
from pybfd.section import *
from pybfd.symbol import *

import pyvex
import archinfo
import binascii
import ctypes
import argparse

from io import BytesIO
from sys import argv
from sys import getsizeof

def getInt(bytestream, start, end, step=1):
    return int(binascii.hexlify(bytearray(bytestream[y] for y in
range(start, end, step))), 16)

def hasNumbers(inputString):
    return all(char.isdigit() for char in inputString)
```

```
def isHexString(inputString):
    return any(char.isdigit() for char in inputString)

def dumpFileSymbols(bfd):
    # code from objdump.py : class DumpFileSymbols, function
do_action()
    print "[+] Symbols         :"
    for symbol_address in bfd.symbols:
        symbol = bfd.symbols[symbol_address]
        print "\t%08x %15s %15s %08x %s" % (symbol_address,
symbol.section.name, symbol.name, symbol.value, ", ".join( [name for
flag, name in SYMBOL_FLAGS_NAMES_SHORT.iteritems() if flag in
symbol.flags] ))

def disassembleSection(bfd, section_name):
    # code from objdump.py : class disassembleSection, function
do_action()

    #
    # Iterate through every section present.
    #
    opcodes = Opcodes(bfd)

    section = bfd.sections[section_name]

    #
    # Obtain the section content and request its disassembly.
    #
    print "\nDisassembly of section %s at %08x\n" % (section.name,
section.vma)
    #opcodes.initialize_smart_disassemble(section.content,
section.vma)

    for vma, size, disasm in opcodes.disassemble(section.content,
section.vma):
        print "%8x (%d)\t%s" % (vma, size, disasm)

def disassembleSections(bfd):
    # code from objdump.py : class disassembleSections, function
do_action()

    #
    # Iterate through every section present.
    #
    opcodes = Opcodes(bfd)

    for section_name in bfd.sections:
        section = bfd.sections[section_name]
        # Skip non-code sections.
        if (section.flags & (SectionFlags.CODE | \
            SectionFlags.HAS_CONTENTS)) != \
            (SectionFlags.CODE | SectionFlags.HAS_CONTENTS):
            continue
        #
        # Obtain the section content and request its disassembly.
        #
```

```
        print "\nDisassembly of section %s at %08x\n" % (section.name,
section.vma)
        for vma, size, disasm in opcodes.disassemble(section.content,
section.vma):
            print "%8x (%d)\t%s" % (vma, size, disasm)
def dumpSectionInfosFromContent(section):
    #
    # Display its name (we get it from the section instance) and its
index
    # inside the binary file.
    #
    print "[+] Selected section information:"
    print "\tName   : %s" % section.name
    print "\tIndex  : %d" % section.index

    # Dump the section content to a buffer
    content = section.content

    # Display approximate section length.
    length = len(content) / 1024

    if length == 0:
        length = len(content) % 1024
        length_unit = "Bytes"
    else:
        length_unit = "Kbytes"

    print "\tLength : %(length)d %(length_unit)s" % vars()

def dumpSectionInfos(bfd, section_name):
    #
    # Display its name (we get it from the section instance) and its
index
    # inside the binary file.
    #
    section = bfd.sections.get(section_name)
    print "[+] Selected section information:"
    print "\tName   : %s" % section.name
    print "\tIndex  : %d" % section.index

    # Dump the section content to a buffer
    content = section.content

    # Display approximate section length.
    length = len(content) / 1024

    if length == 0:
        length = len(content) % 1024
        length_unit = "Bytes"
    else:
        length_unit = "Kbytes"

    print "\tLength : %(length)d %(length_unit)s" % vars()

def dumpSectionsInfos(bfd):

    for section_name in bfd.sections:
        section = bfd.sections.get(section_name)
```

```
        if not section:
            print "[-] No section \'%s\' available." % section_name
        dumpSectionInfosFromContent(section)

def printDWord(bfd, bytestream, offset, size, entries_num):
    entry_size = size / (entries_num*4)
    count = 0
    num = 0
    print "\nNext Entry %d\n" % num

    for x in range(0,size/4):
        print "%08x:\t%s" % ((x*4+offset),
binascii.hexlify(bytearray(bytestream[y] for y in range(offset+x*4+3,
offset+x*4-1, -1))))

        count+=1
        if count == entry_size:
            num +=1
            print "\nNext Entry %d\n" % num
            count = 0

    print ""
def printRawSection(bfd, bytestream, section_name):
    section = bfd.sections[section_name]
    if not section:
        print "[-] No section \'%s\' available." % section_name

    print "\nRaw bytes of section %s at %08x\n" % (section.name,
section.vma)

    count = 0
    num = 0

    if binascii.hexlify(bytearray(bytestream[5])) == '01': #endian
check
        for x in range(0,section.size/4):
            if count == 0:
                print "\nNext Entry %d\n" % num

            print "%08x(+%08x):\t%s" % (section.vma+x*4, x*4,
binascii.hexlify(bytearray(bytestream[y] for y in
range(section.file_offset+x*4+3, section.file_offset+x*4-1, -1))))

            count+=1
            if count == section.entry_size:
                num +=1
                count = 0

    else:
        for x in range(0,section.size/4):
            print "%08x:\t%s" % (section.vma+x*4,
binascii.hexlify(bytearray(bytestream[y] for y in
range(section.file_offset+x*4, section.file_offset+x*4+4))))

    print ""

def printRawSectionByEntries(bfd, bytestream, section_name,
entries_num):
```

```
    section = bfd.sections[section_name]
    if not section:
        print "[-] No section \'%s\' available." % section_name

    print "\nRaw bytes of section %s at %08x\n" % (section.name,
section.vma)

    entry_size = section.size / (entries_num*4)
    count = 0
    num = 0

    if binascii.hexlify(bytearray(bytestream[5])) == '01': #endian
check
        for x in range(0,section.size/4):
            if count == 0:
                print "\nNext Entry %d\n" % num

            print "%08x(+%08x):\t%s" % (section.vma+x*4, x*4,
binascii.hexlify(bytearray(bytestream[y] for y in
range(section.file_offset+x*4+3, section.file_offset+x*4-1, -1))))

            count+=1
            if count == entry_size:
                num +=1
                count = 0

    else:
        for x in range(0,section.size/4):
            print "%08x:\t%s" % (section.vma+x*4,
binascii.hexlify(bytearray(bytestream[y] for y in
range(section.file_offset+x*4, section.file_offset+x*4+4))))

    print ""

def printRawSectionByEntrySize(bfd, bytestream, section_name,
entry_size):
    section = bfd.sections[section_name]
    if not section:
        print "[-] No section \'%s\' available." % section_name

    print "\nRaw bytes of section %s at %08x\n" % (section.name,
section.vma)

    count = 0
    num = 0

    if binascii.hexlify(bytearray(bytestream[5])) == '01': #endian
check
        for x in range(0,section.size/4):
            if count == 0:
                print "\nNext Entry %d\n" % num

            print "%08x(+%08x):\t%s" % (section.vma+x*4, x*4,
binascii.hexlify(bytearray(bytestream[y] for y in
range(section.file_offset+x*4+3, section.file_offset+x*4-1, -1))))

            count+=1
            if count == entry_size:
```

```
                num +=1
                count = 0

    else:
        for x in range(0,section.size/4):
            print "%08x:\t%s" % (section.vma+x*4,
binascii.hexlify(bytearray(bytestream[y] for y in
range(section.file_offset+x*4, section.file_offset+x*4+4))))

    print ""

def printStringsFromSection(bfd, bytestream, section_name):
    section = bfd.sections[section_name]
    if not section:
        print "[-] No section \'%s\' available." % section_name

    print "\nStrings of section %s at %08x\n" % (section.name,
section.vma)

    start = section.file_offset
    while start < section.file_offset + section.size :
        end = start
        while end < section.file_offset + section.size :
            if bytestream[end] ==  '\0' :
                end +=1
                break
            end +=1
        print "%08x(+%08x):\t%s" % (start, start -
section.file_offset, bytearray(bytestream[y] for y in range(start,
end) ))
        start = end

    print ""

def printStringsFromAddress(bytestream, offset, size):

    print "\nStrings at %08x\n" % offset

    start = offset
    while start < offset + size :
        end = start
        while end < offset + size :
            if bytestream[end] ==  '\0' :
                end +=1
                break
            end +=1
        print "%08x(+%08x):\t%s" % (start, start - offset,
bytearray(bytestream[y] for y in range(start, end) ))
        start = end

    print ""


def main():

    parser = argparse.ArgumentParser()
    parser.add_argument("filename", help="file to be analyzed")
```

```
    parser.add_argument("-f", "--file_symbols", action="store_true",
help="dump file symbols")
    parser.add_argument("-d", "--disas_section", help="disassemble
section(s) seperated with commas(,)")
    parser.add_argument("--disas_all", action="store_true",
help="disassemble all sections")
    parser.add_argument("-i", "--section_info", help="print infos of
section(s) seperated with commas(,)")
    parser.add_argument("--all_sections_info", action="store_true",
help="print infos of all sections")
    parser.add_argument("-b", "--bytes", help="print raw bytes of
section(s)/address(es) seperated with commas(,)")
    parser.add_argument("-e", "--entries", help="number of entries of
section(s) seperated with commas(,) - for raw bytes of a section")
    parser.add_argument("-s", "--entry_size", help="size of entry of
section(s) seperated with commas(,) - for raw bytes of a section")
    parser.add_argument("-S", "--section_size", help="size of
section(s) seperated with commas(,) in HEX - for print strings/raw
bytes from address of a section")
    parser.add_argument("-c", "--strings", help="print strings of
section(s)/address(es) seperated with commas(,)")

    args = parser.parse_args()

    f = open(args.filename, "rb")
    bytestream = []
    try:
        byte = f.read(1)
        while byte != "":
            bytestream.append(byte)
            byte = f.read(1)
    finally:
        f.close()

    bfd = None

    try:
        #
        # Initialize BFD instance.
        # We can either pass a filename or a file descriptor and they
will be used
        # in the same way.
        #
        print "[+] Creating BFD instance..."
        bfd = Bfd(argv[1])
        #bfd2 = DumpFileSymbols(argv[1])

        # Print the file format and in case that its an archive then
just show
        # its files and leave.
        print "[+] File format     : %s" % bfd.file_format_name

        if bfd.is_archive:
            print "[-] List of internal files:"
            #for inner_filename in bfd.archive_filenames:
            #    print "\t%s" % inner_filename

            for inner_bfd in bfd.archive_files:
```

```
            print "\t%-40s - sections : %d - symbols : %s" % \
                (inner_bfd.filename,
                len(inner_bfd.sections),
                len(inner_bfd.symbols))

        # The bfd.close() is executed below in the finally clause.
        #return

    #
    # Display some information about the currently open file.
    #
    print "[+] Architecture    : %s (%d)" % \
        (bfd.architecture_name, bfd.architecture)
    print "[+] BFD target name : %s" % bfd.target
    print "[+] Entry point     : 0x%X" % bfd.start_address
    print "[+] Sections        : %d" % len(bfd.sections)
    print "\t"+"\n\t".join([str(s) for s in bfd.sections])

    if args.file_symbols:
        dumpFileSymbols(bfd)

    if args.disas_section:
        sections = args.disas_section.split(',')
        for section in sections:
            disassembleSection(bfd, section)

    if args.disas_all:
        disassembleSections(bfd)

    if args.section_info:
        sections = args.section_info.split(',')
        for section in sections:
            dumpSectionInfos(bfd, section)

    if args.all_sections_info:
        dumpSectionsInfos(bfd)

    if args.bytes:
        sections = args.bytes.split(',')
        entries = []
        if args.entries:
            entries = args.entries.split(',')
        entry_sizes = []
        if args.entry_size:
            entry_sizes = args.entry_size.split(',')

        section_sizes = []
        if args.section_size:
            section_sizes = args.section_size.split(',')

        counter_entries = 0
        counter_entry_sizes = 0
        counter_section_sizes = 0

        for section in sections:

            if isHexString(section):
                ss = 4
```

```
                    ne = 4
                    if counter_section_sizes < len(section_sizes):
                        ss = int(section_sizes[counter_section_sizes])
                        counter_section_sizes += 1
                    if counter_entries < len(entries):
                        ne = int(entries[counter_entries])
                        counter_entries += 1

                    printDWord(bfd, bytestream, int(section,16), ss,
ne)
                else:
                    if args.entries:
                        ne = 1
                        if counter_entries < len(entries):
                            ne = int(entries[counter_entries])
                            counter_entries += 1
                        printRawSectionByEntries(bfd, bytestream,
section, ne)
                    elif args.entry_size:
                        cs = 1
                        if counter_entry_sizes < len(entry_sizes):
                            cs = int(entry_sizes[counter_entry_sizes])
                            counter_entry_sizes += 1
                        printRawSectionByEntrySize(bfd, bytestream,
section, cs)
                    else:
                        print "\nUpdate of pybfd needed to print any
section by its own entry size"
                        #printRawSection(bfd, bytestream, ".dynsym")
#needs update
                        printRawSectionByEntries(bfd, bytestream,
section, 1)

        if args.strings:
            counter = 0
            sections = args.strings.split(',')
            sizes = []
            if args.section_size:
                sizes = args.section_size.split(',')
            for section in sections:
                if isHexString(section):
                    if counter < len(sizes):
                        printStringsFromAddress(bytestream,
int(section, 16), int(sizes[counter],16))
                        counter += 1
                    else:
                        printStringsFromAddress(bytestream,
int(section, 16), 1)
                else:
                    printStringsFromSection(bfd, bytestream, section)


    except BfdException, err:
        #print_exc()
        print "Error : %s" % err

    finally:
        if bfd:
```

```
            # Check is we're working with an archive and close
archived files
            # before closing the container.
            try:
                # Release inner BFD files in case we're an archive
BFD.
                if bfd.is_archive:
                    [inner_bfd.close() for inner_bfd in
bfd.archive_files]
            except TypeError, err:
                pass

            # Release the current BFD and leave.
            bfd.close()


if __name__ == "__main__":
    main()
```

carrion_arm.py

```
#!/usr/bin/env python

from pybfd.bfd import *
from pybfd.opcodes import *
from pybfd.section import *
from pybfd.symbol import *

import pyvex
import archinfo
import binascii
import ctypes
import collections

from io import BytesIO
from sys import argv

from carrion_base import *

class Carrion_arm:

    def __init__(self):
        self.archinfo = archinfo.ArchARM()

    def archinfo(self):
        return self._archinfo

    def get_n_argument_offset(self, n):
        return (n-1)*4 + 8

    def frame_pointer(self):
        return {'r11':self.archinfo.registers['r11']}

    def stack_pointer(self):
        return {'r13':self.archinfo.registers['r13'],
'sp':self.archinfo.registers['sp']}
```

```
    def get_cmpl_flag(self):
        return [0x1, 0x02]

    def get_pc_thunk_func_offset(self, bfd, bytestream):
        return {}

    def get_return_results_register(self):
        return ['r0']

    def find_n_argument_fp_offset(self, bfd, bytestream, irsb,
stmt_addr, block_num, n):

        found = False
        offset = -1
        while block_num > -1:
            #backtrace phase 1 - tracing arguments
            for j in range(stmt_addr-1, -1, -1):
                stmt = irsb[block_num].statements[j]
                if isinstance(stmt, pyvex.IRStmt.Put):
                    if stmt.offset == self.get_n_argument_offset(n):
                        init_temp = fetch_source_register(stmt)
                        #backtrace phase 2 - tracing read from stack
and offset if exists
                        temp = init_temp
                        s = j
                        while s > 0:
                            s -= 1
                            stmt = irsb[block_num].statements[s]
                            if temp == fetch_target_register(stmt):
                                temp = fetch_source_register(stmt)
                                if not isinstance(temp, str):
                                    offset = temp[1]
                                    if 't' in offset:
                                        k = s
                                        while k > 0:
                                            k -= 1
                                            t_stmt =
irsb[block_num].statements[k]
                                            if offset ==
fetch_target_register(t_stmt):
                                                offset =
fetch_source_register(t_stmt)
                                                if 't' not in offset:
                                                    break
                                    temp = temp[0]
                                if temp in self.frame_pointer():
                                    found = True
                                    break
                        if found:
                            break
                        else:
                            offset = value_traceback(bfd, bytestream,
irsb, j, block_num, init_temp)
                            found = True
                if found:
                    break
            block_num -= 1
            if block_num > -1:
```

```
                stmt_addr = len(irsb[block_num].statements) -1

        if found:
            if isinstance(offset, str):
                return int(offset, 16)
            else:
                return offset
        else:
            return -1

    def scan_function_for_external(self, bfd, bytestream, fsym):

        if fsym.size != 0:

            if '.plt' not in bfd.sections or '.rel.plt' not in
bfd.sections or '.dynsym' not in bfd.sections or '.dynstr' not in
bfd.sections or '.gnu.version' not in bfd.sections or '.gnu.version_r'
not in bfd.sections:
                return fsym

            opcodes = Opcodes(bfd)
            fsym_section = bfd.sections[fsym.section_name]

            plt_start = bfd.sections['.plt'].vma
            plt_end = plt_start + bfd.sections['.plt'].size

            rel_plt_start = bfd.sections['.rel.plt'].file_offset
            rel_plt_end = rel_plt_start +
bfd.sections['.rel.plt'].size
            rel_plt_diff = bfd.sections['.rel.plt'].vma -
rel_plt_start

            dynsym_start = bfd.sections['.dynsym'].file_offset

            dynstr_start = bfd.sections['.dynstr'].file_offset

            gnu_version_start =
bfd.sections['.gnu.version'].file_offset

            gnu_version_r_start =
bfd.sections['.gnu.version_r'].file_offset
            gnu_version_r_end = gnu_version_r_start +
bfd.sections['.gnu.version_r'].size

            plt = {}
            for vma, size, disasm in
opcodes.disassemble(bfd.sections['.plt'].content,
bfd.sections['.plt'].vma):
                plt[vma]=disasm

            sp = {'sp':0}

            for vma, size, disasm in
opcodes.disassemble(fsym_section.content, fsym_section.vma):
                if (vma < fsym.address) or (vma > (fsym.address +
fsym.size)) :
                    continue
```

```
                instr = disasm.split("\t")
                if fsym.stack_size == 0:
                    if instr[0] == "sub" and "#" in instr[1]:
                        fsym.stack_size = int(instr[1].split("#")[1])

                if 'str' in instr[0]:
                    pointer = instr[1].split("[")[1].split("]")[0]
                    offset = 0
                    if ', #' in pointer:
                        tmp = pointer.split(", #")
                        pointer = tmp[0]
                        offset = int(tmp[1])

                    if pointer in sp:
                        fsym.stack[sp[pointer]+offset] = 0
                elif 'add' in instr[0]:
                    target = instr[1].split(", ")[0]
                    if target == 'sp':
                        continue

                    offset_str = instr[1].split(", ")[2]
                    if '#' in offset_str:
                        offset = int(offset_str.split("#")[1], 16)
                    else:
                        if offset_str in sp:
                            offset = sp[offset_str]
                        else:
                            continue

                    source = instr[1].split(", ")[1]
                    if source in sp:
                        offset += sp[source]
                    else:
                        continue

                    if target in sp:
                        sp[target] += offset
                    else:
                        sp[target] = offset

                if instr[0].startswith("bl") and
instr[1].startswith('0x'):
                    call_address = int(instr[1],16)

                    if (call_address > plt_start) and (call_address <
plt_end):
                        add_address = call_address + 4
                        ldr_address = call_address + 8

                        add_instr = plt[add_address].split("x")

                        got_diff = int(add_instr[1], 16)

                        ldr_instr = plt[ldr_address].split("#")
                        got_offset = int(ldr_instr[1].split("]")[0])
                        got_address =  ldr_address + got_offset +
got_diff
```

```
                        for rp in range(rel_plt_start, rel_plt_end,
8):

                            if bfd.little_endian:
                                rp_value = getInt(bytestream, rp+3,
rp-1, -1)
                            elif bfd.big_endian:
                                rp_value = getInt(bytestream, rp,
rp+4)

                            if got_address == rp_value:

                                if bfd.little_endian:

                                    dynsym_index = getInt(bytestream,
rp+7, rp+4, -1)
                                    dynstr_offset = getInt(bytestream,
dynsym_start+dynsym_index*16+3, dynsym_start+dynsym_index*16-1, -1)
                                    gnu_version_value =
getInt(bytestream, gnu_version_start+dynsym_index*2+1,
gnu_version_start+dynsym_index*2-1, -1)

                                elif bfd.big_endian:

                                    dynsym_index = getInt(bytestream,
rp+4, rp+7)
                                    dynstr_offset = getInt(bytestream,
dynsym_start+dynsym_index*16, dynsym_start+dynsym_index*16+4)
                                    gnu_version_value =
getInt(bytestream, gnu_version_start+dynsym_index*2,
gnu_version_start+dynsym_index*2+2)

                                str_end = dynstr_start + dynstr_offset
                                while bytestream[str_end] !=  '\0' :
                                    str_end +=1

                                external_function_name = "%s" %
bytearray(bytestream[y] for y in range(dynstr_start + dynstr_offset,
str_end))

                                for gvr in range(gnu_version_r_start,
gnu_version_r_end, 16):

                                    if bfd.little_endian:
                                        gvr_value = getInt(bytestream,
gvr+7, gvr+5, -1)
                                    elif bfd.big_endian:
                                        gvr_value = getInt(bytestream,
gvr+6, gvr+8)

                                    if gvr_value == gnu_version_value:

                                        if bfd.little_endian:
                                            dynstr_offset =
getInt(bytestream, gvr+11, gvr+7, -1)
                                        elif bfd.big_endian:
                                            dynstr_offset =
getInt(bytestream, gvr+8, gvr+12)
```

```
                                    break

                                str_end = dynstr_start + dynstr_offset
                                while bytestream[str_end] !=  '\0' :
                                    str_end +=1

                                external_function_library = "%s" %
bytearray(bytestream[y] for y in range(dynstr_start + dynstr_offset,
str_end))

                                label =
str(external_function_name+'@'+external_function_library)

                                if label not in
fsym.external_functions:
                                    ext_func =
ExternalFunction(external_function_name, external_function_library)
                                    ext_func.addresses.append(vma)
                                    fsym.external_functions[label] =
ext_func
                                else:

fsym.external_functions[label].addresses.append(vma)
                                break
            fsym.stack[0] = 0
            fsym.stack =
collections.OrderedDict(sorted(fsym.stack.items(), reverse=True))

            prev_var = 0
            for var in fsym.stack:
                fsym.stack[var] = prev_var - var
                prev_var = var

        return fsym

    def print_plt(self, bfd, bytestream):
        if '.plt' not in bfd.sections:
            return

        opcodes = Opcodes(bfd)

        plt_start = bfd.sections['.plt'].vma
        plt_end = plt_start + bfd.sections['.plt'].size

        rel_plt_start = bfd.sections['.rel.plt'].file_offset
        rel_plt_end = rel_plt_start + bfd.sections['.rel.plt'].size
        rel_plt_diff = bfd.sections['.rel.plt'].vma - rel_plt_start

        dynsym_start = bfd.sections['.dynsym'].file_offset

        dynstr_start = bfd.sections['.dynstr'].file_offset

        gnu_version_start = bfd.sections['.gnu.version'].file_offset

        gnu_version_r_start =
bfd.sections['.gnu.version_r'].file_offset
```

```
        gnu_version_r_end = gnu_version_r_start +
bfd.sections['.gnu.version_r'].size

        plt = {}
        for vma, size, disasm in
opcodes.disassemble(bfd.sections['.plt'].content,
bfd.sections['.plt'].vma):
            plt[vma]=disasm

        for call_address in range(plt_start+20, plt_end, 12):
            add_address = call_address + 4
            ldr_address = call_address + 8
            add_instr = plt[add_address].split("x")

            got_diff = int(add_instr[1], 16)

            ldr_instr = plt[ldr_address].split("#")
            got_offset = int(ldr_instr[1].split("]")[0])
            got_address =  ldr_address + got_offset + got_diff

            for rp in range(rel_plt_start, rel_plt_end, 8):

                if bfd.little_endian:
                    rp_value = getInt(bytestream, rp+3, rp-1, -1)
                elif bfd.big_endian:
                    rp_value = getInt(bytestream, rp, rp+4)

                if got_address == rp_value:

                    if bfd.little_endian:

                        dynsym_index = getInt(bytestream, rp+7, rp+4,
-1)
                        dynstr_offset = getInt(bytestream,
dynsym_start+dynsym_index*16+3, dynsym_start+dynsym_index*16-1, -1)
                        gnu_version_value = getInt(bytestream,
gnu_version_start+dynsym_index*2+1, gnu_version_start+dynsym_index*2-
1, -1)

                    elif bfd.big_endian:

                        dynsym_index = getInt(bytestream, rp+4, rp+7)
                        dynstr_offset = getInt(bytestream,
dynsym_start+dynsym_index*16, dynsym_start+dynsym_index*16+4)
                        gnu_version_value = getInt(bytestream,
gnu_version_start+dynsym_index*2, gnu_version_start+dynsym_index*2+2)

                    str_end = dynstr_start + dynstr_offset
                    while bytestream[str_end] !=  '\0' :
                        str_end +=1

                    external_function_name = "%s" %
bytearray(bytestream[y] for y in range(dynstr_start + dynstr_offset,
str_end))

                    for gvr in range(gnu_version_r_start,
gnu_version_r_end, 16):
```

```
                        if bfd.little_endian:
                            gvr_value = getInt(bytestream, gvr+7,
gvr+5, -1)
                        elif bfd.big_endian:
                            gvr_value = getInt(bytestream, gvr+6,
gvr+8)

                        if gvr_value == gnu_version_value:

                            if bfd.little_endian:
                                dynstr_offset = getInt(bytestream,
gvr+11, gvr+7, -1)
                            elif bfd.big_endian:
                                dynstr_offset = getInt(bytestream,
gvr+8, gvr+12)

                            break

                    str_end = dynstr_start + dynstr_offset
                    while bytestream[str_end] !=  '\0' :
                        str_end +=1

                    external_function_library = "%s" %
bytearray(bytestream[y] for y in range(dynstr_start + dynstr_offset,
str_end))

                    label =
str(external_function_name+'@'+external_function_library)

                    print "%d.(%x - %x): %s"%((call_address-plt_start-
20)/12+1,call_address, call_address+8, label)
                    break
```

carrion_aarch64.py

```
#!/usr/bin/env python

from pybfd.bfd import *
from pybfd.opcodes import *
from pybfd.section import *
from pybfd.symbol import *

import pyvex
import archinfo
import binascii
import ctypes
import collections

from io import BytesIO
from sys import argv
from sys import getsizeof

from carrion_base import *

class Carrion_aarch64:

    def __init__(self):
```

```
        self.archinfo = archinfo.ArchAArch64()

    def archinfo(self):
        return self._archinfo

    def get_n_argument_offset(self, n):
        return (n-1)*8 + 16

    def frame_pointer(self):
        return {'x29':self.archinfo.registers['x29']}

    def stack_pointer(self):
        return {'x31':self.archinfo.registers['x31'],
'xsp':self.archinfo.registers['sp'],
'sp':self.archinfo.registers['sp']}

    def get_cmpl_flag(self):
        return [0x1, 0x2, 0x03]

    def get_pc_thunk_func_offset(self, bfd, bytestream):
        return {}

    def get_return_results_register(self):
        return ['x0']

    def find_n_argument_fp_offset(self, bfd, bytestream, irsb,
stmt_addr, block_num, n):

        found = False
        offset = -1
        while block_num > -1:
            #backtrace phase 1 - tracing arguments
            for j in range(stmt_addr-1, -1, -1):
                stmt = irsb[block_num].statements[j]
                if isinstance(stmt, pyvex.IRStmt.Put):
                    if stmt.offset == self.get_n_argument_offset(n):
                        init_temp = fetch_source_register(stmt)
                        #backtrace phase 2 - tracing read from stack
and offset if exists
                        temp = init_temp
                        s = j
                        while s > 0:
                            s -= 1
                            stmt = irsb[block_num].statements[s]
                            if temp == fetch_target_register(stmt):
                                temp = fetch_source_register(stmt)
                                if not isinstance(temp, str):
                                    offset = temp[1]
                                    if 't' in offset:
                                        k = s
                                        while k > 0:
                                            k -= 1
                                            t_stmt =
irsb[block_num].statements[k]
                                            if offset ==
fetch_target_register(t_stmt):
                                                offset =
fetch_source_register(t_stmt)
```

```
                                                    if 't' not in offset:
                                                        break
                                                temp = temp[0]
                                            if temp in self.frame_pointer():
                                                found = True
                                                break
                                    if found:
                                        break
                                    else:
                                        offset = value_traceback(bfd, bytestream,
irsb, j, block_num, init_temp)
                                        found = True
                            if found:
                                break
                        block_num -= 1
                        if block_num > -1:
                            stmt_addr = len(irsb[block_num].statements) -1
                    if found:
                        if isinstance(offset, str):
                            return int(offset, 16)
                        else:
                            return offset
                    else:
                        return -1

    def scan_function_for_external(self, bfd, bytestream, fsym):

        if fsym.size != 0:

            if '.plt' not in bfd.sections or '.rela.plt' not in
bfd.sections or '.dynsym' not in bfd.sections or '.dynstr' not in
bfd.sections or '.gnu.version' not in bfd.sections or '.gnu.version_r'
not in bfd.sections:
                return fsym

            opcodes = Opcodes(bfd)
            fsym_section = bfd.sections[fsym.section_name]

            plt_start = bfd.sections['.plt'].vma
            plt_end = plt_start + bfd.sections['.plt'].size

            rela_plt_start = bfd.sections['.rela.plt'].file_offset
            rela_plt_end = rela_plt_start +
bfd.sections['.rela.plt'].size
            rela_plt_diff = bfd.sections['.rela.plt'].vma -
rela_plt_start

            dynsym_start = bfd.sections['.dynsym'].file_offset

            dynstr_start = bfd.sections['.dynstr'].file_offset

            gnu_version_start =
bfd.sections['.gnu.version'].file_offset

            gnu_version_r_start =
bfd.sections['.gnu.version_r'].file_offset
            gnu_version_r_end = gnu_version_r_start +
bfd.sections['.gnu.version_r'].size
```

```
            plt = {}
            for vma, size, disasm in
opcodes.disassemble(bfd.sections['.plt'].content,
bfd.sections['.plt'].vma):
                plt[vma]=disasm

            sp = {'sp':0}

            for vma, size, disasm in
opcodes.disassemble(fsym_section.content, fsym_section.vma):
                if (vma < fsym.address) or (vma > (fsym.address +
fsym.size)) :
                    continue
                instr = disasm.split("\t")
                if fsym.stack_size == 0:
                    if instr[0] == "stp" and "sp,#" in instr[1]:
                        fsym.stack_size = int(instr[1].split("-
")[1].split("]")[0])
                    elif instr[0] == "sub" and "#" in instr[1]:
                        fss = instr[1].split("#")[1]
                        if ', lsl' in fss:
                            lsl = int(instr[1].split("#")[2])
                            fss = int(fss.split(',')[0], 16)
                            fsym.stack_size = fss << lsl
                        else:
                            fsym.stack_size = int(fss, 16)

                if 'str' in instr[0]:
                    pointer = instr[1].split("[")[1].split("]")[0]
                    offset = 0
                    if ',#' in pointer:
                        tmp = pointer.split(",#")
                        pointer = tmp[0]
                        offset = int(tmp[1])

                    if pointer in sp:
                        fsym.stack[sp[pointer]+offset-fsym.stack_size]
= 0

                elif 'add' in instr[0]:

                    target = instr[1].split(", ")[0]
                    if target == 'sp':
                        continue

                    offset_str = instr[1].split(", ")[2]
                    if '#' in offset_str:
                        offset = int(offset_str.split("#")[1], 16)
                    else:
                        if offset_str in sp:
                            offset = sp[offset_str]
                        else:
                            continue

                    source = instr[1].split(", ")[1]
                    if source in sp:
                        sp[target] = offset + sp[source]
```

```
                if instr[0] == "bl" and instr[1].startswith('0x'):
                    call_address = int(instr[1],16)
                    if (call_address > plt_start) and (call_address <
plt_end):

                        adrp_instr = plt[call_address].split(" ")
                        got_plt_vma = int(adrp_instr[1], 16)
                        ldr_instr = plt[call_address+4].split("#")
                        if len(ldr_instr) > 1:
                            got_plt_offset =
int(ldr_instr[1].split("]")[0])
                        else:
                            got_plt_offset = 0

                        rela_plt_offset = got_plt_vma + got_plt_offset

                        for rp in range(rela_plt_start, rela_plt_end,
24):

                            if bfd.little_endian:
                                rp_value = getInt(bytestream, rp+3,
rp-1, -1)
                            elif bfd.big_endian:
                                rp_value = getInt(bytestream, rp,
rp+4)

                            if rela_plt_offset == rp_value:

                                if bfd.little_endian:

                                    dynsym_index = getInt(bytestream,
rp+15, rp+11, -1)
                                    dynstr_offset = getInt(bytestream,
dynsym_start+dynsym_index*24+3, dynsym_start+dynsym_index*24-1, -1)
                                    gnu_version_value =
getInt(bytestream, gnu_version_start+dynsym_index*2+1,
gnu_version_start+dynsym_index*2-1, -1)

                                elif bfd.big_endian:

                                    dynsym_index = getInt(bytestream,
rp+12, rp+16)
                                    dynstr_offset = getInt(bytestream,
dynsym_start+dynsym_index*24, dynsym_start+dynsym_index*24+4)
                                    gnu_version_value =
getInt(bytestream, gnu_version_start+dynsym_index*2,
gnu_version_start+dynsym_index*2+2)

                                str_end = dynstr_start + dynstr_offset
                                while bytestream[str_end] !=  '\0' :
                                    str_end +=1

                                external_function_name = "%s" %
bytearray(bytestream[y] for y in range(dynstr_start + dynstr_offset,
str_end))
```

```
                                for gvr in range(gnu_version_r_start,
gnu_version_r_end, 16):

                                    if bfd.little_endian:
                                        gvr_value = getInt(bytestream,
gvr+7, gvr+5, -1)
                                    elif bfd.big_endian:
                                        gvr_value = getInt(bytestream,
gvr+6, gvr+8)

                                    if gvr_value == gnu_version_value:

                                        if bfd.little_endian:
                                            dynstr_offset =
getInt(bytestream, gvr+11, gvr+7, -1)
                                        elif bfd.big_endian:
                                            dynstr_offset =
getInt(bytestream, gvr+8, gvr+12)

                                        break

                                str_end = dynstr_start + dynstr_offset
                                while bytestream[str_end] !=  '\0' :
                                    str_end +=1

                                external_function_library = "%s" %
bytearray(bytestream[y] for y in range(dynstr_start + dynstr_offset,
str_end))

                                label =
str(external_function_name+'@'+external_function_library)

                                if label not in
fsym.external_functions:
                                    ext_func =
ExternalFunction(external_function_name, external_function_library)
                                    ext_func.addresses.append(vma)
                                    fsym.external_functions[label] =
ext_func
                                else:

fsym.external_functions[label].addresses.append(vma)
                                break
            fsym.stack[0] = 0
            fsym.stack =
collections.OrderedDict(sorted(fsym.stack.items(), reverse=True))

            prev_var = 0
            for var in fsym.stack:
                fsym.stack[var] = prev_var - var
                prev_var = var

        return fsym
```

carrion_i386.py

```
#!/usr/bin/env python

from pybfd.bfd import *
from pybfd.opcodes import *
from pybfd.section import *
from pybfd.symbol import *

import pyvex
import archinfo
import binascii
import ctypes
import collections

from io import BytesIO
from sys import argv
from sys import getsizeof

from carrion_base import *

class Carrion_i386:

    def __init__(self):
        self.archinfo = archinfo.ArchX86()

    def archinfo(self):
        return self._archinfo

    def get_n_argument_offset(self, n):
        return (n-1)*4 + 8

    def frame_pointer(self):
        return {'ebp':self.archinfo.registers['ebp']}

    def stack_pointer(self):
        return {'sp':self.archinfo.registers['sp'],
'esp':self.archinfo.registers['esp']}

    def get_cmpl_flag(self):
        return [0x06]

    def get_pc_thunk_func_offset(self, bfd, bytestream):
        offsets = {}
        for key in bfd.symbols:
            if 'get_pc_thunk' in bfd.symbols[key].name:
                split_name = bfd.symbols[key].name.split('.')
                offsets[key] = 'e'+split_name[len(split_name)-1]
        return offsets

    def get_return_results_register(self):
        return ['eax']

    def find_n_argument_fp_offset(self, bfd, bytestream, irsb,
stmt_addr, block_num, n):
        found = False
        offset = -1
        while block_num > -1:
```

```
            #backtrace phase 1 - tracing arguments
            for j in range(stmt_addr, -1, -1):
                stmt = irsb[block_num].statements[j]
                if isinstance(stmt, pyvex.IRStmt.Store):
                    ptr = fetch_target_register(stmt)
                    k = j -1
                    while k > 0:
                        stmt_ptr = irsb[block_num].statements[k]
                        if not isinstance(ptr, str):
                            ptr = ptr[0]

                        if ptr in self.stack_pointer():
                            if n == 1:
                                temp = fetch_source_register(stmt)
                                #backtrace phase 2 - tracing read from
stack and offset if exists
                                s = j
                                while s > 0:
                                    s -= 1
                                    stmt_off =
irsb[block_num].statements[s]
                                    add32 = False
                                    if isinstance(stmt_off,
pyvex.IRStmt.WrTmp):
                                        if isinstance(stmt_off.data,
pyvex.IRExpr.Binop):
                                            if 'Add32' in
stmt_off.data.op:
                                                add32 = True
                                    if temp ==
fetch_target_register(stmt_off):
                                        temp =
fetch_source_register(stmt_off)
                                        if not isinstance(temp, str):
                                            offset = temp[1]
                                            if 't' in offset:
                                                k = s
                                                while k > 0:
                                                    k -= 1
                                                    t_stmt =
irsb[block_num].statements[k]
                                                    if offset ==
fetch_target_register(t_stmt):
                                                        offset =
fetch_source_register(t_stmt)
                                                        if not
isinstance(offset, str):
                                                            offset[0]
= value_traceback(bfd, bytestream, irsb, k, block_num, offset[0])
                                                            offset[1]
= value_traceback(bfd, bytestream, irsb, k, block_num, offset[1])
                                                            if
offset[0] == -1 or offset[1] == -1:
                                                                return
-1
                                                            if
isinstance(t_stmt.data, pyvex.IRExpr.Binop):
```

```
                                                                    if
'Add32' in t_stmt.data.op:

offset = offset[0] + offset[1]
                                                                    elif
'Sub32' in t_stmt.data.op:

offset = offset[0] - offset[1]
                                                                    elif
'Shl32' in t_stmt.data.op:

offset = offset[0] << offset[1]
                                                                        break
                                                                if 't' not in
offset:
                                                                        break
                                                            if isinstance(offset,
str):
                                                                if
offset.startswith("0x"):
                                                                    offset =
int(offset, 16)
                                                                else:
                                                                    offset =
value_traceback(bfd, bytestream, irsb, s, block_num, temp[1])
                                                                if
isinstance(offset, str):
                                                                    return -1
                                                            if add32:
                                                                if offset >
0xff000000:
                                                                    offset =
0xffffffff - offset + 1
                                                            temp = temp[0]
                                                        if temp in
self.frame_pointer() or temp in self.stack_pointer():
                                                            found = True
                                                            break
                                        break
                                    else:
                                        n -= 1
                                        break
                                else:
                                    if ptr == fetch_target_register(stmt_ptr):
                                        ptr = fetch_source_register(stmt_ptr)
                                k -= 1
                        if found:
                            break
                    block_num -= 1
                    if block_num > -1:
                        stmt_addr = len(irsb[block_num].statements) -1
                if found:
                    if isinstance(offset, str):
                        return int(offset, 16)
                    else:
                        return offset
                else:
                    return -1
```

```
    def scan_function_for_external(self, bfd, bytestream, fsym):

        if fsym.size != 0:
            if '.plt' not in bfd.sections or '.got.plt' not in
bfd.sections or '.rel.plt' not in bfd.sections or '.dynsym' not in
bfd.sections or '.dynstr' not in bfd.sections or '.gnu.version' not in
bfd.sections or '.gnu.version_r' not in bfd.sections:
                return fsym

            opcodes = Opcodes(bfd)
            fsym_section = bfd.sections[fsym.section_name]

            plt_start = bfd.sections['.plt'].vma
            plt_end = plt_start + bfd.sections['.plt'].size

            got_plt_start = bfd.sections['.got.plt'].file_offset
            got_plt_end = got_plt_start +
bfd.sections['.got.plt'].size
            got_plt_diff = bfd.sections['.got.plt'].vma -
got_plt_start

            rel_plt_start = bfd.sections['.rel.plt'].file_offset
            rel_plt_end = rel_plt_start +
bfd.sections['.rel.plt'].size
            rel_plt_diff = bfd.sections['.rel.plt'].vma -
rel_plt_start

            dynsym_start = bfd.sections['.dynsym'].file_offset

            dynstr_start = bfd.sections['.dynstr'].file_offset

            gnu_version_start =
bfd.sections['.gnu.version'].file_offset

            gnu_version_r_start =
bfd.sections['.gnu.version_r'].file_offset
            gnu_version_r_end = gnu_version_r_start +
bfd.sections['.gnu.version_r'].size

            for vma, size, disasm in
opcodes.disassemble(fsym_section.content, fsym_section.vma):
                if (vma < fsym.address) or (vma > (fsym.address +
fsym.size)) :
                    continue
                instr = disasm.split(" ")
                if fsym.stack_size == 0:
                    if instr[0] == "sub":
                        if ',' in instr[4]:
                            ss = instr[4].split(",")[1]
                            if ss.startswith('0x'):
                                fsym.stack_size = int(ss, 16)

                if "mov" in instr[0] and len(instr) == 7:
                    if "[ebp" in instr[6] and "," in instr[6]:
                        ofs = instr[6].split("]")[0].split("[ebp")[1]
                        if ofs.startswith('0x'):
```

```
                            offset = int(ofs, 16)
                            fsym.stack[offset] = 0
                    if instr[0] == "call" and instr[3].startswith("0x"):
                        call_address = int(instr[3],16)
                        if (call_address > plt_start) and (call_address <
plt_end):
                            got_plt_entry = call_address + 6

                            for gp in range(got_plt_start, got_plt_end,
4):

                                if bfd.little_endian:
                                    gp_value = getInt(bytestream, gp+3,
gp-1, -1)
                                elif bfd.big_endian:
                                    gp_value = getInt(bytestream, gp,
gp+4)

                                if got_plt_entry == gp_value:

                                    rel_plt_offset = gp + got_plt_diff

                                    for rp in range(rel_plt_start,
rel_plt_end, 8):

                                        if bfd.little_endian:
                                            rp_value = getInt(bytestream,
rp+3, rp-1, -1)
                                        elif bfd.big_endian:
                                            rp_value = getInt(bytestream,
rp, rp+4)

                                        if rel_plt_offset == rp_value:

                                            if bfd.little_endian:

                                                dynsym_index =
getInt(bytestream, rp+7, rp+4, -1)
                                                dynstr_offset =
getInt(bytestream, dynsym_start+dynsym_index*16+3,
dynsym_start+dynsym_index*16-1, -1)
                                                gnu_version_value =
getInt(bytestream, gnu_version_start+dynsym_index*2+1,
gnu_version_start+dynsym_index*2-1, -1)

                                            elif bfd.big_endian:

                                                dynsym_index =
getInt(bytestream, rp+4, rp+7)
                                                dynstr_offset =
getInt(bytestream, dynsym_start+dynsym_index*16,
dynsym_start+dynsym_index*16+4)
                                                gnu_version_value =
getInt(bytestream, gnu_version_start+dynsym_index*2,
gnu_version_start+dynsym_index*2+2)

                                            str_end = dynstr_start +
dynstr_offset
```

```
                                        while bytestream[str_end] !=
'\0' :
                                            str_end +=1

                                        external_function_name = "%s"
% bytearray(bytestream[y] for y in range(dynstr_start + dynstr_offset,
str_end))

                                        for gvr in
range(gnu_version_r_start, gnu_version_r_end, 16):

                                            if bfd.little_endian:
                                                gvr_value =
getInt(bytestream, gvr+7, gvr+5, -1)
                                            elif bfd.big_endian:
                                                gvr_value =
getInt(bytestream, gvr+6, gvr+8)

                                            if gvr_value ==
gnu_version_value:

                                                if bfd.little_endian:
                                                    dynstr_offset =
getInt(bytestream, gvr+11, gvr+7, -1)
                                                elif bfd.big_endian:
                                                    dynstr_offset =
getInt(bytestream, gvr+8, gvr+12)

                                                break

                                        str_end = dynstr_start +
dynstr_offset
                                        while bytestream[str_end] !=
'\0' :
                                            str_end +=1

                                        external_function_library =
"%s" % bytearray(bytestream[y] for y in range(dynstr_start +
dynstr_offset, str_end))

                                        label =
str(external_function_name+'@'+external_function_library)

                                        if label not in
fsym.external_functions:
                                            ext_func =
ExternalFunction(external_function_name, external_function_library)

ext_func.addresses.append(vma)

fsym.external_functions[label] = ext_func
                                        else:

fsym.external_functions[label].addresses.append(vma)
                                        break
                                break

            fsym.stack[0] = 0
```

```
            fsym.stack =
collections.OrderedDict(sorted(fsym.stack.items(), reverse=True))

            prev_var = 0
            for var in fsym.stack:
                fsym.stack[var] = prev_var - var
                prev_var = var


        return fsym
```

carrion_64.py

```
#!/usr/bin/env python

from pybfd.bfd import *
from pybfd.opcodes import *
from pybfd.section import *
from pybfd.symbol import *

import pyvex
import archinfo
import binascii
import ctypes
import collections

from io import BytesIO
from sys import argv
from sys import getsizeof

from carrion_base import *

class Carrion_64:

    def __init__(self):
        self.archinfo = archinfo.ArchAMD64()

    def archinfo(self):
        return self._archinfo

    def get_n_argument_offset(self, n):
        return 72 - (n-1)*8

    def frame_pointer(self):
        return {'rbp':self.archinfo.registers['rbp']}

    def stack_pointer(self):
        return {'rsp':self.archinfo.registers['rsp'],
'sp':self.archinfo.registers['sp']}

    def get_cmpl_flag(self):
        return [0x07, 0x08, 0x14]

    def get_pc_thunk_func_offset(self, bfd, bytestream):
        return {}

    def get_return_results_register(self):
```

```
        return ['rax']

    def find_n_argument_fp_offset(self, bfd, bytestream, irsb,
stmt_addr, block_num, n):

        found = False
        offset = -1
        while block_num > -1:
            #backtrace phase 1 - tracing arguments
            for j in range(stmt_addr-1, -1, -1):
                stmt = irsb[block_num].statements[j]
                if isinstance(stmt, pyvex.IRStmt.Put):
                    if stmt.offset == self.get_n_argument_offset(n):
                        init_temp = fetch_source_register(stmt)
                        #backtrace phase 2 - tracing read from stack
and offset if exists
                        temp = init_temp
                        s = j
                        while s > 0:
                            s -= 1
                            stmt = irsb[block_num].statements[s]
                            add64 = False
                            if isinstance(stmt, pyvex.IRStmt.WrTmp):
                                if isinstance(stmt.data,
pyvex.IRExpr.Binop):
                                    if 'Add64' in stmt.data.op:
                                        add64 = True
                            if temp == fetch_target_register(stmt):
                                temp = fetch_source_register(stmt)
                                if not isinstance(temp, str):
                                    offset = temp[1]
                                    if 't' in offset:
                                        k = s
                                        while k > 0:
                                            k -= 1
                                            t_stmt =
irsb[block_num].statements[k]
                                            if offset ==
fetch_target_register(t_stmt):
                                                offset =
fetch_source_register(t_stmt)
                                                if not
isinstance(offset, str):
                                                    if
isinstance(t_stmt, pyvex.IRStmt.WrTmp):
                                                        if
isinstance(t_stmt.data, pyvex.IRExpr.Binop):
                                                            if 'And8'
in t_stmt.data.op:
                                                                ofs1 =
value_traceback(bfd, bytestream, irsb, s, block_num, offset[0])
                                                                ofs2 =
value_traceback(bfd, bytestream, irsb, s, block_num, offset[1])
                                                                if
isinstance(ofs1, str):
                                                                    ofs1 = int(ofs1, 16)
```

```
                                                                    if
isinstance(ofs2, str):

ofs2 = int(ofs2, 16)
                                                                    offset
= ofs1 and ofs2
                                                        if isinstance(offset,
str):
                                                            if 't' not in
offset:
                                                                break

                                        if isinstance(offset, str):
                                            if offset.startswith('0x'):
                                                offset = int(offset, 16)
                                            else:
                                                return -1
                                        elif not isinstance(offset, int):
                                            offset = value_traceback(bfd,
bytestream, irsb, s, block_num, temp[1])
                                        if add64:
                                            if offset >
0xffff000000000000:
                                                offset =
0xffffffffffffffff - offset + 1
                                        temp = temp[0]
                                    if temp in self.frame_pointer():
                                        found = True
                                        break
                            if found:
                                break
                            else:
                                offset = value_traceback(bfd, bytestream,
irsb, j, block_num, init_temp)
                                found = True
                    if found:
                        break
                block_num -= 1
                if block_num > -1:
                    stmt_addr = len(irsb[block_num].statements) -1

        if found:
            if isinstance(offset, str):
                return int(offset, 16)
            else:
                return offset
        else:
            return -1

    def scan_function_for_external(self, bfd, bytestream, fsym):

        if fsym.size != 0:

            if '.plt' not in bfd.sections or '.got.plt' not in
bfd.sections or '.rela.plt' not in bfd.sections or '.dynsym' not in
bfd.sections or '.dynstr' not in bfd.sections or '.gnu.version' not in
bfd.sections or '.gnu.version_r' not in bfd.sections:
                return fsym
```

```
            opcodes = Opcodes(bfd)
            fsym_section = bfd.sections[fsym.section_name]

            plt_start = bfd.sections['.plt'].vma
            plt_end = plt_start + bfd.sections['.plt'].size

            got_plt_start = bfd.sections['.got.plt'].file_offset
            got_plt_end = got_plt_start +
bfd.sections['.got.plt'].size
            got_plt_diff = bfd.sections['.got.plt'].vma -
got_plt_start

            rela_plt_start = bfd.sections['.rela.plt'].file_offset
            rela_plt_end = rela_plt_start +
bfd.sections['.rela.plt'].size
            rela_plt_diff = bfd.sections['.rela.plt'].vma -
rela_plt_start

            dynsym_start = bfd.sections['.dynsym'].file_offset

            dynstr_start = bfd.sections['.dynstr'].file_offset

            gnu_version_start =
bfd.sections['.gnu.version'].file_offset

            gnu_version_r_start =
bfd.sections['.gnu.version_r'].file_offset
            gnu_version_r_end = gnu_version_r_start +
bfd.sections['.gnu.version_r'].size

            for vma, size, disasm in
opcodes.disassemble(fsym_section.content, fsym_section.vma):
                if (vma < fsym.address) or (vma > (fsym.address +
fsym.size)) :
                    continue
                instr = disasm.split(" ")
                if fsym.stack_size == 0:
                    if instr[0] == "sub" and instr[4].split(",")[0] ==
self.stack_pointer():
                        fsym.stack_size =
int(instr[4].split(",")[1], 16)

                if "mov" in instr[0] and len(instr) == 7:
                    if 'rbp' in instr[6]:
                        offset = instr[6].split("]")[0]
                        if offset.startswith("[rbp"):
                            offset = offset.split("[rbp")[1]
                            if offset.startswith('0x'):
                                offset = int(offset, 16)
                                fsym.stack[offset] = 0
                if instr[0] == "call" and instr[3].startswith('0x'):
                    if instr[3].startswith("0x"):
                        call_address = int(instr[3],16)
                        if (call_address > plt_start) and
(call_address < plt_end):
                            got_plt_entry = call_address + 6
```

```
                            for gp in range(got_plt_start,
got_plt_end, 8):

                                if bfd.little_endian:
                                    gp_value = getInt(bytestream,
gp+3, gp-1, -1)
                                elif bfd.big_endian:
                                    gp_value = getInt(bytestream, gp,
gp+4)

                                if got_plt_entry == gp_value:

                                    rela_plt_offset = gp +
got_plt_diff

                                    for rp in range(rela_plt_start,
rela_plt_end, 24):

                                        if bfd.little_endian:
                                            rp_value =
getInt(bytestream, rp+3, rp-1, -1)
                                        elif bfd.big_endian:
                                            rp_value =
getInt(bytestream, rp, rp+4)

                                        if rela_plt_offset ==
rp_value:

                                            if bfd.little_endian:

                                                dynsym_index =
getInt(bytestream, rp+15, rp+11, -1)
                                                dynstr_offset =
getInt(bytestream, dynsym_start+dynsym_index*24+3,
dynsym_start+dynsym_index*24-1, -1)
                                                gnu_version_value =
getInt(bytestream, gnu_version_start+dynsym_index*2+1,
gnu_version_start+dynsym_index*2-1, -1)

                                            elif bfd.big_endian:

                                                dynsym_index =
getInt(bytestream, rp+12, rp+16)
                                                dynstr_offset =
getInt(bytestream, dynsym_start+dynsym_index*24,
dynsym_start+dynsym_index*24+4)
                                                gnu_version_value =
getInt(bytestream, gnu_version_start+dynsym_index*2,
gnu_version_start+dynsym_index*2+2)

                                            str_end = dynstr_start +
dynstr_offset
                                            while bytestream[str_end]
!=  '\0' :
                                                str_end +=1
```

```
                                                external_function_name =
"%s" % bytearray(bytestream[y] for y in range(dynstr_start +
dynstr_offset, str_end))

                                                for gvr in
range(gnu_version_r_start, gnu_version_r_end, 16):

                                                    if bfd.little_endian:
                                                        gvr_value =
getInt(bytestream, gvr+7, gvr+5, -1)
                                                    elif bfd.big_endian:
                                                        gvr_value =
getInt(bytestream, gvr+6, gvr+8)

                                                    if gvr_value ==
gnu_version_value:

                                                        if
bfd.little_endian:
                                                            dynstr_offset
= getInt(bytestream, gvr+11, gvr+7, -1)
                                                        elif
bfd.big_endian:
                                                            dynstr_offset
= getInt(bytestream, gvr+8, gvr+12)

                                                        break

                                                str_end = dynstr_start +
dynstr_offset
                                                while bytestream[str_end]
!=  '\0' :
                                                    str_end +=1

                                                external_function_library
= "%s" % bytearray(bytestream[y] for y in range(dynstr_start +
dynstr_offset, str_end))

                                                label =
str(external_function_name+'@'+external_function_library)

                                                if label not in
fsym.external_functions:
                                                    ext_func =
ExternalFunction(external_function_name, external_function_library)

ext_func.addresses.append(vma)

fsym.external_functions[label] = ext_func
                                                else:

fsym.external_functions[label].addresses.append(vma)
                                                break
                                            break
            fsym.stack[0] = 0
            fsym.stack =
collections.OrderedDict(sorted(fsym.stack.items(), reverse=True))
```

```
            prev_var = 0
            for var in fsym.stack:
                fsym.stack[var] = prev_var - var
                prev_var = var

        return fsym
```

carrion.py

```
#!/usr/bin/env python

from pybfd.bfd import *
from pybfd.opcodes import *
from pybfd.section import *
from pybfd.symbol import *

import pyvex
import archinfo
import binascii
import ctypes

from io import BytesIO
from sys import argv
from sys import getsizeof

from carrion_arm import *
from carrion_aarch64 import *
from carrion_i386 import *
from carrion_64 import *
from carrion_info import *
from carrion_base import *

class Carrion:

    def __init__(self, bfd, bytestream):
        self.bfd = bfd
        self.bytestream = bytestream
        self.arch = 0
        self.function_symbols = []
        self.get_pc_thunk_func_offset = {}
        if bfd.target == 'elf32-i386':
            self.arch = Carrion_i386()
            self.get_pc_thunk_func_offset =
self.arch.get_pc_thunk_func_offset(self.bfd, self.bytestream)
        elif bfd.target == 'elf64-x86-64':
            self.arch = Carrion_64()
        elif bfd.target == 'elf32-littlearm':
            self.arch = Carrion_arm()
        elif bfd.target == 'elf64-littleaarch64':
            self.arch = Carrion_aarch64()
        else:
            print "Unsupported Architecture"

        for symbol_address in self.bfd.symbols:
            symbol = self.bfd.symbols[symbol_address]
            for flag, name in SYMBOL_FLAGS_NAMES_SHORT.iteritems():
                if flag in symbol.flags and name == 'FUNCTION':
```

```
                    for flag, name in
SYMBOL_FLAGS_NAMES_SHORT.iteritems():
                        if flag in symbol.flags and name == 'EXPORT':
                            fsym = FunctionSymbol(symbol.name,
symbol_address, symbol.section.name,
findFunctionSizeByAddress(self.bfd, self.bytestream, symbol_address))
                            fsym =
self.arch.scan_function_for_external(self.bfd, self.bytestream, fsym)
                            self.function_symbols.append(fsym)

    def bfd(self):
        return self._bfd

    def bytestream(self):
        return self._bytestream

    def function_symbols(self):
        return self._function_symbols

    def arch(self):
        return self._arch

    def get_pc_thunk_func_offset(self):
        return self._get_pc_thunk_func_offset

    def get_n_argument_offset(self, n):
        return self.arch.get_n_argument_offset(n)

    def frame_pointer(self):
        return self.arch.frame_pointer()

    def stack_pointer(self):
        return self.arch.stack_pointer()

    def is_frame_pointer(self, offset):
        for fp in self.arch.frame_pointer():
            if offset == self.arch.frame_pointer()[fp]:
                return True
        return False

    def is_stack_pointer(self, offset):
        for sp in self.arch.stack_pointer():
            if offset == self.arch.stack_pointer()[sp][0]:
                return True
        return False

    def get_cc_op_offset(self):
        return self.arch.archinfo.registers['cc_op'][0]

    def get_cc_dep1_offset(self):
        return self.arch.archinfo.registers['cc_dep1'][0]

    def get_cc_dep2_offset(self):
        return self.arch.archinfo.registers['cc_dep2'][0]

    def get_cc_ndep_offset(self):
        return self.arch.archinfo.registers['cc_ndep'][0]
```

```
    def get_cmpl_flag(self):
        return self.arch.get_cmpl_flag()

    def find_n_argument_fp_offset(self, irsb, stmt_addr, block_num,
n):
        return self.arch.find_n_argument_fp_offset(self.bfd,
self.bytestream, irsb, stmt_addr, block_num, n)

    def get_return_results_register(self):
        return self.arch.get_return_results_register()

    def print_function_symbols(self):
        print "\n%10s\t%8s\t%10s\t%s\t%s" % ("Name", "Address",
"Section", "Size", "Stack Size")
        for fsym in self.function_symbols:
            print "\n[+]%10s\t%08x\t%10s\t%d\t%d:\n" % (fsym.name,
fsym.address, fsym.section_name, fsym.size, fsym.stack_size)
            for ext_func_label in fsym.external_functions:
                ext_func = fsym.external_functions[ext_func_label]
                print "%15s@%s at:" % (ext_func.name,
ext_func.library)
                for address in ext_func.addresses:
                    print "\t\t%08x" % address
            print "\n\tStack:"
            print "\t\tOffset\tSize"
            for offset in fsym.stack:
                print "\t\t%d:\t%d" %(offset, fsym.stack[offset])

    def print_plt(self):
        return self.arch.print_plt(self.bfd, self.bytestream)

def check_get_pc_thunk(carrion, irsb, reg):
    for j in range(len(irsb)):
        for i in range(len(irsb[j].statements)):
            stmt = irsb[j].statements[i]
            if isinstance(stmt, pyvex.IRStmt.Put):
                stmt2 = irsb[j].statements[i-1]
                if isinstance(stmt2, pyvex.IRStmt.Store):
                    if fetch_target_register(stmt) == 'eip':
                        for gpo in carrion.get_pc_thunk_func_offset:
                            if gpo ==
int(fetch_source_register(stmt),16):
                                return
int(fetch_source_register(stmt2), 16)
    return 0

def traceback(carrion, irsb, stmt_addr, block_num, reg):
    #return the value of reg
    if (stmt_addr == 0 and block_num == 0) or (isinstance(reg, int)):
        return reg

    if isinstance(reg, str):
        if reg.startswith('0x'):
            return int(reg, 16)
    for j in range(block_num, -1, -1):
        for i in range(stmt_addr, -1, -1):
            stmt = irsb[j].statements[i]
            if reg == fetch_target_register(stmt):
```

```
                if isinstance(stmt, pyvex.IRStmt.WrTmp):
                    if isinstance(stmt.data, pyvex.IRExpr.Load):
                        load_address = traceback(carrion, irsb, i, j,
fetch_source_register(stmt))
                        if isinstance(load_address, str):
                            return load_address
                        reg = load_value_from_address(carrion.bfd,
carrion.bytestream, load_address)
                        if not isinstance(reg, int):
                            reg = int(reg,16)
                    elif isinstance(stmt.data, pyvex.IRExpr.Binop):
                        source = fetch_source_register(stmt)
                        source[0] = traceback(carrion, irsb, i-1, j,
source[0])
                        source[1] = traceback(carrion, irsb, i-1, j,
source[1])
                        if isinstance(source[0], str):
                            if source[0].startswith('0x'):
                                source[0] = int(source[0], 16)
                            elif source[0] in carrion.frame_pointer()
or source[0] in carrion.stack_pointer():
                                return source[0]
                            else:
                                source[0] =
check_get_pc_thunk(carrion, irsb, source[0])
                        if isinstance(source[1], str):
                            if source[1].startswith('0x'):
                                source[1] = int(source[1], 16)
                            elif source[1] in carrion.frame_pointer()
or source[1] in carrion.stack_pointer():
                                return source[1]
                            else:
                                source[1] =
check_get_pc_thunk(carrion, irsb, source[1])


                        if 'Add64' in stmt.data.op:
                            if source[1] > 0xffff000000000000:
                                return 0xffffffffffffffff - source[0]
- source[1] + 1
                        if 'Add32' in stmt.data.op:
                            if source[1] > 0xff000000:
                                return 0xffffffff - source[0] -
source[1] +1
                        if 'Add' in stmt.data.op:
                            return source[0] + source[1]
                        elif 'Sub' in stmt.data.op:
                            return source[0] - source[1]
                        elif 'Shl' in stmt.data.op:
                            return source[0] << source[1]
                        #TODO more ops
                    else:
                        reg = fetch_source_register(stmt)
                else:
                    reg = fetch_source_register(stmt)
            if j > 0:
                stmt_addr = len(irsb[j-1].statements) -1
        if isinstance(reg, str):
```

```
        if reg.startswith('0x'):
            reg = int(reg, 16)
    return reg

def trace_rodata(carrion, irsb, stmt_addr, block_num, reg):
    if '.rodata' not in carrion.bfd.sections:
        return False
    if isinstance(reg, int):
        if reg >= carrion.bfd.sections['.rodata'].vma and reg <
carrion.bfd.sections['.rodata'].vma +
carrion.bfd.sections['.rodata'].size:
            return True

    if stmt_addr == 0 and block_num == 0:
        return False
    for j in range(block_num, -1, -1):
        for i in range(stmt_addr, -1, -1):
            stmt = irsb[j].statements[i]
            if reg == fetch_target_register(stmt):
                if isinstance(stmt, pyvex.IRStmt.WrTmp):
                    if isinstance(stmt.data, pyvex.IRExpr.Load):
                        load_address = traceback(carrion, irsb, i, j,
fetch_source_register(stmt))
                        if isinstance(load_address, str):
                            return load_address
                        if load_address >=
carrion.bfd.sections['.rodata'].vma and load_address <
carrion.bfd.sections['.rodata'].vma +
carrion.bfd.sections['.rodata'].size:
                            return True
                        reg = load_value_from_address(carrion.bfd,
carrion.bytestream, load_address)
                    elif isinstance(stmt.data, pyvex.IRExpr.Binop):
                        source = fetch_source_register(stmt)
                        source[0] = traceback(carrion, irsb, i-1, j,
source[0])
                        source[1] = traceback(carrion, irsb, i-1, j,
source[1])
                        if isinstance(source[0], str):
                            if source[0].startswith('0x'):
                                source[0] = int(source[0], 16)
                            else:
                                return False
                        if isinstance(source[1], str):
                            if source[1].startswith('0x'):
                                source[1] = int(source[1], 16)
                            else:
                                return False
                        if 'Add' in stmt.data.op:
                            reg = source[0] + source[1]
                        elif 'Sub' in stmt.data.op:
                            reg = source[0] - source[1]
                        elif 'Shl' in stmt.data.op:
                            reg = source[0] << source[1]
                        if reg >= carrion.bfd.sections['.rodata'].vma
and reg < carrion.bfd.sections['.rodata'].vma +
carrion.bfd.sections['.rodata'].size:
                            return True
```

```
                        else:
                            return False
                    else:
                        reg = fetch_source_register(stmt)
                else:
                    reg = fetch_source_register(stmt)
            if j > 0:
                stmt_addr = len(irsb[j-1].statements) -1

    if isinstance(reg, str):
        if reg.startswith('0x'):
            return True
    return reg

def get_fp_offset(carrion, irsb, stmt_addr, block_num, reg):
    #return the offset of the reg from the frame pointer
    if reg in carrion.frame_pointer() or (stmt_addr == 0 and block_num
== 0):
        return reg

    for j in range(block_num, -1, -1):
        for i in range(stmt_addr, -1, -1):
            stmt = irsb[j].statements[i]
            if reg == fetch_target_register(stmt):
                source = fetch_source_register(stmt)
                if isinstance(stmt, pyvex.IRStmt.WrTmp):
                    if isinstance(stmt.data, pyvex.IRExpr.Binop):
                        source = fetch_source_register(stmt)
                        source[0] = get_fp_offset(carrion, irsb, i-1,
j, source[0])
                        source[1] = get_fp_offset(carrion, irsb, i-1,
j, source[1])
                        # if not isinstance(source[0],str) or not
isinstance(source[1], str):
                        #     return -1
                        if source[0] in carrion.frame_pointer():
                            if isinstance(source[1], str):
                                source[1] =
value_traceback(carrion.bfd, carrion.bytestream, irsb, i, j,
source[1])
                            if 'Add64' in stmt.data.op:
                                if source[1] > 0xffff000000000000:
                                    return 0xffffffffffffffff -
source[1] + 1
                            if 'Add32' in stmt.data.op:
                                if source[1] > 0xff000000:
                                    return 0xffffffff - source[1] +1

                            return source[1]
                        elif source[1] in carrion.frame_pointer():
                            if isinstance(source[0], str):
                                source[0] = int(source[0], 16)
                            return source[0]
                        else:
                            if isinstance(source[0], str):
                                if source[0].startswith('0x'):
                                    source[0] = int(source[0], 16)
                                else:
```

```
                                source[0] = 0
                            if isinstance(source[1], str):
                                if source[1].startswith('0x'):
                                    source[1] = int(source[1], 16)
                                else:
                                    source[1] = 0
                            if 'Add64' in stmt.data.op:
                                if source[1] > 0xffff000000000000:
                                    return 0xffffffffffffffff -
source[0] - source[1] + 1
                            if 'Add32' in stmt.data.op:
                                if source[1] > 0xff000000:
                                    return 0xffffffff - source[0] -
source[1] +1
                            if 'Add' in stmt.data.op:
                                return source[0] + source[1]
                            elif 'Sub' in stmt.data.op:
                                return source[0] - source[1]
                            elif 'Shl' in stmt.data.op:
                                return source[0] << source[1]
                            elif 'And' in stmt.data.op:
                                return source[0] & source[1]
                            #TODO more ops
                        else:
                            reg = fetch_source_register(stmt)
                    elif reg in carrion.frame_pointer():
                        return reg
                    else:
                        reg = fetch_source_register(stmt)
            if j > 0:
                stmt_addr = len(irsb[j-1].statements) -1

    if isinstance(reg, str):
        if reg.startswith('0x'):
            reg = int(reg, 16)
    return reg

def is_external_argument_by_fp_offset(carrion, irsb, offset):
    if offset == -1:
        return True
    for x in range(len(irsb)):
        bl = irsb[x]
        for i in range(len(bl.statements)):
            stmt = bl.statements[i]
            source = fetch_source_register(stmt)
            if isinstance(source, str):
                if source in carrion.frame_pointer():
                    stack_reg = fetch_target_register(stmt)
                    stack_reg_offset = -1
                    for j in range(i+1, len(bl.statements)):
                        stmt = bl.statements[j]
                        source = fetch_source_register(stmt)
                        if not isinstance(source, str):
                            if stack_reg == source[0]:
                                pointer = fetch_target_register(stmt)
                                stack_reg_offset = traceback(carrion,
irsb, j, x, source[1])
```

```
                                if stack_reg_offset in
carrion.stack_pointer() or stack_reg_offset in
carrion.frame_pointer():
                                    return True
                                break
                            elif stack_reg == source:
                                stack_reg = fetch_target_register(stmt)
                            if isinstance(stack_reg_offset, str):
                                stack_reg_offset = int(stack_reg_offset)
                            if isinstance(stmt, pyvex.IRStmt.WrTmp):
                                if isinstance(stmt.data,
pyvex.IRExpr.Binop):
                                    if 'Add32' in stmt.data.op:
                                        if stack_reg_offset > 0xff000000:
                                            ptr = get_fp_offset(carrion,
irsb, j, x, source[0])
                                            if ptr in
carrion.frame_pointer():
                                                ptr = 0
                                            stack_reg_offset = 0xffffffff
- ptr- stack_reg_offset +1
                                    elif 'Add64' in stmt.data.op:
                                        if stack_reg_offset >
0xffff000000000000:
                                            ptr = get_fp_offset(carrion,
irsb, j, x, source[0])
                                            if ptr in
carrion.frame_pointer():
                                                ptr = 0
                                            stack_reg_offset =
0xffffffffffffffff - ptr- stack_reg_offset +1
                            if isinstance(stack_reg_offset, str):
                                if 'ITE' in stack_reg_offset:
                                    return False
                                if hasNumbers(stack_reg_offset):
                                    stack_reg_offset = int(stack_reg_offset)
                                elif stack_reg_offset.startswith('0x'):
                                    stack_reg_offset = int(stack_reg_offset,
16)
                                else:
                                    return True

                            if isinstance(stmt, pyvex.IRStmt.WrTmp):
                                if isinstance(stmt.data, pyvex.IRExpr.Binop):
                                    if 'Add32' in stmt.data.op:
                                        if stack_reg_offset > 0xff000000:
                                            ptr = get_fp_offset(carrion, irsb,
j, x, source[0])
                                            if ptr in carrion.frame_pointer():
                                                ptr = 0
                                            stack_reg_offset = 0xffffffff -
ptr- stack_reg_offset +1
                                    elif 'Add64' in stmt.data.op:
                                        if stack_reg_offset >
0xffff000000000000:
                                            ptr = get_fp_offset(carrion, irsb,
j, x, source[0])
                                            if ptr in carrion.frame_pointer():
```

```
                                        ptr = 0
                                    stack_reg_offset =
0xffffffffffffffff - ptr- stack_reg_offset +1
                    if stack_reg_offset == offset:
                        for k in range(j+1, len(bl.statements)):
                            stmt = bl.statements[k]
                            source = fetch_source_register(stmt)
                            if isinstance(source, str):
                                if source == pointer:
                                    pointer =
fetch_target_register(stmt)
                            if isinstance(stmt, pyvex.IRStmt.Store):
                                if pointer ==
fetch_target_register(stmt):
                                    arg = traceback(carrion, irsb, k,
x, fetch_source_register(stmt))
                                    if isinstance(arg, int):
                                        return False
                                    else:
                                        return True
    return False

def is_static_string(carrion, irsb, offset):
    if offset == -1:
        return False
    if trace_rodata(carrion, irsb, len(irsb[len(irsb)-1].statements)-
1, len(irsb)-1, offset) == True:
        return True
    for j in range(len(irsb)):
        for i in range(len(irsb[j].statements)):
            stmt = irsb[j].statements[i]
            if isinstance(stmt, pyvex.IRStmt.Store):
                fp_offset = get_fp_offset(carrion, irsb, i-1, j,
fetch_target_register(stmt))
                if fp_offset == -offset or fp_offset == offset:
                    if trace_rodata(carrion, irsb, i-1, j,
fetch_source_register(stmt)) == True:
                        return True
    return False

def is_static_string_x86(carrion, irsb, stmt_addr, block_num, n):
    if '.rodata' not in carrion.bfd.sections:
        return False
    offset = -1
    while block_num > -1:
        #backtrace phase 1 - tracing arguments
        for j in range(stmt_addr, -1, -1):
            stmt = irsb[block_num].statements[j]
            if isinstance(stmt, pyvex.IRStmt.Store):
                ptr = fetch_target_register(stmt)
                k = j -1
                while k > 0:
                    stmt_ptr = irsb[block_num].statements[k]
                    if not isinstance(ptr, str):
                        ptr = ptr[0]

                    if ptr in carrion.arch.stack_pointer():
                        if n == 1:
```

```
                            reg = fetch_source_register(stmt)
                            for j in range(block_num, -1, -1):
                                for i in range(stmt_addr, -1, -1):
                                    stmt = irsb[j].statements[i]
                                    if reg ==
fetch_target_register(stmt):
                                        if isinstance(stmt,
pyvex.IRStmt.WrTmp):
                                            if isinstance(stmt.data,
pyvex.IRExpr.Load):
                                                load_address =
traceback(carrion, irsb, i, j, fetch_source_register(stmt))
                                                if
isinstance(load_address, str):
                                                    return False
                                                if load_address >=
carrion.bfd.sections['.rodata'].vma and load_address <
carrion.bfd.sections['.rodata'].vma +
carrion.bfd.sections['.rodata'].size:
                                                    return True
                                                reg =
load_value_from_address(carrion.bfd, carrion.bytestream, load_address)
                                            elif isinstance(stmt.data,
pyvex.IRExpr.Binop):
                                                source =
fetch_source_register(stmt)
                                                source[0] =
traceback(carrion, irsb, i-1, j, source[0])
                                                source[1] =
traceback(carrion, irsb, i-1, j, source[1])
                                                if
isinstance(source[0], str):
                                                    if
source[0].startswith('0x'):
                                                        source[0] =
int(source[0], 16)
                                                    else:
                                                        return False
                                                if
isinstance(source[1], str):
                                                    if
source[1].startswith('0x'):
                                                        source[1] =
int(source[1], 16)
                                                    else:
                                                        return False

                                                if 'Add32' in
stmt.data.op:
                                                    if source[1] >
0xff000000:
                                                        reg =
0xffffffff - source[0] - source[1] +1
                                                elif 'Add' in
stmt.data.op:
                                                    reg = source[0] +
source[1]
```

```
                                                elif 'Sub' in
stmt.data.op:
                                                    reg = source[0] -
source[1]
                                                elif 'Shl' in
stmt.data.op:
                                                    reg = source[0] <<
source[1]


                                                if reg < 0:
                                                    reg *= -1
                                                if reg >=
carrion.bfd.sections['.rodata'].vma and reg <
carrion.bfd.sections['.rodata'].vma +
carrion.bfd.sections['.rodata'].size:
                                                    return True
                                                else:
                                                    return False
                                            else:
                                                reg =
fetch_source_register(stmt)
                                        else:
                                            reg =
fetch_source_register(stmt)
                                    if j > 0:
                                        stmt_addr = len(irsb[j-
1].statements) -1
                                #if isinstance(reg, int):
                                if reg >=
carrion.bfd.sections['.rodata'].vma and reg <
carrion.bfd.sections['.rodata'].vma +
carrion.bfd.sections['.rodata'].size:
                                    return True
                                else:
                                    return False
                            else:
                                n -= 1
                                break
                        else:
                            if ptr == fetch_target_register(stmt_ptr):
                                ptr = fetch_source_register(stmt_ptr)
                        k -= 1
            block_num -= 1
            if block_num > -1:
                stmt_addr = len(irsb[block_num].statements) -1
    return False


def check_results(carrion, irsb, stmt_addr, block_num):
    start_stmt = stmt_addr
    counter = 0
    found = False
    for j in range(block_num, len(irsb)):
        for i in range(start_stmt, len(irsb[j].statements)):
            stmt = irsb[j].statements[i]
            if isinstance(stmt, pyvex.IRStmt.IMark):
                counter += 1
                if counter > 20:
```

```
                    break
            if isinstance(stmt, pyvex.IRStmt.Store):
                result_offset = get_fp_offset(carrion, irsb, i-1, j,
fetch_target_register(stmt))
                found = True
            if isinstance(stmt, pyvex.IRStmt.Put) and found == True:
                if stmt.offset == carrion.get_cc_op_offset() and
isinstance(stmt.data, pyvex.expr.Const):
                    if stmt.data.con.value in carrion.get_cmpl_flag():
                        for x in range(j, len(irsb)):
                            for y in range(i,
len(irsb[x].statements)):
                                stmt = irsb[x].statements[y]
                                if isinstance(stmt, pyvex.IRStmt.Put):
                                    if stmt.offset ==
carrion.get_cc_dep1_offset() or stmt.offset ==
carrion.get_cc_dep2_offset():
                                        if result_offset ==
get_fp_offset(carrion, irsb, y-1, x, fetch_source_register(stmt)):
                                            return True
        start_stmt = 0

    #traceback until the begining of the block, which the next instr
after the call, which stores the results in some register.
    #Results are in r0 (arm), x0 (aarch64), eax (x32), rax (x64)
    if block_num+1 < len(irsb):
        for i in range(len(irsb[block_num+1].statements)):
            stmt = irsb[block_num+1].statements[i]
            if isinstance(stmt, pyvex.IRStmt.IMark):
                counter += 1
                if counter > 20:
                    break
            if isinstance(stmt, pyvex.IRStmt.Put):
                if stmt.offset == carrion.get_cc_op_offset():
                    if stmt.data.con.value in carrion.get_cmpl_flag():
                        for y in range(i,
len(irsb[block_num+1].statements)):
                            stmt = irsb[block_num+1].statements[y]
                            if isinstance(stmt, pyvex.IRStmt.Put):
                                if stmt.offset ==
carrion.get_cc_dep1_offset() or stmt.offset ==
carrion.get_cc_dep2_offset():
                                    reg = fetch_source_register(stmt)
                                    for x in range(y, -1, -1):
                                        stmt =
irsb[block_num+1].statements[x]
                                        if reg ==
fetch_target_register(stmt):
                                            reg =
fetch_source_register(stmt)
                                    if reg in
carrion.get_return_results_register():
                                        return True

    return False

def malloc_check(carrion, irsb, ex_func_addr):
    results = []
```

```
    for j in range(len(irsb)):
        for i in range(len(irsb[j].statements)):
            stmt = irsb[j].statements[i]
            if isinstance(stmt, pyvex.IRStmt.IMark):
                if ex_func_addr == stmt.addr:
                    if is_external_argument_by_fp_offset(carrion,
irsb, carrion.find_n_argument_fp_offset(irsb, i, j, 1)):
                        results.append(["Warning", "First argument of
malloc is external"])
                    if not check_results(carrion, irsb, i, j):
                        results.append(["Possible Vulnerability",
"Returned results are not being checked"])

    return results

def strcpy_check(carrion, irsb, ex_func_addr):
    results = []
    for j in range(len(irsb)):
        for i in range(len(irsb[j].statements)):
            stmt = irsb[j].statements[i]
            if isinstance(stmt, pyvex.IRStmt.IMark):
                if ex_func_addr == stmt.addr:
                    if carrion.bfd.target == 'elf32-i386':
                        if not is_static_string_x86(carrion, irsb, i,
j, 2):
                            if not is_static_string(carrion, irsb,
carrion.find_n_argument_fp_offset(irsb, i, j, 2)):
                                results.append(["Warning", "Second
argument of strcpy is external"])
                                if not is_static_string_x86(carrion,
irsb, i, j, 1):
                                    if not is_static_string(carrion,
irsb, carrion.find_n_argument_fp_offset(irsb, i, j, 1)):
                                        results.append(["Possible
Vulnerability", "First argument of strcpy is internal, while second
argument is external"])
                        else:
                            if not is_static_string(carrion, irsb,
carrion.find_n_argument_fp_offset(irsb, i, j, 2)):
                                results.append(["Warning", "Second
argument of strcpy is external"])
                            if not is_static_string(carrion, irsb,
carrion.find_n_argument_fp_offset(irsb, i, j, 1)):
                                results.append(["Possible
Vulnerability", "First argument of strcpy is internal, while second
argument is external"])
    return results

def strncpy_check(carrion, irsb, ex_func_addr):
    results = []
    for j in range(len(irsb)):
        for i in range(len(irsb[j].statements)):
            stmt = irsb[j].statements[i]
            if isinstance(stmt, pyvex.IRStmt.IMark):
                if ex_func_addr == stmt.addr:
                    if carrion.bfd.target == 'elf32-i386':
                        if not is_static_string_x86(carrion, irsb, i,
j, 2):
```

```
                            if not is_static_string(carrion, irsb,
carrion.find_n_argument_fp_offset(irsb, i, j, 2)):
                                results.append(["Warning", "Second
argument of strncpy is external"])
                                if not is_static_string_x86(carrion,
irsb, i, j, 1):
                                    if not is_static_string(carrion,
irsb, carrion.find_n_argument_fp_offset(irsb, i, j, 1)):
                                        results.append(["Possible
Vulnerability", "First argument of strncpy is internal, while second
argument is external"])
                    else:
                        if not is_static_string(carrion, irsb,
carrion.find_n_argument_fp_offset(irsb, i, j, 2)):
                            results.append(["Warning", "Second
argument of strncpy is external"])
                            if not is_static_string(carrion, irsb,
carrion.find_n_argument_fp_offset(irsb, i, j, 1)):
                                results.append(["Possible
Vulnerability", "First argument of strncpy is internal, while second
argument is external"])
    return results

def strcat_check(carrion, irsb, ex_func_addr):
    results = []
    for j in range(len(irsb)):
        for i in range(len(irsb[j].statements)):
            stmt = irsb[j].statements[i]
            if isinstance(stmt, pyvex.IRStmt.IMark):
                if ex_func_addr == stmt.addr:
                    if carrion.bfd.target == 'elf32-i386':
                        if not is_static_string_x86(carrion, irsb, i,
j, 2):
                            if not is_static_string(carrion, irsb,
carrion.find_n_argument_fp_offset(irsb, i, j, 2)):
                                results.append(["Warning", "Second
argument of strcat is external"])
                                if not is_static_string(carrion, irsb,
carrion.find_n_argument_fp_offset(irsb, i, j, 1)):
                                    if not
is_static_string_x86(carrion, irsb, i, j, 1):
                                        results.append(["Possible
Vulnerability", "First argument of strcat is internal, while second
argument is external"])
                    else:
                        if not is_static_string(carrion, irsb,
carrion.find_n_argument_fp_offset(irsb, i, j, 2)):
                            results.append(["Warning", "Second
argument of strcat is external"])
                            if not is_static_string(carrion, irsb,
carrion.find_n_argument_fp_offset(irsb, i, j, 1)):
                                results.append(["Possible
Vulnerability", "First argument of strcat is internal, while second
argument is external"])
    return results

def strncat_check(carrion, irsb, ex_func_addr):
    results = []
```

```
    for j in range(len(irsb)):
        for i in range(len(irsb[j].statements)):
            stmt = irsb[j].statements[i]
            if isinstance(stmt, pyvex.IRStmt.IMark):
                if ex_func_addr == stmt.addr:
                    if carrion.bfd.target == 'elf32-i386':
                        if not is_static_string_x86(carrion, irsb, i,
j, 2):
                            if not is_static_string(carrion, irsb,
carrion.find_n_argument_fp_offset(irsb, i, j, 2)):
                                results.append(["Warning", "Second
argument of strncat is external"])
                                if not is_static_string_x86(carrion,
irsb, i, j, 1):
                                    if not is_static_string(carrion,
irsb, carrion.find_n_argument_fp_offset(irsb, i, j, 1)):
                                        results.append(["Possible
Vulnerability", "First argument of strncat is internal, while second
argument is external"])
                    else:
                        if not is_static_string(carrion, irsb,
carrion.find_n_argument_fp_offset(irsb, i, j, 2)):
                            results.append(["Warning", "Second
argument of strncat is external"])
                            if not is_static_string(carrion, irsb,
carrion.find_n_argument_fp_offset(irsb, i, j, 1)):
                                results.append(["Possible
Vulnerability", "First argument of strncat is internal, while second
argument is external"])
    return results

def sprintf_check(carrion, irsb, ex_func_addr):
    results = []
    for j in range(len(irsb)):
        for i in range(len(irsb[j].statements)):
            stmt = irsb[j].statements[i]
            if isinstance(stmt, pyvex.IRStmt.IMark):
                if ex_func_addr == stmt.addr:
                    if carrion.bfd.target == 'elf32-i386':
                        if not is_static_string_x86(carrion, irsb, i,
j, 2):
                            if not is_static_string(carrion, irsb,
carrion.find_n_argument_fp_offset(irsb, i, j, 2)):
                                results.append(["Warning", "Second
argument of sprintf is external"])
                    else:
                        if not is_static_string(carrion, irsb,
carrion.find_n_argument_fp_offset(irsb, i, j, 2)):
                            results.append(["Warning", "Second
argument of sprintf is external"])
    return results

def snprintf_check(carrion, irsb, ex_func_addr):
    results = []
    for j in range(len(irsb)):
        for i in range(len(irsb[j].statements)):
            stmt = irsb[j].statements[i]
            if isinstance(stmt, pyvex.IRStmt.IMark):
```

```
                if ex_func_addr == stmt.addr:
                    if carrion.bfd.target == 'elf32-i386':
                        if not is_static_string_x86(carrion, irsb, i,
j, 3):
                            if not is_static_string(carrion, irsb,
carrion.find_n_argument_fp_offset(irsb, i, j, 3)):
                                results.append(["Warning", "Third
argument of snprintf is external"])
                    else:
                        if not is_static_string(carrion, irsb,
carrion.find_n_argument_fp_offset(irsb, i, j, 3)):
                            results.append(["Warning", "Third argument
of snprintf is external"])
                    if not check_results(carrion, irsb, i, j):
                        results.append(["Possible Vulnerability",
"Returned results are not being checked"])
    return results

def memcpy_check(carrion, irsb, ex_func_addr):
    results = []
    offset = -1
    for j in range(len(irsb)):
        for i in range(len(irsb[j].statements)):
            stmt = irsb[j].statements[i]
            if isinstance(stmt, pyvex.IRStmt.IMark):
                if ex_func_addr == stmt.addr:
                    if is_external_argument_by_fp_offset(carrion,
irsb, carrion.find_n_argument_fp_offset(irsb, i, j, 2)):
                        results.append(["Warning", "Second argument of
mempcy is external"])
                    if not
is_external_argument_by_fp_offset(carrion, irsb,
carrion.find_n_argument_fp_offset(irsb, i, j, 1)):
                        results.append(["Possible Vulnerability",
"First argument of memcpy is internal, while second argument is
external"])
    return results

def system_check(carrion, irsb, ex_func_addr):
    results = []
    offset = -1
    for j in range(len(irsb)):
        for i in range(len(irsb[j].statements)):
            stmt = irsb[j].statements[i]
            if isinstance(stmt, pyvex.IRStmt.IMark):
                if ex_func_addr == stmt.addr:
                    if carrion.bfd.target == 'elf32-i386':
                        if not is_static_string_x86(carrion, irsb, i,
j, 1):
                            if not is_static_string(carrion, irsb,
carrion.find_n_argument_fp_offset(irsb, i, j, 1)):
                                results.append(["Possible
Vulnerability", "First argument of system is external"])
                    else:
                        if not is_static_string(carrion, irsb,
carrion.find_n_argument_fp_offset(irsb, i, j, 1)):
                            results.append(["Possible Vulnerability",
"First argument of system is external"])
```

```
    return results

def open_check(carrion, irsb, ex_func_addr):
    results = []
    offset = -1
    for j in range(len(irsb)):
        for i in range(len(irsb[j].statements)):
            stmt = irsb[j].statements[i]
            if isinstance(stmt, pyvex.IRStmt.IMark):
                if ex_func_addr == stmt.addr:
                    if carrion.bfd.target == 'elf32-i386':
                        if not is_static_string_x86(carrion, irsb, i,
j, 1):
                            if not is_static_string(carrion, irsb,
carrion.find_n_argument_fp_offset(irsb, i, j, 1)):
                                results.append(["Possible
Vulnerability", "First argument of open is external"])
                    else:
                        if not is_static_string(carrion, irsb,
carrion.find_n_argument_fp_offset(irsb, i, j, 1)):
                            results.append(["Possible Vulnerability",
"First argument of open is external"])
                    if not check_results(carrion, irsb, i, j):
                        results.append(["Possible Vulnerability",
"Returned results are not being checked"])

    return results

def fopen_check(carrion, irsb, ex_func_addr):
    results = []
    offset = -1
    for j in range(len(irsb)):
        for i in range(len(irsb[j].statements)):
            stmt = irsb[j].statements[i]
            if isinstance(stmt, pyvex.IRStmt.IMark):
                if ex_func_addr == stmt.addr:
                    if carrion.bfd.target == 'elf32-i386':
                        if not is_static_string_x86(carrion, irsb, i,
j, 1):
                            if not is_static_string(carrion, irsb,
carrion.find_n_argument_fp_offset(irsb, i, j, 1)):
                                results.append(["Possible
Vulnerability", "First argument of fopen is external"])
                    else:
                        if not is_static_string(carrion, irsb,
carrion.find_n_argument_fp_offset(irsb, i, j, 1)):
                            results.append(["Possible Vulnerability",
"First argument of fopen is external"])
                    if not check_results(carrion, irsb, i, j):
                        results.append(["Possible Vulnerability",
"Returned results are not being checked"])
    return results

def print_results(results, fsym, ex_func, ex_func_addr):
    if len(results) > 0:
        results.sort(key=lambda x: x[0], reverse=True) #first
warnings, then vulnerabilities
        print "\n"
```

```
        print "%s(0x%x): %s(0x%x)" % (fsym.name, fsym.address,
ex_func, ex_func_addr)
        for r in results:
            print "\t%s:\t%s"%(r[0],r[1])

def main():

    parser = argparse.ArgumentParser()
    parser.add_argument("filename", help="file to be analyzed")
    parser.add_argument("-f", "--file_symbols", action="store_true",
help="dump file symbols")
    parser.add_argument("-d", "--disas_section", help="disassemble
section(s) seperated with commas(,)")
    parser.add_argument("--disas_all", action="store_true",
help="disassemble all sections")
    parser.add_argument("-i", "--section_info", help="print infos of
section(s) seperated with commas(,)")
    parser.add_argument("--all_sections_info", action="store_true",
help="print infos of all sections")
    parser.add_argument("-b", "--bytes", help="print raw bytes of
section(s)/address(es) seperated with commas(,)")
    parser.add_argument("-e", "--entries", help="number of entries of
section(s) seperated with commas(,) - for raw bytes of a section")
    parser.add_argument("-s", "--entry_size", help="number of
DWords(32bits) of entry of section(s) seperated with commas(,) - for
raw bytes of a section")
    parser.add_argument("-S", "--section_size", help="size of
section(s) seperated with commas(,) in bytes - for print strings/raw
bytes from address of a section")
    parser.add_argument("-c", "--strings", help="print strings of
section(s)/address(es) seperated with commas(,)")
    parser.add_argument("-E", "--export_functions",
action="store_true", help="print functions and their addresses found
in the file")
    parser.add_argument("-v", "--vex_ir", action="store_true",
help="vex ir translate")
    parser.add_argument("-p", "--print_plt", action="store_true",
help="print functions in plt table")

    args = parser.parse_args()

    f = open(args.filename, "rb")
    bytestream = []
    try:
        byte = f.read(1)
        while byte != "":
            bytestream.append(byte)
            byte = f.read(1)
    finally:
        f.close()

    bfd = None

    try:
        #
        # Initialize BFD instance.
        # We can either pass a filename or a file descriptor and they
will be used
```

```
        # in the same way.
        #
        print "[+] Creating BFD instance..."
        bfd = Bfd(argv[1])
        #bfd2 = DumpFileSymbols(argv[1])

        # Print the file format and in case that its an archive then
just show
        # its files and leave.
        print "[+] File format     : %s" % bfd.file_format_name

        if bfd.is_archive:
            print "[-] List of internal files:"
            #for inner_filename in bfd.archive_filenames:
            #    print "\t%s" % inner_filename

            for inner_bfd in bfd.archive_files:
                print "\t%-40s - sections : %d - symbols : %s" % \
                    (inner_bfd.filename,
                    len(inner_bfd.sections),
                    len(inner_bfd.symbols))

            # The bfd.close() is executed below in the finally clause.
            #return

        #
        # Display some information about the currently open file.
        #
        print "[+] Architecture    : %s (%d)" %
(bfd.architecture_name, bfd.architecture)
        print "[+] BFD target name : %s" % bfd.target
        print "[+] Entry point     : 0x%X" % bfd.start_address
        print "[+] Sections        : %d" % len(bfd.sections)

        if args.file_symbols:
            dumpFileSymbols(bfd)

        if args.disas_section:
            sections = args.disas_section.split(',')
            for section in sections:
                disassembleSection(bfd, section)

        if args.disas_all:
            disassembleSections(bfd)

        if args.section_info:
            sections = args.section_info.split(',')
            for section in sections:
                dumpSectionInfos(bfd, section)

        if args.all_sections_info:
            dumpSectionsInfos(bfd)

        if args.bytes:
            sections = args.bytes.split(',')
            entries = []
            if args.entries:
                entries = args.entries.split(',')
```

```
            entry_sizes = []
            if args.entry_size:
                entry_sizes = args.entry_size.split(',')

            section_sizes = []
            if args.section_size:
                section_sizes = args.section_size.split(',')

            counter_entries = 0
            counter_entry_sizes = 0
            counter_section_sizes = 0

            for section in sections:

                if isHexString(section):
                    ss = 4
                    ne = 4
                    if counter_section_sizes < len(section_sizes):
                        ss = int(section_sizes[counter_section_sizes])
                        counter_section_sizes += 1
                    if counter_entries < len(entries):
                        ne = int(entries[counter_entries])
                        counter_entries += 1

                    printDWord(bfd, bytestream, int(section,16), ss,
ne)
                else:
                    if args.entries:
                        ne = 1
                        if counter_entries < len(entries):
                            ne = int(entries[counter_entries])
                            counter_entries += 1
                        printRawSectionByEntries(bfd, bytestream,
section, ne)
                    elif args.entry_size:
                        cs = 1
                        if counter_entry_sizes < len(entry_sizes):
                            cs = int(entry_sizes[counter_entry_sizes])
                            counter_entry_sizes += 1
                        printRawSectionByEntrySize(bfd, bytestream,
section, cs)
                    else:
                        print "\nUpdate of pybfd needed to print any
section by its own entry size"
                        #printRawSection(bfd, bytestream, ".dynsym")
#needs update
                        printRawSectionByEntries(bfd, bytestream,
section, 1)

        if args.strings:
            counter = 0
            sections = args.strings.split(',')
            sizes = []
            if args.section_size:
                sizes = args.section_size.split(',')
            for section in sections:
                if isHexString(section):
                    if counter < len(sizes):
```

```
                        printStringsFromAddress(bytestream,
int(section, 16), int(sizes[counter]))
                        counter += 1
                    else:
                        printStringsFromAddress(bytestream,
int(section, 16), 1)
                else:
                    printStringsFromSection(bfd, bytestream, section)

        carrion = Carrion(bfd, bytestream)

        if args.export_functions:
            carrion.print_function_symbols()
        if args.print_plt:
            carrion.print_plt()
        if args.vex_ir:
            for fsym in carrion.function_symbols:
                if fsym.size != 0:
                    size_left = fsym.size
                    irsb = []
                    while size_left > 0:
                        irsb.append(pyvex.IRSB("".join(bytestream[y]
for y in range(fsym.address+(fsym.size-size_left),
(fsym.address+fsym.size))), fsym.address+(fsym.size-size_left),
carrion.arch.archinfo))

                        if irsb[len(irsb)-1].size == 0:
                            size_left -= 4
                        else:
                            size_left -= irsb[len(irsb)-1].size

                    print "\n\n\t%s(0x%x + 0x%x)\n" % (fsym.name,
fsym.address, fsym.size)

                    for bl in irsb:
                        for stmt in bl.statements:
                            if isinstance(stmt, pyvex.IRStmt.IMark):
                                print ""
                            print stmt
                    print
"\n================================================="

        for fsym in carrion.function_symbols:
            if fsym.size != 0:
                size_left = fsym.size
                irsb = []
                while size_left > 0:
                    if carrion.bfd.target == 'elf32-littlearm' or
carrion.bfd.target == 'elf64-littleaarch64': #arm thumb code
                        ex = 0
                        ex2 = 0
                        if not (fsym.address+fsym.size-size_left)%4 ==
0:
                            ex = 2
                        if size_left < 4:
                            ex2 = 2
                        irsb.append(pyvex.IRSB("".join(bytestream[y]
for y in range(fsym.address+(fsym.size-size_left-ex2),
```

```
(fsym.address+fsym.size))), fsym.address+(fsym.size-size_left)+ex,
carrion.arch.archinfo))
                        if irsb[len(irsb)-1].size == 0:
                            size_left -= 4
                        else:
                            size_left -= irsb[len(irsb)-1].size
                    else:   #intel
                        irsb.append(pyvex.IRSB("".join(bytestream[y]
for y in range(fsym.address+(fsym.size-size_left),
(fsym.address+fsym.size))), fsym.address+(fsym.size-size_left),
carrion.arch.archinfo))
                        if irsb[len(irsb)-1].size == 0:
                            size_left -= 4
                        else:
                            size_left -= irsb[len(irsb)-1].size

                for ex_func in fsym.external_functions:
                    for ex_func_addr in
fsym.external_functions[ex_func].addresses:
                        if
fsym.external_functions[ex_func].library.startswith('GLIBC') or
fsym.external_functions[ex_func].library.startswith('LIBC'):
                            if fsym.external_functions[ex_func].name
== "malloc":
                                print_results(malloc_check(carrion,
irsb, ex_func_addr), fsym, ex_func, ex_func_addr)
                            elif fsym.external_functions[ex_func].name
== "sprintf":
                                print_results(sprintf_check(carrion,
irsb, ex_func_addr), fsym, ex_func, ex_func_addr)
                            elif fsym.external_functions[ex_func].name
== "snprintf":
                                print_results(snprintf_check(carrion,
irsb, ex_func_addr), fsym, ex_func, ex_func_addr)
                            elif fsym.external_functions[ex_func].name
== "memcpy":
                                print_results(memcpy_check(carrion,
irsb, ex_func_addr), fsym, ex_func, ex_func_addr)
                            elif fsym.external_functions[ex_func].name
== "strcpy":
                                print_results(strcpy_check(carrion,
irsb, ex_func_addr), fsym, ex_func, ex_func_addr)
                            elif fsym.external_functions[ex_func].name
== "strncpy":
                                print_results(strncpy_check(carrion,
irsb, ex_func_addr), fsym, ex_func, ex_func_addr)
                            elif fsym.external_functions[ex_func].name
== "strcat":
                                print_results(strcat_check(carrion,
irsb, ex_func_addr), fsym, ex_func, ex_func_addr)
                            elif fsym.external_functions[ex_func].name
== "strncat":
                                print_results(strncat_check(carrion,
irsb, ex_func_addr), fsym, ex_func, ex_func_addr)
                            elif fsym.external_functions[ex_func].name
== "system":
                                print_results(system_check(carrion,
irsb, ex_func_addr), fsym, ex_func, ex_func_addr)
```

```
                            elif fsym.external_functions[ex_func].name
== "open":
                                print_results(open_check(carrion,
irsb, ex_func_addr), fsym, ex_func, ex_func_addr)
                            elif fsym.external_functions[ex_func].name
== "fopen":
                                print_results(fopen_check(carrion,
irsb, ex_func_addr), fsym, ex_func, ex_func_addr)


        print ""


    except BfdException, err:
        #print_exc()
        print "Error : %s" % err

    finally:
        if bfd:
            # Check is we're working with an archive and close
archived files
            # before closing the container.
            try:
                # Release inner BFD files in case we're an archive
BFD.
                if bfd.is_archive:
                    [inner_bfd.close() for inner_bfd in
bfd.archive_files]
            except TypeError, err:
                pass

            # Release the current BFD and leave.
            bfd.close()

    print "Analysis finished successfully"

if __name__ == "__main__":
    main()
```

9 Βιβλιογραφία

[1] "Static Code Analysis - OWASP," [Online]. Available: https://www.owasp.org/index.php/Static_Code_Analysis.

[2] B. A. Wichmann, A. A. Canning, D. L. Clutterbuck, L. A. Winsborrow, N. J. Ward and D. W. R. M. Marsh, "Industrial perspective on static," *Software Engineering Journal,* vol. 10, pp. 69-75, 1995.

[3] "Source Code Analysis Tools - OWASP," [Online]. Available: https://www.owasp.org/index.php/Source_Code_Analysis_Tools.

[4] "Supported SAST and DAST Tools for Code Dx," [Online]. Available: https://codedx.com/supported-tools/.

[5] A. Gosain and G. Sharma, "A Survey of Dynamic Program Analysis Techniques and Tools," 2014.

[6] S. Anand, P. Godefroid and N. Tillmann, "Demand-Driven Compositional Symbolic Execution," 2008.

[7] K.-K. Ma, Y. P. Khoo, J. S. Foster and M. W. Hicks, "Directed Symbolic Execution," 2011.

[8] "Symbolic execution - Wikipedia," [Online]. Available: https://en.wikipedia.org/wiki/Symbolic_execution.

[9] "GitHub - angr/pyvex: Python bindings for Valgrind's VEX IR," [Online]. Available: https://github.com/angr/pyvex.

[10] Y. Shoshitaishvili, R. Wang, C. Hauser, C. Kruegel and G. Vigna, "Firmalice - Automatic Detection of Authentication Bypass Vulnerabilities in Binary Firmware," 2015.

[11] "zynamics BinNavi 5.0 Manual - REIL Specification," [Online]. Available: https://www.zynamics.com/binnavi/manual/html/reil_language.htm.

[12] "The REIL language - Part I | blog.zynamics.com," [Online]. Available: https://blog.zynamics.com/2010/03/07/the-reil-language-part-i/.

[13] "The REIL language - Part II | blog.zynamics.com," [Online]. Available: https://blog.zynamics.com/2010/06/22/the-reil-language-part-ii/.

[14] "The REIL language - Part III | blog.zynamics.com," [Online]. Available: https://blog.zynamics.com/2010/07/19/the-reil-language-part-iii/.

[15] "The REIL language - Part IV | blog.zynamics.com," [Online]. Available: https://blog.zynamics.com/2010/08/24/the-reil-language-part-iv/.

[16] D. Thomas and P. Sebastian, "REIL: A platform-independent intermediate representation," 2009.

[17] "GitHub - Cr4sh/openreil: Open source library that implements translator and tools for REIL (Reverse Engineering Intermediate Language)," [Online]. Available: https://github.com/Cr4sh/openreil.

[18] "The GNU C Library," [Online]. Available: https://www.gnu.org/software/libc/documentation.html.

[19] "The GNU C Library - GNU Project - Free Software Foundation (FSF)," [Online]. Available: https://www.gnu.org/software/libc/manual/.

[20] "x86 Assembly Language Reference Manual," [Online]. Available: https://docs.oracle.com/cd/E19253-01/817-5477/817-5477.pdf.

[21] "Intel® 64 and IA-32 Architectures Software Developer's Manual," [Online]. Available: https://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-software-developer-instruction-set-reference-manual-325383.pdf.

[22] "ARM® Developer Suite Version 1.2 Assembler Guide," [Online]. Available: http://infocenter.arm.com/help/topic/com.arm.doc.dui0068b/DUI0068.pdf.

[23] "Code Injection - OWASP," [Online]. Available: https://www.owasp.org/index.php/Code_Injection.

[24] "CERN Computer Security Information," [Online]. Available: https://security.web.cern.ch/security/recommendations/en/codetools/c.shtml.

[25] "Buffer Overflow - OWASP," [Online]. Available: https://www.owasp.org/index.php/Buffer_Overflow.

[26] "Format string attack - OWASP," [Online]. Available: https://www.owasp.org/index.php/Format_string_attack.

[27] "Command Injection - OWASP," [Online]. Available: https://www.owasp.org/index.php/Command_Injection.

[28] "objdump(1): info from object files - Linux man page," [Online]. Available: https://linux.die.net/man/1/objdump.

[29] "readelf(1): Displays info about ELF files - Linux man page," [Online]. Available: https://linux.die.net/man/1/readelf.

[30] "GitHub - Groundworkstech/pybfd: A Python interface to the GNU Binary File Descriptor (BFD) library.," [Online]. Available: https://github.com/Groundworkstech/pybfd.

[31] "GitHub - angr/archinfo: Classes with architecture-specific information useful to other projects.," [Online]. Available: https://github.com/angr/archinfo.

[32] "The Python Tutorial," [Online]. Available: https://docs.python.org/2/tutorial/.