ΠΟΛΥΤΕΧΝΕΙΟ ΚΡΗΤΗΣ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ & ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Αδρομερής Δυναμική Ανάλυση για Python

Συγγραφέας:
Κωνσταντίνος Φιλόπουλος

Εξεταστική επιτροπή:
Αναπληρωτής Καθηγητής Σωτήριος Ιωαννίδης
Καθηγητής Μιχαήλ Λαγουδάκης
Επίκουρος Καθηγητής Νίκος Βασιλάκης

Δεκέμβριος 2022

Ευχαριστίες

Θα ήθελα να εκφράσω τις βαθύτατες ευχαριστίες μου στον Αναπληρωτή Καθηγητή Σωτήριο Ιωαννίδη, Διευθυντή του Εργαστηρίου Μικροεπεξεργαστών και Υλικού, για την ευκαιρία που μου έδωσε να ασχοληθώ με το αντικείμενο της ασφάλειας λογισμικού. Είμαι επίσης βαθιά υπόχρεος στον Επίκουρο Καθηγητή Νίκο Βασιλάκη, του Brown University, για την τεχνογνωσία που μοιράστηκε μαζί μου μέσω της συνεργασίας μας. Επιπλέον, είμαι εξαιρετικά ευγνώμων στον Πρύτανη, καθηγητή Μιχαήλ Λαγουδάκη για την παρουσία του και τον πολύτιμο χρόνο του για την Επιτροπή Διατριβής μου.

Τέλος, αυτή η προσπάθεια δεν θα ήταν δυνατή χωρίς την υπομονή και την καθοδήγηση του κ. Γρηγόρη Ντουσάκη και τις παρατηρήσεις του κ. Αντρέα Μπροκαλάκη.

Τέλος, θα ήθελα να αναφέρω τη στήριξη και την παρουσία δίπλα μου των γονιών μου και του κατοικίδιού μου, Νάλα, όλες τις μοναχικές ώρες ανάπτυξης του προγράμματος.

Περίληψη

Η δυναμική ανάλυση προγράμματος είναι μια διαδεδομένη τεχνική παρακολούθησης, κατανόησης, και εξαγωγής συμπερασμάτων σχετικά με την ασφαλή χρήση και τα χαρακτηριστικά απόδοσης μιας βιβλιοθήκης, δυνητικά παρεμβαίνοντας στη συμπεριφορά του προγράμματος κατά τη διάρκεια της εκτέλεσής του, παρέχοντας συνεχή πληροφόρηση. Τα υφιστάμενα εργαλεία δυναμικής ανάλυσης συχνά επιφέρουν σημαντικό κόστος εκτέλεσης (runtime overhead) σε ένα πρόγραμμα. Για προγράμματα σε Python τα ήδη εφαρμοζόμενα εργαλεία δεν παρέχουν την δυνατότητα δυναμικής ανάλυσης και οι προγραμματιστές χρησιμοποιούν για δική τους χρήση ad-hoc εφαρμογές. Σε αυτή τη διπλωματική εργασία αναπτύσσεται το πρώτο εργαλείο αδρομερούς δυναμικής ανάλυσης (allow/deny), `PySecu`, πλήρως γραμμένο σε Python με δυνατότητα επιλεκτικής τροποποίησης στοιχείων της αναλυόμενης βιβλιοθήκης. Δεδομένης της ευρείας χρήσης βιβλιοθηκών τρίτων (third-party libraries), η τεχνική αδρομερούς δυναμικής ανάλυσης, επιχειρεί να *εξαργυρώσει* λεπτομέρεια και ακρίβεια με την μείωση του κόστους εκτέλεσης της ανάλυσης ενός προγράμματος. Γίνεται σε επίπεδο πλαισίου των στοιχείων της βιβλιοθήκης, υποστηρίζοντας ωστόσο τις σημαντικές διεργασίες του προγράμματος, διατηρώντας την αρχική λειτουργικότητά τους και χωρίς να απαιτείται τροποποίηση του περιβάλλοντος εκτέλεσης ή των χαρακτηριστικών παραγωγής της δυναμικής γλώσσας. Αξιοποιεί τα χαρακτηριστικά των σύγχρονων δυναμικών γλωσσών όπως η JavaScript, η Lua και η Python, ώστε να τροποποιεί δυναμικά την κάθε αναλυόμενη βιβλιοθήκη εισάγοντας κώδικα του χρήστη στον πηγαίο κώδικά της, πριν φορτωθεί. Η εφαρμογή τού εργαλείου ανάλυσης `PySecu` σε 25 βιβλιοθήκες κατέδειξε ότι επιβάλλει 3x μέσο χρονικό κόστος εκτέλεσης στην χωρίς ανάλυση εκτέλεση των βιβλιοθηκών. Βρίσκεται στην ίδια τάξη μεγέθους με αντίστοιχα εργαλεία άλλων γλωσσών (JavaScript), καθώς και με το ενσωματωμένο εργαλείο ανάλυσης API που παρέχεται από την Python, `sys.settrace`. Εμφανίζει δε πολύ ενθαρρυντικά αποτελέσματα στο χρόνο ανάλυσης που επιβάλλει, σε σχέση με τις αναλύσεις του πλαισίου `DynaPyt` για Python.

Abstract

Dynamic program analysis is a widespread technique for monitoring, understanding, and inferring the safe use and performance characteristics of a library, potentially interfering with the program's behavior during its execution, providing continuous information. Existing dynamic analysis tools often impose significant runtime overhead on a program. For programs in Python, existing tools do not provide dynamic analysis and the programmers use ad-hoc applications for their own use. This thesis proposes, the first allow/deny coarse-grained dynamic analysis tool, `PySecu` fully developed in Python with the ability to selectively transform attributes of the analyzed library. Given the widespread use of third-party libraries, this coarse-grained dynamic analysis technique attempts to trade off detail and accuracy by reducing the runtime overhead of the analysis. It is done at the frame level of the library attributes, while still supporting the original program's semantics, retaining their original functionality and without requiring modification of the program's execution environment or the production features of the dynamic language. It leverages the features of modern dynamic languages such as JavaScript, Lua and Python to dynamically modify each library by injecting user code into its source code before it is loaded. Applying the `PySecu` analysis tool to 25 libraries shows that it imposes 3x average runtime overhead on executing the libraries without analysis. It is in the same order of magnitude as corresponding tools in other languages (JavaScript), as well as the built-in API analysis tool provided by Python, `sys.settrace`. It shows very encouraging results in runtime overhead, in relation to the analyses of the `DynaPyt` framework for Python.

Περιεχόμενα

Κατάλογος Γραφημάτων **6**

1 Εισαγωγή **8**
1.1 Σκοπός . . . 8
1.2 Κεντρική Ιδέα . . . 9
1.3 Συμβολή της Διπλωματικής Εργασίας . . . 9
1.3.1 Επιβεβαίωση συμβατότητας της αδρομερούς ανάλυσης . . . 9
1.3.2 Μελέτη περιπτώσεων (case study) και αξιολόγηση . . . 10
1.3.3 Διευκόλυνση χρηστών Python . . . 10
1.4 Επισκόπηση Διπλωματικής Εργασίας . . . 10

2 Θεωρητικό Πλαίσιο **13**
2.1 Αρθρωτός Προγραμματισμός . . . 13
2.2 Βιβλιοθήκες (Libraries) - Ασφάλεια Λογισμικού . . . 16
2.3 Σύγκριση της Δυναμικής Ανάλυσης με τη Στατική Ανάλυση . . . 17
2.4 Λεπτομερής – Αδρομερής Δυναμική Ανάλυση . . . 18
2.5 Επαναπλαισιοποίηση Βιβλιοθηκών . . . 19
2.5.1 Δομή και Λειτουργία . . . 19
2.5.2 Πλεονεκτήματα εφαρμογής της Επαναπλαισιοποίησης Βιβλιοθηκών . . . 19

3 PySecu **21**
3.1 Επισκόπηση . . . 21
3.2 Συνοπτική Λειτουργία . . . 22
3.3 Διάγραμμα ροής της ανάλυσης . . . 23

4 Υλοποίηση **25**
4.1 Αγκιστροποίηση import . . . 25
4.2 Δημιουργία της ανάλυσης . . . 25

5 Παραδείγματα εφαρμογής του PySecu **39**
5.1 Εφαρμογή στη βιβλιοθήκη left-pad . . . 39
5.1.1 Normal test . . . 39
5.1.2 Εξαγωγή αδειών . . . 40
5.1.3 Επιβολή αδειών . . . 41
5.2 Εφαρμογή στη βιβλιοθήκη pythonbenchmark . . . 42
5.2.1 Δοκιμή χωρίς την ανάλυση . . . 42
5.2.2 Εξαγωγή αδειών . . . 44
5.2.3 Επιβολή αδειών . . . 45

6 Αξιολόγηση **46**
6.1 Εξοπλισμός μετρήσεων . . . 46
6.2 Ανίχνευση προβλημάτων - Ασφάλεια . . . 46
6.3 Επιβολή χρονικού κόστους εκτέλεσης . . . 47
6.4 Στοιχεία ανάλυσης . . . 52
6.4.1 Αποτελεσματικότητα - Αξιοπιστία ανάλυσης . . . 52
6.4.2 Ταυτότητα μετασχηματισμένων χαρακτηριστικών . . . 52

7 Σύγκριση με άλλα εργαλεία **56**
7.1 DynaPyt για Python . . . 56
7.2 Εργαλεία δυναμικής ανάλυσης άλλων γλωσσών . . . 57

8 Σχετικές εργασίες και Μελλοντικές πρωτοβουλίες **58**

9 Συμπεράσματα **59**

Βιβλιογραφία **60**

Παραρτήματα **64**

A A Python Guide **65**
A.1 General Information . . . 65
A.2 Install Python . . . 65
A.3 Basic Python Commands . . . 65
A.3.1 A Hello World Example . . . 66
A.3.2 Python Program to Check if a Number is Odd or Even . . . 66

B Installing PySecu **67**
B.1 Install PySecu from pip . . . 67
B.2 Install PySecu from GitHub . . . 67

C Πληροφορίες Βιβλιοθηκών **68**
C.1 Ενσωματωμένα (Built-in) στοιχεία . . . 68
C.2 Περιτυλιγμένα (wrapped) στοιχεία . . . 72
C.3 Προσδιορισμός αναλυθέντων βιβλιοθηκών . . . 76

Κατάλογος Γραφημάτων

2.1 **Περιτυλίγματα (Wrappers)/Αρθρωτός προγραμματισμός σε Python.** 15
2.2 **Κακόβουλο λογισμικό (Malware) και πιθανώς ανεπιθύμητες εφαρμογές (PUA) ετών 2008-2022.** 17

3.1 **Διαδικασία περιτυλίγματος (wrapping).** 22
3.2 **Διάγραμμα ροής της ανάλυσης PySecu.** 24

6.1 **Σύγκριση χρόνων ανάλυσης βιβλιοθηκών με την original εκτέλεση.** 51

C.1 **Περιτυλιγμένα στοιχεία χρήστη έναντι ενσωματωμένων στην Validus.** 75
C.2 **Περιτυλιγμένα στοιχεία χρήστη έναντι ενσωματωμένων στην pybite.** 75
C.3 **Περιτυλιγμένα στοιχεία χρήστη έναντι ενσωματωμένων στην set-algebra.** 75
C.4 **Συνολικά περιτυλιγμένα στοιχεία χρήστη έναντι ενσωματωμένων.** 76

Κατάλογος Πινάκων

6.1 **Μέσοι χρόνοι ανάλυσης και απλής εκτέλεσης ανά βιβλιοθήκη.** . . . 48
6.2 **Ποσοστιαία χρονική καθυστέρηση ανάλυσης.** 49
6.3 **Συντελεστής χρόνου ανάλυσης επί της απλής εκτέλεσης.** 50
6.4 **Αριθμός των καταχωρημένων προσβάσεων στο αρχείο.** 54

7.1 **Αξιολόγηση PySecu με DynaPyt και άλλες εφαρμογές.** 57

C.1 **Ανάλυση ενσωματωμένων στοιχείων Python με το PySecu** 69
C.2 **Περιτυλιγμένα στοιχεία χρήστη έναντι ενσωματωμένων ανά βιβλιοθήκη.** 72

Κεφάλαιο 1

Εισαγωγή

1.1 Σκοπός

Η σύγχρονη πρόκληση αντιμετώπισης απαιτητικών θεμάτων προγραμματισμού οδηγεί συχνά στην ανάπτυξη πολύπλοκων προγραμμάτων μεγάλου μεγέθους. Για την ανάλυση τους και την εξυπηρέτηση του χρήστη χρησιμοποιείται η μεθοδολογία της στατικής ή της δυναμικής ανάλυσης με εργαλεία που έχουν γραφεί σε πολλές γλώσσες προγραμματισμού.

Ειδικά, η δυναμική ανάλυση προγράμματος είναι μια τεχνική για την παρακολούθηση, κατανόηση και δυναμική παρέμβαση στη συμπεριφορά του προγράμματος ή μέρους βιβλιοθήκης κατά την εκτέλεσή του. Λόγος εφαρμογής της δυναμικής ανάλυσης αποτελεί η εξαγωγή συμπερασμάτων, ο έλεγχος των περιορισμών ασφαλείας, ο εντοπισμός σφαλμάτων και η εξαγωγή χαρακτηριστικών απόδοσης. Τα υπάρχοντα εργαλεία δυναμικής ανάλυσης κατά βάση αυξάνουν το κόστος εκτέλεσης του προγράμματος γι' αυτό συνηθίζεται η εφαρμογή τους να γίνεται offline.

Με αφορμή τα παραπάνω, σκοπός αυτής της διπλωματικής εργασίας είναι η ανάπτυξη εργαλείου δυναμικής ανάλυσης στη σύγχρονη δυναμική γλώσσα `Python`. Η δημοφιλία της `Python` σε συνδυασμό με τα δυναμικά χαρακτηριστικά της αποτελούν βασικό και φιλόδοξο στόχο της δυναμικής ανάλυσης. Παρότι αρκετές προτάσεις έχουν κατά καιρούς διατυπωθεί, δεν υπάρχει αξιόλογη πρόοδος σε αυτή την κατεύθυνση, όπως σε άλλες επίσης δημοφιλείς σύγχρονες δυναμικές γλώσσες προγραμματισμού. Η δυναμική ανάλυση για `Python` δεν έχει γίνει ακόμα δημοφιλής mainstream. Το εργαλείο που θα αναπτυχθεί, στηρίζεται σε μια νέα τεχνική αδρομερούς δυναμικής ανάλυσης, η οποία δεν στοχεύει στην αντικατάσταση ή παράβλεψη υφιστάμενων διαδικασιών που λειτουργούν με την ακρίβεια και την λεπτομέρεια που υπαγορεύει το αναλυόμενο πρόγραμμα, αλλά αντ 'αυτού παρέχει μια αδρομερή (χονδροειδή) ανάλυση με ενδεχομένως μικρότερη χρονική καθυστέρηση σε σχέση με τα άλλα εργαλεία δυναμικής ανάλυσης. Αναλυτικά η αρχιτεκτονική και οι λεπτομέρειες λειτουργίας αναπτύσσονται στο Κεφάλαιο 3.

Από την άλλη μεριά, επιτακτική είναι η ανάγκη ανάλυσης των βιβλιοθηκών που ευρέως χρησιμοποιούνται σήμερα από τους χρήστες. Η βιβλιοθήκη ως σύνολο αυτοτελών τμημάτων, με διακριτές ονομασίες, που επιτελούν μία συγκεκριμένη εργασία με τη χρήση πολύπλοκων αλγορίθμων, δύσκολων στην αναπαραγωγή, μπορούν να καλούνται κατά βούληση, και πιθανώς επανειλημμένα, ως διακλαδώσεις μέσα σε ένα ευρύτερο εκτελούμενο κώδικα, αποσκοπώντας, μεταξύ των άλλων, στη μείωση της έκτασης του κώδικα. Παραδείγματα εφαρμογής τους είναι η ταξινόμηση ενός πίνακα, η δημιουργία ενός προγράμματος μηχανικής εκμάθησης, ένας πολύπλοκος μαθηματικός υπολογισμός, η δημιουργία ενός παιχνιδιού κλπ. Οι χρήστες, σήμερα, χρησιμοποιούν έτοιμες βιβλιοθήκες από το διαδίκτυο. Όμως, λόγω του γεγονότος ότι οι βιβλιοθήκες είναι δημόσιες και ενδέχεται να προέρχονται από άγνωστες πηγές, υπάρχουν διάφοροι κίνδυνοι και έτσι η αξιοποίησή τους κρίνεται

επισφαλής. Οι κίνδυνοι αυτοί προκύπτουν από το γεγονός ότι οι δημιουργοί και οι προθέσεις τους, είναι άγνωστες και σε πολλές περιπτώσεις μπορεί να αποδειχθούν ύποπτες ή κακόβουλες (§2.2).

Σημείωση: Συνήθως, χρησιμοποιούνται εναλλακτικά και κατά περίπτωση οι παρακάτω όροι:

- βιβλιοθήκη, πακέτο, πρόγραμμα, module, τμήμα, μονάδα, μέρος βιβλιοθήκης
- στοιχείο, χαρακτηριστικό βιβλιοθήκης
- τροποποίηση, μετασχηματισμός, περιτύλιγμα
- προσβάσεις, αλληλεπιδράσεις, άδειες, δικαιώματα
- απλή, original εκτέλεση
- χρόνος ανάλυσης, runtime overhead

1.2 Κεντρική Ιδέα

Στην διπλωματική αυτή εργασία επιχειρείται η ανάπτυξη ενός εργαλείου - `PySecu` – βασισμένου σε μια νέα προσέγγιση της τεχνικής της δυναμικής ανάλυσης, την αδρομερή δυναμική ανάλυση, που στοχεύει στις σύγχρονες δυναμικές γλώσσες προγραμματισμού Python, Lua, Ruby, JavaScript [3, 34], με δυνατότητα επιλεκτικής τροποποίησης των στοιχείων μιας βιβλιοθήκης.

Πιο συγκεκριμένα το εργαλείο PySecu λειτουργεί μειώνοντας την λεπτομέρεια ανάλυσης στο περίγραμμα των βιβλιοθηκών (**Αδρομερής Δυναμική Ανάλυση - Coarse-grained Dynamic Analysis**), δηλαδή περιτυλίγοντας στοιχεία του προγράμματος που δύνανται να ενθυλακώνουν επαναχρησιμοποιούμενη λειτουργικότητα, ώστε να διατηρούν την αρχική τους λειτουργικότητα παρέ-
χοντας πληροφόρηση με συνεχή ρυθμό. Αποτέλεσμα είναι μεταξύ άλλων, η ασφάλεια των βιβλιοθηκών και η πληροφόρηση για μόλυνσή τους από κακόβουλο λογισμικό. Ο τρόπος εφαρμογής και συμπεριφοράς του θα παρουσιαστεί αναλυτικά στα Κεφάλαια 3 και 4.

1.3 Συμβολή της Διπλωματικής Εργασίας

Η παρούσα διπλωματική εργασία αποτελεί μέρος της γενικότερης ερευνητικής εργασίας με την ονομασία *Efficient Module-Level Dynamic Analysis for Dynamic Languages with Module Recontextualization* [40] και ακολουθεί τη μέχρι τώρα έρευνα σε παρόμοια θέματα αδρομερούς δυναμικής ανάλυσης, με αποτέλεσμα την ανάπτυξη του εργαλείου PySecu για Python. Στο χώρο της δυναμικής ανάλυσης βιβλιοθηκών η συμβολή της παρούσας εργασίας αφορά στους παρακάτω τρεις τομείς:

1.3.1 Επιβεβαίωση συμβατότητας της αδρομερούς ανάλυσης

Η εφαρμογή τεχνικής αδρομερούς δυναμικής ανάλυσης σε προγράμματα της Python αποδεικνύεται καταρχάς συμβατή και ικανοποιητικά αποτελεσματική. Το `PySecu` επιλέχθηκε να εφαρμοστεί σε όλες τις συναρτήσεις, κλάσεις και μεθόδους κλάσεων μιας βιβλιοθήκης, λαμβάνοντας υπόψη ότι το κόστος εκτέλεσης που θα επιβληθεί, θα είναι ανάλογο των προς μετασχηματισμό επιλεγέντων στοιχείων. Αποτέλεσμα ήταν όλα τα στοιχεία εκτός από τα σφάλματα, τις εξαιρέσεις και τις διακοπές, να λειτουργούν κανονικά. Όμως, δεν έχει βρεθεί λύση για ειδικές ενσωματωμένες κλάσεις που είναι χρήσιμες για τον διερμηνέα της Python (printers, quitters και helpers) καθώς και μεθόδους εσωτερικά των κλάσεων int, str, dict κ.λπ. Τέλος, το χρονικό κόστος του PySecu

αποδεικνύεται μικρό και ιδιαίτερα στην περίπτωση μικρο-πακέτων, η ανάλυση απαιτεί χιλιοστά του δευτερολέπτου για να ολοκληρώσει μια βιβλιοθήκη, σε σύγκριση με την εκτέλεση της βιβλιοθήκης χωρίς ανάλυση.

1.3.2 Μελέτη περιπτώσεων (case study) και αξιολόγηση

Παρουσιάζεται εφαρμογή του PySecu στη δυναμική ανάλυση δύο εφαρμογών και βιβλιοθηκών της Python και αποδεικνύεται ότι μπορεί να εντοπίσει, εκτός των άλλων, με ακρίβεια ζητήματα εφαρμογής και ανωμαλίες δυναμικού τύπου, που εμφανίζονται μόνο κατά την εκτέλεση του πακέτου.

1.3.3 Διευκόλυνση χρηστών Python

Επισημαίνεται ότι η σύγχρονη δυναμική γλώσσα προγραμματισμού Python είναι γνωστή για την αναγνωσιμότητα και την απλότητά της και χρησιμεύει ως μια καλή εισαγωγική γλώσσα για τους εκκολαπτόμενους προγραμματιστές. Είναι εξαιρετικά σημαντική για την ανάλυση δεδομένων, κερδίζει σε δημοτικότητα, και υιοθετείται πλέον στα εισαγωγικά μαθήματα προγραμματισμού σε πανεπιστημιακό επίπεδο. Η τυπική βιβλιοθήκη της Python είναι πολύ εκτεταμένη, προσφέροντας ένα ευρύ φάσμα λογισμικού. Περιέχει ενσωματωμένες βιβλιοθήκες (γραμμένες σε C ή Python) που παρέχουν πρόσβαση σε λειτουργίες του συστήματος, όπως αρχεία εισόδου/εξόδου που διαφορετικά θα ήταν απρόσιτα στους προγραμματιστές Python, καθώς και βιβλιοθήκες γραμμένες σε Python που παρέχουν τυποποιημένες λύσεις για πολλά προβλήματα, που παρουσιάζονται στον καθημερινό προγραμματισμό.

Με την εφαρμογή αυτής της μεθόδου αδρομερούς δυναμικής ανάλυσης πλήρως στην Python, χωρίς αλλαγή του κώδικα των βιβλιοθηκών, δίνεται η δυνατότητα διευκόλυνσης της ανάλυσης από προγραμματιστές εξοικειωμένους με τη Python. Το PySecu λειτουργεί, όπως αναφέρθηκε, στο περίγραμμα κάθε βιβλιοθήκης περιτυλίγοντας μόνο δομικά στοιχεία (functions, classes κλπ). Με την περιτύλιξη προστίθεται η λειτουργία της αποθήκευσης του πλήρους ονόματος του στοιχείου, ή η χρήση του για επιβολή. Το συγκεκριμένο εργαλείο αυτής της διπλωματικής εργασίας ενδεχομένως σε συνέχεια εργαλείων δυναμικής ανάλυσης που αναπτύσσονται ή θα αναπτυχθούν στην Python, προσφέρει μια σταθερή, αξιόπιστη και σε ασφαλές περιβάλλον προγραμματισμού λύση για τους χρήστες [10].

1.4 Επισκόπηση Διπλωματικής Εργασίας

Τα επόμενα κεφάλαια της εργασίας αυτής και τα περιεχόμενά τους έχουν ως ακολούθως:

Κεφάλαιο 2: Θεωρητικό Πλαίσιο
Σε αυτό το κεφάλαιο, παρουσιάζονται οι βασικές θεωρητικές γνώσεις, που αφορούν:

- Τον Αρθρωτό Προγραμματισμό αναλύοντας τα πλεονεκτήματα και μειονεκτήματα του Αρθρωτού Προγραμματισμού (Modular Programming) και τον τρόπο με τον οποίο μια δυναμική γλώσσα εισάγει μέρη βιβλιοθηκών.
- Τις βιβλιοθήκες (libraries) των οποίων η ευρεία χρήση εγκυμονεί κινδύνους για την ασφάλεια των εφαρμογών.
- Τις διαφορές μεταξύ της Δυναμικής Ανάλυσης και της Στατικής Ανάλυσης καθώς και το ρόλο της Δυναμικής Ανάλυσης (Dynamic Analysis) στην ανάλυση βιβλιοθηκών.
- Την Λεπτομερή (Fine-Grained) και την Αδρομερή (Coarse-grained) Δυναμική Ανάλυση με σχολιασμό των state-of-the-art εργαλείων ανάλυσης που έχουν ήδη αναπτυχθεί.

- Δομή και Λειτουργία της *Επαναπλαισιοποίησης Βιβλιοθηκών* (Module Recontextualization) με περιγραφή των διαδοχικών φάσεων (Αποσύνθεσης, Μετασχηματισμού κύριου και πλαισίου, σύνδεσης πλαισίου, μετασχηματισμού διεπαφής, επανασύνθεσης). Παρουσιάζεται αυτή η νέα προσέγγιση δυναμικής ανάλυσης που στοχεύει σε σύγχρονες δυναμικές γλώσσες και λειτουργεί στα όρια της κάθε επιλεγμένης βιβλιοθήκης.

Κεφάλαιο 3: PySecu
Στο κεφάλαιο αυτό αναφέρονται σχετικά με το `PySecu` τα παρακάτω:

- Επισκόπηση λειτουργίας του `PySecu` με τις ειδικότερες τεχνικές και συμπεριφορές του.
- Συνοπτική λειτουργία του `PySecu`.
- Διάγραμμα ροής της ανάλυσης.

Κεφάλαιο 4: Υλοποίηση
Σε αυτό το κεφάλαιο, περιγράφεται ο τρόπος υλοποίησης του εργαλείου αδρομερούς δυναμικής ανάλυσης PySecu. Πιο συγκεκριμένα, η υλοποίηση αγκίστρων και των περιτυλιγμάτων που χρησιμοποιούνται για την καταγραφή των αλληλεπιδράσεων των στοιχείων Python και πολλών άλλων βοηθητικών λειτουργιών. Κάθε φορά που εισάγεται ένα μέρος βιβλιοθήκης (module), το εργαλείο εκτελεί και αναλύει με αδρομερή δυναμική ανάλυση, αλλά σε σύντομο χρόνο. Κατά την εκτέλεση, η ανάλυση αποθηκεύει τα δικαιώματα σε ένα αρχείο ή τα επιβάλλει στο πρόγραμμα.

Κεφάλαιο 5: Παραδείγματα - Εφαρμογές
Σε αυτό το κεφάλαιο, παρουσιάζονται δύο παραδείγματα εφαρμογής του εργαλείου PySecu:

- στο left-pad και
- στο pythonbenchmark

Πρώτα, εκτελούμε κάθε παράδειγμα χωρίς το εργαλείο ανάλυσης και μετά το εκτελούμε με το εργαλείο ανάλυσης. Καθώς εκτελούμε το καθένα με το εργαλείο ανάλυσης, πρώτα δοκιμάζεται η επιλογή αποθήκευσης (-s) για να εξαχθούν τα δικαιώματα και, στη συνέχεια, δοκιμάζουμε την επιλογή επιβολής (-e) για να επιβληθούν αυτά τα δικαιώματα.

Κεφάλαιο 6: Αξιολόγηση
Σε αυτό το κεφάλαιο, το εργαλείο δυναμικής ανάλυσης PySecu αξιολογείται ως προς τη λειτουργικότητά του και το χρόνο ανάλυσης που επιβάλει. Πιο συγκεκριμένα αξιολογούνται τα παρακάτω:

- Αποτελεσματικότητα του PySecu ως εργαλείο ανάλυσης ασφαλείας.
- Χρόνος ανάλυσης συγκριτικά με το χρόνο της απλής εκτέλεσης και το χρόνο ανάλυσης με το ενσωματωμένο εργαλείο API της Python.
- Αξιοπιστία διαδικασίας ανάλυσης και μελέτη ταυτότητας των τροποποιημένων στοιχείων.

Κεφάλαιο 7: Σύγκριση με άλλα εργαλεία
Σε αυτό το κεφάλαιο εξετάζεται η σύγκριση με άλλα εργαλεία δυναμικής ανάλυσης και με αντίστοιχα εργαλεία αδρομερούς δυναμικής ανάλυσης άλλων δυναμικών γλωσσών προγραμματισμού.

Κεφάλαιο 8: Σχετικές εργασίες και Μελλοντικές πρωτοβουλίες
Σε αυτό το κεφάλαιο παρατίθενται προτάσεις για μελλοντικές ερευνητικές εργασίες για βελτίωση του PySecu και αξιοποίηση της αδρομερούς δυναμικής ανάλυσης.

Κεφάλαιο 9: Συμπεράσματα
Στο κεφάλαιο αυτό παρουσιάζονται τα γενικά συμπεράσματα σχετικά με την εφαρμογή του PySecu που η λειτουργία του ακολουθεί την αρχιτεκτονική της τεχνικής αδρομερούς δυναμικής ανάλυσης.

Παράρτημα Α: Σε αυτό το παράρτημα, υπάρχει ένα εγχειρίδιο με δείγματα αποσπασμάτων κώδικα για να μπορεί ο χρήστης να εκτελέσει ένα απλό πρόγραμμα στην Python και να κατανοήσει την διαδικασία εκτέλεσής του.

Παράρτημα Β: Σε αυτό το παράρτημα, παρουσιάζεται το εγχειρίδιο που περιγράφει τον τρόπο εγκατάστασης του εργαλείου δυναμικής ανάλυσης PySecu, με τη χρήση του διαχειριστή πακέτων της Python που ονομάζεται pip. Επίσης παρουσιάζεται το εγχειρίδιο της εγκατάστασης του PySecu από το GitHub.

Παράρτημα C: Σε αυτό το παράρτημα, παρουσιάζονται τα στοιχεία που χρησιμοποιούνται στο κεφάλαιο 6 . Κατ' αρχάς, υπάρχουν πληροφορίες για κάθε ενσωματωμένο στοιχείο της Python σε έναν πίνακα και το πώς αντιδρά με την ανάλυση. Στη συνέχεια, στο ίδιο παράρτημα, υπάρχει ένας άλλος πίνακας σχετικά με τα ποσοστά των περιτυλιγμένων στοιχείων για κάθε βιβλιοθήκη, τα συνολικά καθώς και τα αντίστοιχα στοιχεία για τις μεγαλύτερες και πιο σημαντικές βιβλιοθήκες. Στο τέλος, υπάρχουν πληροφορίες για κάθε βιβλιοθήκη που χρησιμοποιήθηκε για την ανάλυση μικρο-πακέτων.

Κεφάλαιο 2

Θεωρητικό Πλαίσιο

2.1 Αρθρωτός Προγραμματισμός

Ο αρθρωτός προγραμματισμός (modular programming) είναι μια τεχνική σχεδίασης λογισμικού που βασίζεται στον διαχωρισμό της λειτουργικότητας ενός προγράμματος σε ανεξάρτητες, εναλλάξιμες ενότητες, έτσι ώστε το καθένα να περιέχει όλα τα απαραίτητα για την εκτέλεση μόνο μιας πτυχής της λειτουργίας του που θέλουμε να επιτύχουμε [26]. Αυτές οι μονάδες μπορούν να δημιουργηθούν ανεξάρτητα, να δοκιμαστούν επίσης ανεξάρτητα και σε πολλές περιπτώσεις μπορούν να χρησιμοποιηθούν ακόμη και σε άλλα συστήματα που λειτουργούν με παρόμοιες αρχές λειτουργίας. Αυτή η λειτουργία των modules μπορεί να παρέχεται είτε από τη γλώσσα προγραμματισμού, πιθανώς περιτυλίγοντας διεπαφές λειτουργικού συστήματος, ώστε να συμμορφώνεται με τις συμβάσεις της γλώσσας είτε παρέχεται από άλλους προγραμματιστές, αφού η κοινή χρήση κώδικα μπορεί να αποβαίνει χρήσιμη για κάποιους.

Η προσέγγιση αυτή του αρθρωτού σχεδιασμού έχει αποδειχτεί πολύ χρήσιμη και απαραίτητη στη μηχανική ακόμη και πολύ πριν από τους πρώτους υπολογιστές. Δεν υπάρχει σχεδόν κανένα προϊόν στις μέρες μας που να μην βασίζεται σε μεγάλο βαθμό σε αυτή τη διαμόρφωση. Τα αυτοκίνητα, διάφορα εργαλεία, η πολεμική βιομηχανία και τα κινητά τηλέφωνα είναι ενδεικτικά παραδείγματα αυτού του τρόπου σχεδιασμού.

Οι υπολογιστές όμως ανήκουν σε εκείνα τα προϊόντα που είναι κατασκευασμένα σχεδόν εξαρχής, σε μέγιστο επίπεδο, με αρθρωτό σχεδιασμό. Αυτό εμφανίστηκε από τους πρώτους ακόμη υπολογιστές, όπως ο 360 mainframe υπολογιστής της IBM που ήταν πραγματικά αρθρωτός και σχεδιάστηκε για να έχει διάφορα μέρη, που ονομάζονται modules. Οι μονάδες σχεδιάστηκαν και κατασκευάστηκαν ανεξάρτητα η μία από την άλλη, αλλά, όταν συνδυάζονταν, συνεργάζονταν άψογα. Ως αποτέλεσμα, όλα τα συστήματα που είναι κατασκευασμένα από μονάδες System/360 θα μπορούσαν να εκτελούν το ίδιο λογισμικό. Επιπλέον, νέες μονάδες θα μπορούσαν να προστεθούν στο σύστημα και οι παλιές θα μπορούσαν να αναβαθμιστούν, χωρίς να ξαναγραφεί ο κώδικας ή να διακοπεί η λειτουργία [20].

Έτσι, αυτό που είναι απαραίτητο και οικονομικότερη λύση για το υλικό, είναι πια μια αναπόφευκτη αναγκαιότητα πλέον για τον σχεδιασμό του λογισμικού που εκτελείται στους υπολογιστές. Εάν ο προγραμματιστής θέλει να αναπτύξει προγράμματα που είναι ευανάγνωστα, αξιόπιστα και με δυνατότητα συντήρησης με άμεσο, οικονομικό και βέλτιστο τρόπο, πρέπει να χρησιμοποιήσει κάποια εφαρμογή σχεδιασμού του λογισμικού η οποία θα προβλέπει το σχεδιασμό σε modules, και θα τον βοηθήσει σε αυτή τη προσπάθειά του, ειδικά αν η εφαρμογή που πρόκειται να αναπτύξει έχει μεγάλο μέγεθος.

Υπάρχει μεγάλη ποικιλία από τεχνικές για το σχεδιασμό λογισμικού σε αρθρωτή μορφή. Ο

αρθρωτός προγραμματισμός όπως αναπτύχθηκε σαν τεχνική σχεδιασμού λογισμικού χωρίζει τον κώδικα σε διακριτά μέρη τα οποία ονομάζονται ενότητες ή μέρη βιβλιοθηκών (modules). Το επίκεντρο του σχεδιασμού σε αυτή τη περίπτωση είναι ότι με αυτό τον διαχωρισμό θα πρέπει να υπάρχουν modules χωρίς ή με το κατά το δυνατό μικρότερες εξαρτήσεις από άλλα modules. Με άλλα λόγια, η ελαχιστοποίηση των εξαρτήσεων είναι ένας από τους σημαντικούς στόχους του σχεδιασμού. Κατά τη δημιουργία ενός αρθρωτού συστήματος, πολλά modules κατασκευάζονται χωριστά και λίγο πολύ ανεξάρτητα. Η εκτελέσιμη εφαρμογή θα δημιουργηθεί τοποθετώντας τις μαζί και συνδέοντάς τις κατά τη ροή του προγράμματος [28].

Τα πλεονεκτήματα του αρθρωτού προγραμματισμού είναι τα ακόλουθα:

- **Ο κώδικας είναι πιο ευανάγνωστος**: Η εργασία με τον αρθρωτό προγραμματισμό διευκολύνει την ανάγνωση του κώδικα επειδή οι συναρτήσεις εκτελούν διαφορετικές εργασίες σε σύγκριση με τους συνεχούς ροής κώδικες. Μερικές φορές ο αρθρωτός προγραμματισμός μπορεί να είναι μπλεγμένος εάν μεταφέρουμε ορίσματα και μεταβλητές σε διαφορετικές συναρτήσεις για το λόγο αυτό απαιτείται μεγάλη προσοχή στο σχεδιασμό. Η χρήση των μονάδων πρέπει να γίνεται με τάξη και λογική ροή ώστε να αποφεύγονται οποιαδήποτε προβλήματα. Οι λειτουργίες πρέπει να είναι τακτοποιημένες, ξεκάθαρες και περιγραφικές.

- **Ο κώδικας ελέγχεται ευκολότερα**: Στο λογισμικό, ορισμένες λειτουργίες εκτελούν περιορισμένο αριθμό εργασιών και κάποιες άλλες εκτελούν πολλές εργασίες. Εάν το λογισμικό χωρίζεται εύκολα με χρήση μονάδων, γίνεται πιο εύκολο να δοκιμαστεί. Μπορούμε επίσης να εστιάσουμε στις πιο επικίνδυνες λειτουργίες κατά τη διάρκεια της δοκιμής οι οποίες θα απαιτήσουν περισσότερες δοκιμές για να πιστοποιηθεί ότι δεν έχουν ανεπιθύμητα σφάλματα.

- **Επαναχρησιμοποίηση**: Υπάρχουν φορές που ένα κομμάτι κώδικα εφαρμόζεται σε πάρα πολλά σημεία στο πρόγραμμά μας. Αντί να το αντιγράφουμε και να το επικολλάμε ξανά και ξανά, η αρθρωτή βαθμίδα μας δίνει το πλεονέκτημα της επαναχρησιμοποίησης, ώστε να μπορούμε να τραβάμε τον κώδικά μας από οπουδήποτε χρησιμοποιώντας διεπαφές ή βιβλιοθήκες. Η χρήση της επαναχρησιμοποίησης μειώνει το μέγεθος του προγράμματός μας.

- **Ταχύτερες επιδιορθώσεις**: Ας υποθέσουμε ότι υπάρχει ένα σφάλμα στις επιλογές πληρωμής σε οποιαδήποτε εφαρμογή και το σφάλμα πρέπει να αφαιρεθεί. Ο αρθρωτός προγραμματισμός μπορεί να βοηθήσει πολύ αν για παράδειγμα γνωρίζουμε ότι υπάρχει μια ξεχωριστή συνάρτηση που περιέχει τον κώδικα πληρωμών και έτσι θα απαιτηθεί να διορθωθεί μόνο αυτή η λειτουργία. Άρα, η χρήση μονάδων κάνει την εύρεση και τη διόρθωση σφαλμάτων, πολύ πιο ομαλή και εύκολα συντηρήσιμη.

- **Ενημερώσεις χαμηλού κινδύνου**: Στον αρθρωτό προγραμματισμό, ένα καθορισμένο επίπεδο API προστατεύει όσους το χρησιμοποιούν από την πραγματοποίηση αλλαγών μέσα στη βιβλιοθήκη. Αν δεν υπάρξει αλλαγή στο API, τότε ο κίνδυνος για παραβίαση κώδικα από κάποιον είναι μικρός. Για παράδειγμα, εάν δεν υπάρχουν ρητά API και κάποιος αλλάξει μια συνάρτηση που έχει χρησιμοποιηθεί και αλλού ενώ εκείνος δεν μπορεί να αναγνωρίσει όποιες αλλαγές αφού πιστεύει ότι χρησιμοποιήθηκε μόνο στην ίδια βιβλιοθήκη, θα μπορούσε κατά λάθος να προκαλέσει δυσλειτουργία του κώδικά του.

- **Ευκολία στη συνεργασία**: Διαφορετικοί προγραμματιστές εργάζονται σε ένα μόνο κομμάτι κώδικα στην ομάδα. Υπάρχουν πιθανότητες διενέξεων που μπορεί να προκύψουν από κυκλική επαναφορά των συναρτήσεων. Αυτή η διένεξη μπορεί να μειωθεί εάν ο κώδικας χωριστεί σε περισσότερες λειτουργίες, αρχεία, repos, κ.λπ. Μπορούμε επίσης να παρέχουμε

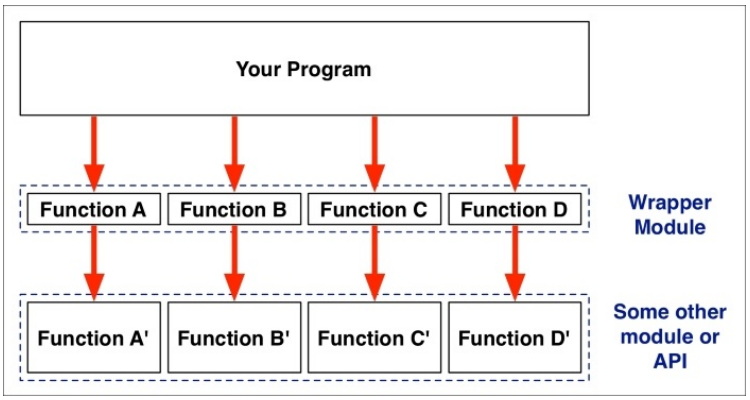
αρμοδιότητες διαχειριστή σε ένα μέλος της ομάδας για να μπορεί να αναλύσει συγκεκριμένα τμήματα κώδικα σε μικρότερες εργασίες.

Τα παρακάτω είναι μειονεκτήματα του αρθρωτού προγραμματισμού:

- Στον αρθρωτό προγραμματισμό, αν απαιτηθεί να κατασκευαστεί ένα επιπλέον module (που δεν υπάρχει ήδη έτοιμο), θα απαιτηθεί επιπλέον χρόνος και πόροι.
- Σε ορισμένες περιπτώσεις, είναι δύσκολη πρόκληση να συνδυαστούν όλα τα modules μεταξύ τους.
- Απαιτείται επιπλέον προσοχή στις όποιες αλλαγές, ώστε να μην επηρεάζονται άλλα μέρη του προγράμματος.
- Ορισμένα modules ενδέχεται να επαναλαμβάνουν εν μέρει την εργασία που εκτελείται από άλλα. Ως εκ τούτου, τα αρθρωτά προγράμματα χρειάζονται περισσότερο χώρο στη μνήμη και επιπλέον χρόνο για την εκτέλεσή τους.
- Η ενσωμάτωση διαφόρων λειτουργικών μονάδων σε ένα ενιαίο πρόγραμμα μπορεί να μην είναι πάντα η σωστή λύση, όταν διαφορετικά άτομα που εργάζονται για τη σχεδίαση διαφορετικών λειτουργικών τμημάτων μπορεί να μην έχουν το ίδιο στυλ ή προγραμματιστική νοοτροπία. Για το λόγο αυτό θα πρέπει να εκτιμάται και να γίνεται σωστή διαχείριση του διαμερισμού του προγραμματισμού των τμημάτων από την ομάδα εργασίας.
- Στις επιπλοκές συχνά περιλαμβάνεται και η χρήση προσωρινής μνήμης ώστε να αποφευχθεί το loading overhead και να εξασφαλιστεί η συνέπεια των modules, που φορτώνονται πολλές φορές από διαφορετικά σημεία του κώδικα. Η χρήση προσωρινής μνήμης υποστηρίζει επίσης αναδρομικές εισαγωγές και κυκλικές εξαρτήσεις. Ολοένα και πιο συχνά συναντάται το παράδοξο, να επιτρέπονται διαφορετικές εκδόσεις του ίδιου module σε ένα πρόγραμμα, γνωστό ως *dependency hell*. Αυτές οι χρήσεις μπορούν να περιπλέξουν σημαντικά τις δυναμικές αναλύσεις που λειτουργούν σε υψηλό βαθμό ανάλυσης των modules [37].

Ο αρθρωτός προγραμματισμός σαν έννοια έχει πλέον μια ώριμη διαδικασία με πολύχρονη εφαρμογή, όμως εξακολουθεί να είναι μια ευρέως διαδεδομένη διαδικασία μεταξύ των προγραμματιστών. Οι προγραμματιστές, στις μέρες μας, είναι υποχρεωτικό να μάθουν να κωδικοποιούν σε ενότητες, αφού ο αρθρωτός προγραμματισμός έχει τέτοια χαρακτηριστικά, που και στη γλώσσα Python τον καθιστούν απαραίτητο [42].

Σχηματικά σε Python ο αρθρωτός προγραμματισμός φαίνεται στο παρακάτω Γράφημα 2.1.



Source: oreilly.com

Γράφημα 2.1: Περιτυλίγματα (Wrappers)/Αρθρωτός προγραμματισμός σε Python.

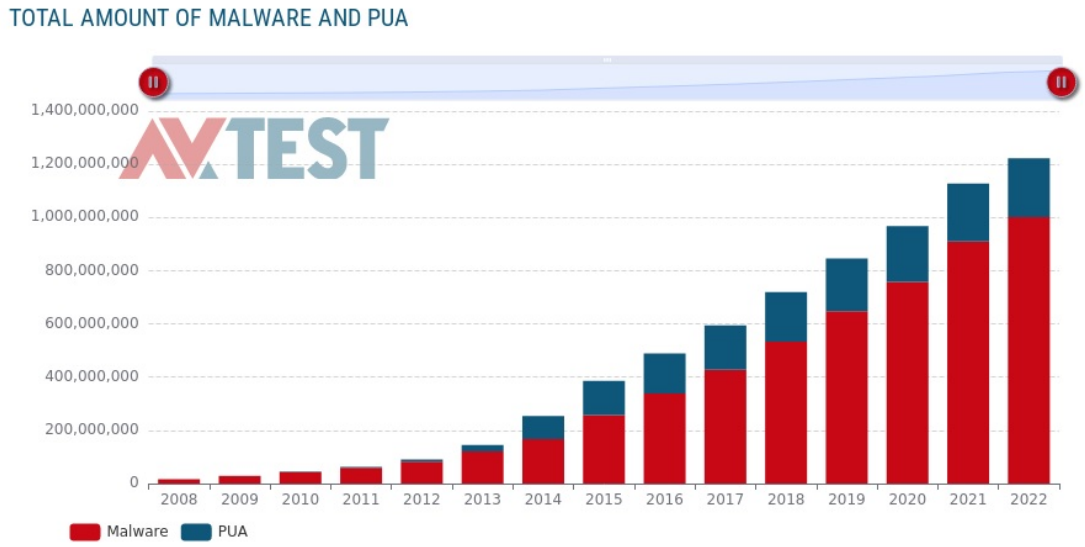
2.2 Βιβλιοθήκες (Libraries) - Ασφάλεια Λογισμικού

Βιβλιοθήκη (library) είναι μια συλλογή από έτοιμα υποπρογράμματα που χρησιμοποιείται για την ανάπτυξη ενός ευρύτερου τελικού προγράμματος. Υπάρχουν βιβλιοθήκες για κάθε δυναμική γλώσσα προγραμματισμού. Οι βιβλιοθήκες περιέχουν μεταξύ άλλων υποβοηθητικό κώδικα και δεδομένα σε τρόπο ώστε να επιτρέπεται ο διαμοιρασμός και η χρήση τους με δομημένο τρόπο. Ως εκ τούτου η έννοια της βιβλιοθήκης αποτελεί αναπόσπαστο τμήμα του αρθρωτού προγραμματισμού. Ειδικότερα οι δυναμικές βιβλιοθήκες ενσωματώνονται και επανατοποθετούνται στον τελικό κώδικα απευθείας στη μνήμη και ενώ το πρόγραμμα εκτελείται, με αποτέλεσμα τα εκτελέσιμα αρχεία να έχουν σαφώς μικρότερο μέγεθος. Πολλές φορές μάλιστα, το μεγαλύτερο μέρος του κώδικα που χρησιμοποιούν οι σύγχρονες εφαρμογές, παρέχεται από τις βιβλιοθήκες και δεν χρειάζεται να γραφεί από την αρχή για κάθε νέο πρόγραμμα.

Σε κάθε περίπτωση λοιπόν, εστιάζοντας στις καινοφανείς λειτουργίες του προγράμματος και όχι στα προτυποποιημένα τμήματα κώδικα των βιβλιοθηκών, εξοικονομούνται πολύτιμοι πόροι όπως χρόνος και πολλές ανθρωποώρες εργασίας για την ανάπτυξη ενός λογισμικού, εκτός του σημαντικού αποτυπώματος στο μέγεθος κώδικα (LoC) και στον χρόνο εκτέλεσης του προγράμματος. Συμπερασματικά, μια βιβλιοθήκη είναι μια συλλογή αντικειμένων και συναρτήσεων που μπορούν να χρησιμοποιηθούν μεμονωμένα αλλά ρυθμίζονται ώστε να λειτουργούν και σαν ενιαίο σύνολο για την επίλυση ενός συγκεκριμένου προβλήματος, επιτρέποντας στον χρήστη τον σχεδόν απόλυτο έλεγχο της ροής μιας εφαρμογής.

Η σύγχρονη ανάπτυξη λογισμικού βασίζεται σε μεγάλο βαθμό στις βιβλιοθήκες που έχουν δημιουργηθεί κατά βάση από τρίτους (*third-party libraries*). Οι εφαρμογές χρησιμοποιούν πολλές δεκάδες ή και εκατοντάδες βιβλιοθήκες, που έχουν δημιουργηθεί από πολλούς διαφορετικούς συγγραφείς και έχει δοθεί δικαίωμα πρόσβασης μέσω δημόσιων αποθετηρίων [41]. Η συχνή χρήση βιβλιοθηκών είναι κοινή ιδιαίτερα σε εφαρμογές γλωσσών προγραμματισμού [16, 24], όπου οι προγραμματιστές έχουν εκατομμύρια βιβλιοθήκες στη διάθεσή τους μέσω του διαχειριστή των πακέτων της γλώσσας. Η αύξηση του κακόβουλου λογισμικού τα τελευταία χρόνια [27] καθιστά τη δημοσίευση των βιβλιοθηκών στο διαδίκτυο ιδιαίτερα επισφαλή. Στοιχεία για το κακόβουλο λογισμικό και τις πιθανώς ανεπιθύμητες εφαρμογές για την χρονική περίοδο 2008-2022 [6] φαίνονται στο Γράφημα 2.2.

Κατά τη διάρκεια του χρόνου εκτέλεσης μιας βιβλιοθήκης μέσω των εισόδων της, είναι δυνατό να επηρεαστεί η ασφάλεια ολόκληρης της εφαρμογής και το ευρύτερο περιβάλλον λειτουργίας της. Για παράδειγμα, υπήρξε περιστατικό, όπου συμπεριλήφθηκε κώδικας για τη συλλογή στοιχείων λογαριασμού από επιλεγμένα πορτοφόλια Bitcoin κατά την εκτέλεση ως μέρος του πορτοφολιού Copay [5]. Την εποχή αυτού του περιστατικού, ο κώδικας χρησιμοποιήθηκε από εκατοντάδες εφαρμογές και κατά μέσο όρο ήταν περίπου δύο εκατομμύρια λήψεις την εβδομάδα με αποτέλεσμα την διαρροή ευαίσθητων στοιχείων σε κακόβουλους χρήστες. Για να διασφαλιστεί η αξιοπιστία των βιβλιοθηκών, όπως έχει προαναφερθεί, έχουν αναπτυχθεί τεχνικές ανάλυσης των εφαρμογών που διακινούνται μέσω αυτών. Για την ανάλυση τους προτιμώνται τεχνικές δυναμικής ανάλυσης για τους λόγους που αναφέρονται παρακάτω.



Source: av-atlas.org

Γράφημα 2.2: Κακόβουλο λογισμικό (Malware) και πιθανώς ανεπιθύμητες εφαρμογές (PUA) ετών 2008-2022.

2.3 Σύγκριση της Δυναμικής Ανάλυσης με τη Στατική Ανάλυση

Στην ενότητα που ακολουθεί, εμφανίζονται οι διαφορές της δυναμικής ανάλυσης με τη στατική ανάλυση και επίσης τα πλεονεκτήματα υπό προϋποθέσεις της δυναμικής ανάλυσης έναντι της στατικής ανάλυσης.

Στη διαδικασία της στατικής ανάλυσης, ένα πρόγραμμα ελέγχεται και αξιολογείται εξετάζοντας τον κώδικα του, χωρίς να εκτελείται. Η στατική ανάλυση έχει το πλεονέκτημα ότι μπορεί να αποκαλύψει, πως θα συμπεριφερθεί ένα πρόγραμμα κάτω από ασυνήθιστες συνθήκες. Αυτό συμβαίνει γιατί μπορούμε να εξετάσουμε κάποια μέρη ενός προγράμματος που δεν θα είχαν εκτελεστεί στην κανονική ροή εκτέλεσης του προγράμματος. Προκειμένου να επιτευχθεί στατική ανάλυση θα πρέπει κάποιος να έχει καλές γνώσεις της συμβολικής γλώσσας και του λειτουργικού συστήματος του στόχου. Αποτελεί όμως μια χρονοβόρα διαδικασία που απαιτεί ειδικές γνώσεις και που ως επί το πλείστον καθίσταται αναποτελεσματική, δεδομένης της πληθώρας κακόβουλων λογισμικών.

Στη διαδικασία της δυναμικής ανάλυσης ο έλεγχος και η αξιολόγηση πραγματοποιείται κατά τη διάρκεια της εκτέλεσης του κώδικα και δίνεται η δυνατότητα εξαγωγής εύκολων και γρήγορων συμπερασμάτων για τις λειτουργίες του. Η δυναμική ανάλυση αποκαλύπτει ελαττώματα ή τρωτά σημεία, κατά την εκτέλεση του κώδικα, των οποίων η αιτία απαιτεί αρκετά περίπλοκη διαδικασία για να αποκαλυφθεί με τη χρήση στατικής ανάλυσης. Αυτό γίνεται με χρήση debugger [2] με σκοπό την εξέταση της εσωτερικής κατάστασης ενός μέρους βιβλιοθήκης κατά την εκτέλεση του. Μέσω αυτής της τεχνικής, παρέχονται χρήσιμες λεπτομέρειες οι οποίες είναι αδύνατο να αποκτηθούν μέσω οποιασδήποτε άλλης τεχνικής. Debugger ονομάζεται ένα πρόγραμμα το οποίο χρησιμοποιείται για την εξέταση και αποσφαλμάτωση άλλων προγραμμάτων (προγράμματα *στόχοι*). Επιπλέον, με την απαραίτητη τεχνογνωσία και με τις κατάλληλες παραμετροποιήσεις η ανάλυση γίνεται σε βάθος και σχηματίζεται μία αρκετά ολοκληρωμένη εικόνα για τα χαρακτηριστικά του κακόβουλου λογισμικού.

Η διαδικασία της δυναμικής ανάλυσης κακόβουλου λογισμικού πρέπει να πραγματοποιείται σε ένα απομονωμένο και ελεγχόμενο περιβάλλον (φυσικό ή εικονικό), μέσα στο οποίο, με την χρήση κατάλληλων εργαλείων, μπορούμε με ασφάλεια να εκτελέσουμε το κακόβουλο λογισμικό και να παρακολουθήσουμε την συμπεριφορά του. Ο πρωταρχικός στόχος της δυναμικής ανάλυσης είναι η εύρεση και ο εντοπισμός σφαλμάτων [13]. Στη παρούσα διπλωματική εργασία όμως χρησιμοποιείται για την ανίχνευση στοιχείων καλόβουλου λογισμικού.

Πολλά εργαλεία δυναμικής αναλύσης έχουν γραφεί σε JavaScript [33, 17, 40, 36, 15, 9, 14] και λίγα σε Python [30, 10]. Όπως και με τη διαδικασία εντοπισμού σφαλμάτων έτσι και στην ασφάλεια η χρήση ενός εργαλείου δυναμικής ανάλυσης είναι προτιμότερο να πραγματοποιείται στην ίδια γλώσσα με την βιβλιοθήκη που αναλύεται, λόγω της οικειότητας του χρήστη με τη συγκεκριμένη γλώσσα προγραμματισμού.

2.4 Λεπτομερής – Αδρομερής Δυναμική Ανάλυση

Στο πλαίσιο της δυναμικής ανάλυσης, ένα πρόγραμμα δοκιμάζεται και αξιολογείται κατά τη διάρκεια της εκτέλεσης του.

Η λεπτομερής δυναμική ανάλυση ενός προγράμματος είναι μια χρονοβόρα τεχνική παρακολούθησης, κατανόησης και πιθανής παρέμβασης στη συμπεριφορά του προγράμματος, κατά την εκτέλεσή του. Πολλά εργαλεία δυναμικής ανάλυσης δημιουργούνται σήμερα για την αντιμετώπιση κάποιων απειλών. Γενικά υπάρχει η προτίμηση, σε δυναμικές γλώσσες προγραμματισμού να χρησιμοποιούνται δυναμικές αναλύσεις. Ειδικά για τη γλώσσα προγραμματισμού JavaScript, η οποία είναι μια πολύ δημοφιλής δυναμική γλώσσα προγραμματισμού, αναπτύχθηκαν πολλά εργαλεία (όπως Jalangi και NodeProf) [9, 14, 35, 21, 33, 25]. Τα εργαλεία αυτά επιτρέπουν συνήθως πολύ περίπλοκες και λεπτομερείς αναλύσεις, διατηρώντας επίσης πληροφορίες σχετικά με τα στοιχεία της γλώσσας προγραμματισμού, όπως if, while, break. Όμως υπάρχει δυσκολία με αυτά τα εργαλεία αφού έχουν χαμηλές ταχύτητες και είναι πολύπλοκα ώστε συχνά η κατανόηση και κατά συνέπεια η χρήση τους να παρουσιάζει δυσχέρειες για τους χρήστες. Ένα παράδειγμα εργαλείου δυναμικής ανάλυσης είναι το NodeProf [35], που βασίζεται στη χρήση αφηρημένων συντακτικών δέντρων (AST) για την εισαγωγή του κώδικα της ανάλυσης. Χρησιμοποιεί τα εργαλεία Graal [43] και Truffle [44], παρέχει αρκετά λεπτομερείς, αλλά αργού ρυθμού αναλύσεις, αφού η χρήση του Truffle – διαφορετικό από το Node.js – απαιτεί τροποποιήσεις για να λειτουργήσει σε περιβάλλον συγκεκριμένης γλώσσας. Ένα άλλο παράδειγμα εργαλείου δυναμικής ανάλυσης που αναπτύχθηκε στη WebAssembly, υποσύνολο της JavaScript, είναι το πρωτοπόρο Wasabi που υλοποιείται εύκολα, ελέγχει ομαλά τον κώδικα παρά τη μορφοποίηση του, αλλά είναι πολύ αργό.

Σε αντίθεση με τα παραπάνω, σύγχρονα εργαλεία που αναπτύσσονται σήμερα, βασίζονται στην αδρομερή δυναμική ανάλυση, στοχεύουν σε γρήγορες και ικανοποιητικές αναλύσεις και είναι απόλυτα αξιοποιήσιμα κατά την εκτέλεση -*τρέξιμο*- του προγράμματος όπως το Lya [40], το οποίο έχει αναπτυχθεί επίσης σε JavaScript. Στην αδρομερή δυναμική ανάλυση, αναλύονται μεγαλύτερα, αλλά πιο διακριτά τμήματα από τη λεπτομερή δυναμική ανάλυση, η οποία αναφέρεται σε μικρότερα τμήματα κώδικα από τα οποία αποτελούνται τα μεγαλύτερα. Με άλλα λόγια, μια αδρομερής περιγραφή ενός συστήματος αναφέρεται σε μεγάλα τμήματα του προγράμματος. Κατά συνέπεια επικεντρώνεται στη μείωση της λεπτομέρειας της ανάλυσης και ως αποτέλεσμα, το κόστος γίνεται σημαντικά χαμηλότερο σε σύγκριση με τις λεπτομερείς τεχνικές δυναμικής ανάλυσης. Στόχος αυτών των αναλύσεων είναι να τρέχουν παράλληλα με το πρόγραμμα κατά τη διάρκεια της παραγωγής χωρίς να χρειάζεται να απενεργοποιηθούν, δίνοντας συνεχή ενημέρωση. Η αδρομερής δυναμική ανάλυση λειτουργεί στα περιγράμματα των βιβλιοθηκών, αναλύοντας τις αλληλεπιδράσεις τους [39, 38]. Η λεπτομερής δυναμική ανάλυση επεξεργάζεται όλες τις μεταβλητές, τις συναρτήσεις, τις δηλώσεις και ακόμη και τον ίδιο τον κώδικα, ενώ η αδρομερής δυναμική ανάλυση επεξεργάζεται κομμάτια της βιβλιοθήκης

χωρίς λεπτομέρεια. Για παράδειγμα, το Lya είναι λιγότερο λεπτομερές από το NodeProf και το Wasabi αλλά προσφέρει σαφώς γρηγορότερη ανάλυση.

2.5 Επαναπλαισιοποίηση Βιβλιοθηκών

2.5.1 Δομή και Λειτουργία

Η τεχνική δυναμικής ανάλυσης *Επαναπλαισιοποίηση Βιβλιοθηκών* (Module Recontextualization), αναπτύχθηκε παράλληλα με την ανάπτυξη του εργαλείου Lya για Javascript, *εξαργυρώνει* λεπτομέρεια και ακρίβεια με την μείωση του κόστους εκτέλεσης της ανάλυσης ενός προγράμματος, υποστηρίζοντας ωστόσο τις σημαντικές διαδικασίες της ανάλυσης. Υλοποιείται με διαδοχική αποσύνθεση, μετασχηματισμό και ανασύνθεση των βιβλιοθηκών, διατηρώντας την αρχική δομή και λειτουργικότητά τους. Έχει τη δυνατότητα να αξιοποιεί τα χαρακτηριστικά των σύγχρονων δυναμικών γλωσσών όπως τη JavaScript, τη Lua και τη Python, ώστε να τροποποιεί δυναμικά κάθε βιβλιοθήκη εισάγοντας κώδικα του χρήστη στον πηγαίο κώδικά της, πριν φορτωθεί. Η ανάλυση των αλληλεπιδράσεων γίνεται σε επίπεδο πλαισίου της βιβλιοθήκης, που πρόκειται να συνδεθεί, χωρίς να απαιτείται τροποποίηση του περιβάλλοντος εκτέλεσης του προγράμματος ή των χαρακτηριστικών παραγωγής της δυναμικής γλώσσας. Αναλυτικότερα ισχύουν τα παρακάτω:

- **Αποσύνθεση**: Το εργαλείο φορτώνεται μαζί με ένα πρόγραμμα που θα αναλυθεί και ξεκινά με την αναδρομική αποσύνθεση ενός προγράμματος εντός των εξαρτήσεων του. Αυτό επιτυγχάνεται με την αλλαγή του τελεστή εισαγωγής import ώστε να περάσει από το εργαλείο και να καλείται με κάθε επίκληση του import. Θα τρέξει αναδρομικά τη δομή εξαρτήσεων του προγράμματος εντός του χρόνου εκτέλεσης. Κατά τη διάρκεια αυτής της φάσης, πρέπει να προσδιορίσει τα υψηλού βαθμού ανάλυσης μέρη βιβλιοθηκών (π.χ ανώτατου επιπέδου, ιδιαίτερης σημασίας κ.λπ.) ώστε να εφαρμοστούν οι μετασχηματισμοί στο σωστό επίπεδο αλλά και να σημειώσει τα άγκιστρα (hooks) της ανάλυσης στα αντίστοιχα modules.

- **Επαναπλαισιοποίηση**: Στη συνέχεια, το εργαλείο συνεχίζει ξεκινώντας το ουσιαστικό κομμάτι της ανάλυσης μετασχηματίζοντας κάθε διεπαφή του προγράμματος, το περιβάλλον του και πιθανώς τις τιμές που διέρχονται από το όριο κάθε βιβλιοθήκης ή μέρους της. Κώδικες μετασχηματισμού εισάγονται και περτυλίγουν (wrap) κάθε μία από αυτές τις τιμές με βάση τον τύπο τους. Αυτή η φάση απαιτεί την επίλυση πολλών προκλήσεων, συμπεριλαμβανομένης της απαρίθμησης όλων των σημείων εισόδου και εξόδου από ένα module και μετατροπή όλων των αποτελεσμάτων του προγράμματος περιτυλίγοντας τα με μηχανισμούς παρεμβολής.

- **Επανασύνθεση**: Τέλος, το εργαλείο επανασυνθέτει τα μεμονωμένα τροποποιημένα modules στην αρχική δομή του προγράμματος. Βασική πρόκληση σε αυτή τη φάση είναι η επεξεργασία της προσωρινής μνήμης (cache) του προγράμματος, η δυνατότητα της οποίας πρέπει να επαυξηθεί (σε ένα πρόγραμμα που έχει μόνο ένα επίπεδο προσωρινής μνήμης, καθίσταται αναγκαίο να επεκταθεί κατά δύο επίπεδα δηλαδή συνολικά τρία επίπεδα), για να μπορεί να υποστηρίζει πολλαπλά περιτυλίγματα ανά πρόγραμμα, ώστε το καθένα να αναλάβει ένα μέρος της συνολικής ανάλυσης.

2.5.2 Πλεονεκτήματα εφαρμογής της Επαναπλαισιοποίησης Βιβλιοθηκών

Τα πλεονεκτήματά εφαρμογής αναλύσεων που ακολουθούν την αρχιτεκτονική της Επαναπλαισιοποίησης Βιβλιοθηκών, δηλαδή του συγκεκριμένου τύπου αδρομερούς δυναμικής ανάλυσης, είναι:

1. Η νέα τεχνική ανάλυσης, έναντι συστημάτων δυναμικής ανάλυσης τελευταίας γενιάς, ανταλλάσσει λεπτομέρεια και ακρίβεια της ανάλυσης με τη μείωση του χρόνου καθυστέρησης της ανάλυσης, ενώ εξακολουθεί να υποστηρίζει τις κρίσιμες και ουσιαστικές διαδικασίες του προγράμματος.

2. Αξιοποιεί τις δυνατότητες των σύγχρονων δυναμικών γλωσσών προγραμματισμού μετασχηματίζοντας δυναμικά κάθε module, όταν αυτό φορτώνεται, εφαρμόζοντας μετασχηματισμούς πηγαίου κώδικα και αντικειμένων χωρίς επομένως να συντελούνται μεταβολές στο χρόνο εκτέλεσης. Λειτουργεί με εντελώς μη τροποποιημένους χρόνους παραγωγής της γλώσσας.

3. Τα εργαλεία ανάλυσης είναι κώδικες γραμμένοι στην ίδια γλώσσα με το εκάστοτε αναλυόμενο πρόγραμμα βιβλιοθηκών, modules και κωδίκων, διατηρώντας το προνόμιο της γνώσης των προγραμματιστών, την υπάρχουσα τεχνογνωσία, τις βιβλιοθήκες και την υπάρχουσα εμπειρία στη δυνατότητα ανάπτυξης προγραμμάτων ανάλυσης.

4. Η ανάλυση παραμένει πλήρως υπό τον έλεγχο του developer, αφού η υβριδική αυτή μορφή δυναμικής ανάλυσης υποστηρίζει τη στοχευμένη ανάλυση αποκλειστικά επιλεγμένων modules, μετασχηματίζοντάς τα στα όριά τους και παρέχει τη δυνατότητα δυναμικής εναλλαγής της ενεργοποίησης και απενεργοποίησής της (on/off) καθώς εκτελείται η εφαρμογή.

Κεφάλαιο 3

PySecu

3.1 Επισκόπηση

Το εργαλείο αδρομερούς δυναμικής ανάλυσης που αναπτύχθηκε, μελετήθηκε και παρουσιάζεται σε αυτή την εργασία, το `PySecu`, είναι αντίστοιχο με το εργαλείο αδρομερούς δυναμικής ανάλυσης που χρησιμοποιεί την τεχνική της Επαναπλαισιοποίησης Βιβλιοθηκών για το Lya [40] για τη σύγχρονη δυναμική γλώσσα προγραμματισμού JavaScript. Ωστόσο, η προαναφερθείσα προσέγγιση σε JavaScript [40, 41] έδειξε ότι συγκριτικά αυτού του είδους η δυναμική ανάλυση δίνει πιο ικανοποιητικά αποτελέσματα από άλλες προσπάθειες δυναμικής ανάλυσης στο επίπεδο της ευρείας κατανάλωσης προϊόντων προγραμματισμού.

Η συγκεκριμένη ανάλυση που έχει δομηθεί κατ' αναλογία της δομής και στη λογική της Επαναπλαισιοποίησης Βιβλιοθηκών, αφορά προγράμματα σε Python και οι αλληλεπιδράσεις που καταγράφονται είναι, η κλήση συνάρτησης, η δημιουργία αντικειμένου, η κλήση μεθόδου ενός αντικειμένου, η κλήση μεθόδου μιας κλάσης και η εισαγωγή ενός μέρους βιβλιοθήκης.

Το εργαλείο `PySecu` περιτυλίγει ενσωματωμένα και ορισμένα από το χρήστη στοιχεία με πρόσθετο κώδικα που σχετίζεται με την ανάλυση. Με τον όρο στοιχεία εννοούνται τα χαρακτηριστικά όπως οι συναρτήσεις, οι κλάσεις ή οι μέθοδοι κλάσεων που μπορούν να κληθούν.

Προκειμένου να περιτυλιχθούν τα ενσωματωμένα στοιχεία, το ίδιο το εργαλείο προσθέτει κώδικα ως συμβολοσειρά στην αρχή του κώδικα κάθε βιβλιοθήκης. Σε αυτόν τον πρόσθετο κώδικα, αντικαθιστά τα ενσωματωμένα στοιχεία της βιβλιοθήκης μέσω του λεξικού της με τα αντίστοιχα περιτυλιγμένα τους.

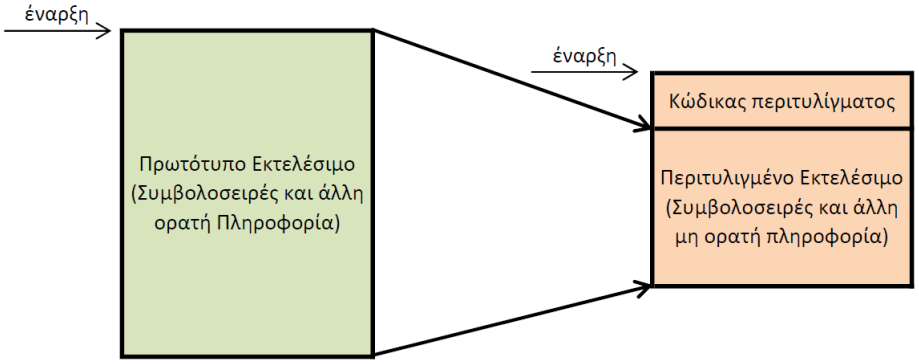
Για να περιτυλίξει τα στοιχεία που ορίζονται από το χρήστη, λαμβάνει τη βιβλιοθήκη που εισήχθη μετά την εκτέλεσή της και αλλάζει το λεξικό της σε ένα νέο με περιτυλιγμένα ορισμένα από το χρήστη στοιχεία. Για να μπορεί η ανάλυση να τρέξει και να εφαρμόσει τις λειτουργίες που έχουν ήδη περιγραφεί, τρέχει μέσα σε ένα άγκιστρο για τη συνάρτηση `__import__()` της Python. Ως αποτέλεσμα, εκτελείται κάθε φορά που ο χρήστης της βιβλιοθήκης εισάγει ένα μέρος βιβλιοθήκης μέσω της δήλωσης `import`.

Το πακέτο `PySecu` πρέπει να εισαχθεί στην αρχή του αναλυθέντος κώδικα για να λειτουργήσει σωστά. Για να τρέξει την ανάλυση, ο χρήστης της έχει δύο επιλογές για να την εκτελέσει, κάθε μία από τις οποίες μπορεί να προστεθεί στην εντολή τερματικού:

- αποθηκεύει τις προσβάσεις, δηλαδή τις χρήσεις στοιχείων της Python που μπορούν να κληθούν, στο αρχείο προσβάσεων της μορφής .json.
- επιβάλλει στο πρόγραμμα τα στοιχεία που μπορούν να κληθούν. Πρώτα φορτώνεται το αρχείο προσβάσεων. Στη συνέχεια, όταν καλείται το περιτυλιγμένο στοιχείο, ενεργοποιείται το

περιτύλιγμά του, για να αναζητήσει το περιεχόμενό του στις προσβάσεις. Αν βρεθεί στις προσβάσεις με την ένδειξη *allow*, το πρόγραμμα συνεχίζεται με τα υπόλοιπα περιτυλιγμένα στοιχεία μέχρι να ολοκληρωθεί η βιβλιοθήκη. Στις περιπτώσεις, όπου το στοιχείο δεν βρίσκεται στις προσβάσεις ή έχει την ένδειξη *deny*, το πρόγραμμα σταματά με ένα `AttributeError` και αντίστοιχο μήνυμα. Η παραπάνω λειτουργία αποδεικνύεται εξαιρετικά χρήσιμη στη περίπτωση που μια μη επιτρεπόμενη ενέργεια, συνήθως η έγχυση ενός κακόβουλου κώδικα, έχει προστεθεί στο αναλυόμενο πακέτο, κατά το χρόνο που το πακέτο είναι δημόσιο. Το PySecu, μπορεί να σταματήσει αμέσως το πρόγραμμα, πριν από την εκτέλεση του πρόσθετου κώδικα, ελέγχοντας το αρχείο με τις προσβάσεις που έχει ήδη αποθηκεύσει.

Οι μετασχηματισμοί υλοποιούνται εισάγοντας σταδιακά περιτυλίγματα που λειτουργούν σχηματικά με τον τρόπο, που φαίνεται στο παρακάτω Γράφημα 3.1. Το αρχείο αριστερά αναπαριστά το αυθεντικό εκτελέσιμο αρχείο μαζί με όλες τις συμβολοσειρές, τις εισαγωγές συναρτήσεων και άλλες πληροφορίες. Στα δεξιά φαίνεται το πακεταρισμένο (περιτυλιγμένο) εκτελέσιμο. Όλες οι πληροφορίες του αυθεντικού εκτελέσιμου είναι πλέον συμπιεσμένες και δεν μπορούν να διαβαστούν από άλλο εργαλείο ανάλυσης. Είναι προφανές ότι η μέθοδος αυτή επιστρατεύεται για την περαιτέρω υποστήριξη της αδρομερούς ανάλυσης. Το PySecu υλοποιείται γενικά ως άγκιστρο για τη συνάρτηση εισαγωγής ώστε να μπορεί να εκτελεί την ανάλυση παράλληλα με την εκτέλεση ενός module κάθε φορά που εισάγεται. Κατά τη διάρκεια της εκτέλεσης, τα στοιχεία αυτού του module με δυνατότητα κλήσης αποθηκεύονται ως προσβάσεις ή ελέγχονται εάν είναι αξιόπιστα διερευνώντας το αρχείο προσβάσεων κατά το χρόνο εκτέλεσης.



Γράφημα 3.1: Διαδικασία περιτυλίγματος (wrapping).

3.2 Συνοπτική Λειτουργία

Συνοπτικά και ανακεφαλαιώνοντας έχουμε:

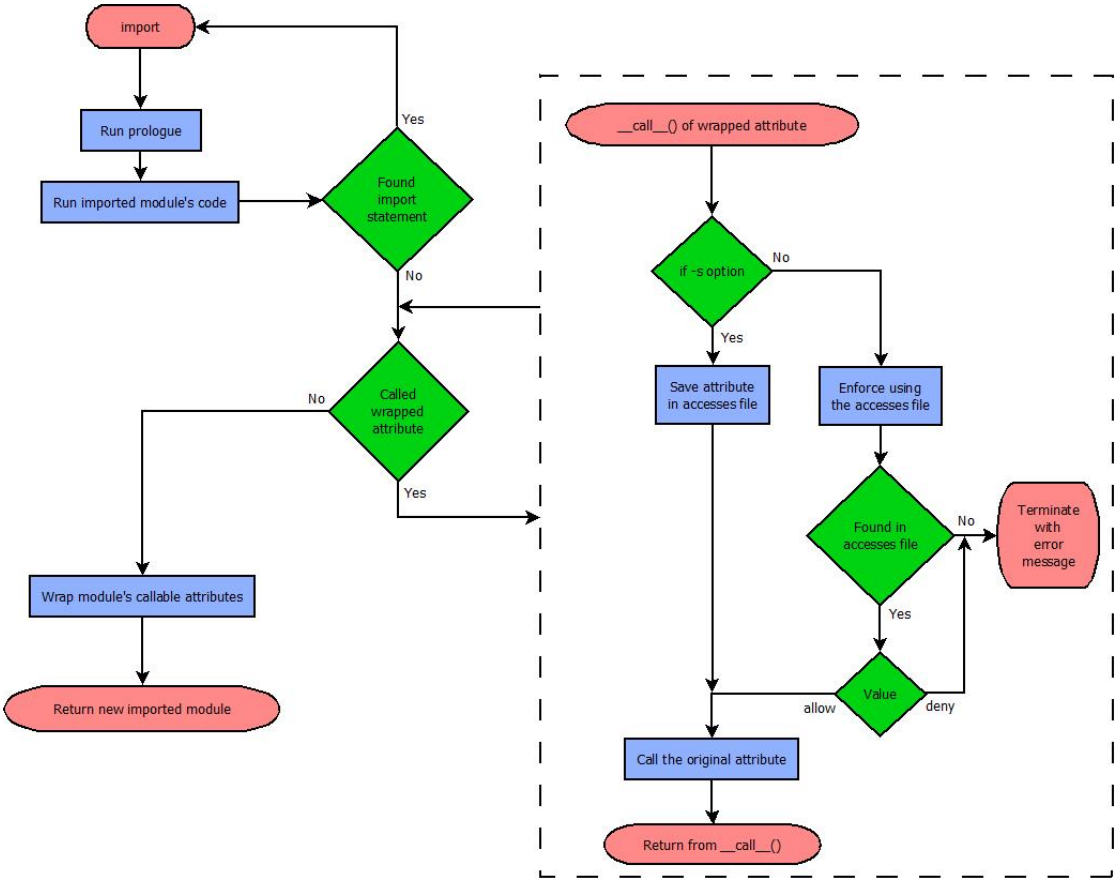
- Το πακέτο PySecu πρέπει να εισαχθεί στην αρχή του αναλυόμενου κώδικα, για να λειτουργήσει σωστά.
- Για να τρέξει την ανάλυση ο χρήστης της έχει δύο επιλογές για να την εκτελέσει, κάθε μία από τις οποίες μπορεί να προστεθεί στην εντολή τερματικού.
    * **-s**: Με αυτήν την επιλογή, το εργαλείο αποθηκεύει τις προσβάσεις, δηλαδή τις χρήσεις στοιχείων της Python που μπορούν να κληθούν, στο αρχείο προσβάσεων.

    * **-e**: Με αυτήν την επιλογή, η ανάλυση επιβάλλει στο πρόγραμμα τα στοιχεία που μπορούν να κληθούν. Πρώτα φορτώνεται το αρχείο πρόσβασης. Στη συνέχεια, όταν καλείται το περιτυλιγμένο στοιχείο, ενεργοποιείται το περιτύλιγμά του, για να αναζητήσει το περιεχόμενό του στις προσβάσεις. Αν βρεθεί στις προσβάσεις με την ένδειξη *allow*, το πρόγραμμα συνεχίζεται με τα υπόλοιπα περιτυλιγμένα στοιχεία μέχρι να ολοκληρωθεί η βιβλιοθήκη. Στις περιπτώσεις, όπου το στοιχείο δεν βρίσκεται στις προσβάσεις ή έχει την ένδειξη *deny*, το πρόγραμμα σταματά με ένα `AttributeError` και αντίστοιχο μήνυμα.

- Για να μπορέσει να *τρέξει* αυτό το εργαλείο και να αναλύσει βιβλιοθήκες, δημιουργείται ένα άγκιστρο (hook) για τη συνάρτηση `__import__()` της Python.
- Το αποτέλεσμα είναι να *τρέχει* η ανάλυση κάθε φορά που ο χρήστης εισάγει ένα μέρος βιβλιοθήκης με τη δήλωση `import`.
- Μέσα στο άγκιστρο της `__import__()`, καλείται η συνάρτηση `_analyze_import()`, όπου πέρα από την αναζήτηση και το φόρτωμα μιας βιβλιοθήκης, εισάγεται ο κώδικας του προλόγου στην αρχή του κώδικά της. Ο πρόλογος περιτυλίγει τα ενσωματωμένα στοιχεία της Python. Ο περιτυλιγμένος κώδικας της βιβλιοθήκης εκτελείται και ξεκινά η ανάλυση.
- Το εργαλείο `PySecu` τροποποιεί και περιτυλίγει ενσωματωμένα (built-in) στοιχεία και τα περιβάλλοντά τους με πρόσθετο κώδικα που σχετίζεται με την ανάλυση.
- Για να περιτυλιχθούν τα ενσωματωμένα στοιχεία, προστίθεται κώδικας ως συμβολοσειρά στην αρχή του κώδικα κάθε module. Σε αυτόν τον πρόσθετο κώδικα, αντικαθιστά τα ενσωματωμένα στοιχεία του module μέσω του λεξικού του με τα αντίστοιχα περιτυλιγμένα τους.
- Με τον όρο στοιχεία εννοούμε συναρτήσεις, κλάσεις ή μεθόδους που μπορούν να κληθούν.
- Κατά την ανάλυση μιας βιβλιοθήκης ή μέρους της, κάθε φορά που καλείται ένα περιτυλιγμένο στοιχείο, αρχικοποιείται το όνομα της πρόσβασης (`key`) ανάλογα με το αν είναι συνάρτηση, κλάση, μέθοδος κλάσης ή συνάρτηση κλάσης.
- Για να περιτυλίξει τα στοιχεία που ορίζονται από το χρήστη, παίρνει το μέρος, που εισήχθη, μετά την ανάλυσή του και αλλάζει το λεξικό του με περιτυλιγμένα τα στοιχεία του χρήστη.
- Το τροποποιημένο μέρος βιβλιοθήκης επιστρέφεται.

3.3 Διάγραμμα ροής της ανάλυσης

Στο παρακάτω Γράφημα 3.2 φαίνεται σχηματικά το διάγραμμα ροής της παραπάνω ανάλυσης.



Γράφημα 3.2: Διάγραμμα ροής της ανάλυσης PySecu.

Ένα απλό παράδειγμα εκτέλεσης της ανάλυσης στη βιβλιοθήκη `left-pad` φαίνεται στην παραγράφους 5.1.2 και 5.1.3.

Κεφάλαιο 4

Υλοποίηση

Σε αυτό το κεφάλαιο, θα περιγραφεί η υλοποίηση του εργαλείου ανάλυσης, το οποίο είναι γραμμένο μέσα στο αρχείο analysis.py.

4.1 Αγκιστροποίηση import

Αρχικά δημιουργείται ένα άγκιστρο για το import της Python. Αγκίστρωμα (hooking) σημαίνει ότι αντικαθιστούμε μία ενσωματωμένη συνάρτηση οποιασδήποτε γλώσσας προγραμματισμού με τη δική μας συνάρτηση και όταν καλείται η πρώτη, εκτελείται ο κώδικας της δικής μας συνάρτησης [7]. Τα παρακάτω αποσπάσματα κώδικα προέρχονται από το αρχείο analysis.py.

```
1  import builtins
2
3
4  def _import(name, globals=None, locals=None, fromlist=(), level=0):
5      ...
6      return original_import(name, globals, locals, fromlist, level)
7
8
9  original_import = builtins.__import__
10 builtins.__import__ = _import
```

Αυτή τη φορά, δημιουργείται πρώτα μια συνάρτηση με το όνομα `_import()`, η οποία προσθέτει επιπλέον κώδικα στον κώδικα της αρχικής συνάρτησης `__import__()`. Μετά, δημιουργείται μια μεταβλητή με το όνομα `original_import`. Σε αυτήν τη μεταβλητή, αποθηκεύεται η αρχική ενσωματωμένη συνάρτηση `__import__()` (στην πραγματικότητα αναφέρεται το όνομα `original_import` στο αντικείμενο `builtins.__import__` στην Python). Στη συνέχεια, αντικαθίσταται η αρχική ενσωματωμένη συνάρτηση με την τροποποιημένη, η οποία ονομάζεται `_import()`. Σαν αποτέλεσμα, εκτελείται η τροποποιημένη συνάρτηση κάθε φορά που εισάγεται ένα module.

4.2 Δημιουργία της ανάλυσης

Όπως αναφέρθηκε και παραπάνω, δημιουργείται ένα άγκιστρο για την ενσωματωμένη συνάρτηση `__import__()`.

_import

```
1  def _import(name, globals=None, locals=None, fromlist=(), level=0):
2      ...
```

Στη συνέχεια, η υλοποίηση της συνάρτησης θα περιγραφτεί με μεγαλύτερη λεπτομέρεια. Πρώτα απ' όλα, θα χρησιμοποιηθούν τα ορίσματα της αρχικής συνάρτησης `__import__()` ως ορίσματα της `_import()`. Πριν ξεκινήσει οτιδήποτε στο κύριο άγκιστρο της ανάλυσης, επιλύεται (παθαίνει resolve) το όνομα του module, το οποίο πρόκειται να εισαχθεί και ενδεχομένως να αναλυθεί. Αυτό γίνεται στην περίπτωση που ο χρήστης της βιβλιοθήκης χρησιμοποιεί τις τελείες (.) στη δήλωση `from import` για να κάνει μια σχετική (relative) εισαγωγή. Επίσης, είναι μια λειτουργία του συστήματος εισαγωγής της Python, που κάνει κάθε φορά όταν εισάγεται ένα module. Έτσι, απαιτείται, για να μπορεί να χρησιμοποιηθεί το απόλυτο (το κανονικό) όνομα του module στην τροποποιημένη εισαγωγή (import) και όχι μεμμονωμένα διαστήματα ή ονόματα από όπου λείπουν σημεία.

```
1      ...
2      if name in CPYTHON_MODULES:
3          return original_import(name, globals, locals, fromlist, level)
4      ...
```

Αργότερα, όπως φαίνεται στη γραμμή 2, εάν το όνομα του module είναι ένα από τα ονόματα τμημάτων `CPython`, τα οποία διατηρήθηκαν, δηλαδή εάν το module είναι μια επέκταση `C` της `Python`, τότε απλώς καλείται η `original_import()` και το module εισάγεται κανονικά. Αυτά τα modules είναι γραμμένα στη γλώσσα προγραμματισμού `C`. Η `CPython` είναι η προεπιλεγμένη και η πιο ευρέως χρησιμοποιούμενη υλοποίηση της γλώσσας `Python`. Με αυτόν τον τρόπο, το `C` module θα εισαχθεί και θα εκτελεστεί κανονικά χωρίς να αναλυθεί παράλληλα.

```
1      ...
2      rname = name
3      while '.' in rname:
4          rname, _, _ = rname.rpartition('.')
5
6      if rname in modules:
7          return original_import(name, globals, locals, fromlist, level)
8      ...
```

Αφού χειρίστηκαν τα modules `CPython`, αυτή τη στιγμή θα ληφθεί το αριστερό όνομα του ονόματος του module, το οποίο χωρίζεται με τελείες (.), όπως φαίνεται στις γραμμές 2-4 του παραπάνω αποσπάσματος κώδικα. Για να γίνει αυτό, αρχικοποιείται πρώτα το `rname`, όπου θα αποθηκευτεί το αριστερό όνομα (string), με το απόλυτο όνομα του module. Στη συνέχεια, σε ένα βρόχο `while`, καλείται η μέθοδος `rpartition()` της κλάσης συμβολοσειρά (string) για το `rname` με όρισμα την τελεία (.), έτσι ώστε να επιστραφεί η αριστερή πλευρά της τελείας (.). Μετά, το αποτέλεσμα, που επιστράφηκε από την `rpartition()`, αποθηκεύεται στο ίδιο το `rname`. Όλη αυτή η διαδικασία συνεχίζεται όσο υπάρχει η τελεία στο `rname`. Αν η τελεία δεν ανήκει πια στο `rname`, σταματά η δεξιά κατάτμηση (right partition) και το τελικό `rname` είναι το επιθυμητό `rname` για τη συνέχεια.

Τώρα τα εγγενή πακέτα ή γενικότερα modules της Python μπορούν να χειριστούν. Αυτά είναι τα ενσωματωμένα και της τυπικής βιβλιοθήκης modules που συνοδεύουν την Python. Στη γραμμή 6, τσεκάρεται αν το `rname`, δηλαδή το πακέτο ανώτατου επιπέδου (top-level), βρίσκεται μέσα στην καθολική λίστα συμβολοσειρών `modules`. Αν είναι μέσα σε αυτή τη λίστα, καλείται η `original_import()` και εισάγεται και εκτελείται το εγγενές module, αφού αποφασίστηκε ότι είναι εγγενές, πάλι κανονικά χωρίς να αναλυθεί. Διαφορετικά, το πρόγραμμα της ανάλυσης συνεχίζεται με τις επόμενες γραμμές κώδικα.

```
    ...
    module = _analyze_import(name, locals)
    ...
```

Τώρα, καλείται η συνάρτηση `_analyze_import()`, η οποία περιέχει όλη τη δουλειά της ανάλυσης, με ορίσματα το απόλυτο όνομα (`name`) και την παράμετρο `locals`. Όταν η ανάλυση ολοκληρωθεί, το αποτέλεσμα, το οποίο είναι ένα αναλυμένο module του οποίου τα καλούμενα χαρακτηριστικά είναι περιτυλιγμένα, αποθηκεύεται στο `module`. Στις επόμενες γραμμές κώδικα χειρίζονται τα ονόματα, που δίνονται από το module όταν χρησιμοποιείται η δήλωση `import` ή η δήλωση `from import`.

```
    ...
    if not fromlist:
        if level == 0:
            return _analyze_import(name.partition('.')[0], locals)
        elif not name:  # Useless since we changed level to 0
            return module
        else:           # Useless since we changed level to 0
            cut_off = len(name) - len(name.partition('.')[0])
            return sys.modules[module.__name__[:len(module.__name__)-cut_off]]
    ...
```

Στον παραπάνω κώδικα, αν το `fromlist` είναι κενό, που σημαίνει ότι ο χρήστης της βιβλιοθήκης απλώς χρησιμοποιεί εντολή τύπου `import <module>`, τότε στην περίπτωση που το επίπεδο (`level`) ισούται με μηδέν, καλείται ξανά η `_analyze_import()`. Τα ορίσματα που παίρνει είναι η συμβολοσειρά μέχρι την πρώτη τελεία στο όνομα, δηλαδή το όνομα του module από όπου δίνεται κάποιο όνομα και το λεξικό `locals`. Στην περίπτωση που το όνομα είναι κενή συμβολοσειρά, απλώς επιστρέφεται το module. Στην περίπτωση που τίποτε από αυτά δεν είναι αληθές, δημιουργείται το `cut_off`, το οποίο είναι το μήκος του υπόλοιπου ονόματος χωρίς τη συμβολοσειρά μέχρι την πρώτη τελεία. Στη συνέχεια, το module με όνομα το συνολικό όνομα του module χωρίς το `cut_off` επιστρέφεται από τη μνήμη cache.

```
    ...
    elif hasattr(module, '__path__'):
        return _handle_fromlist(module, fromlist, _analyze_import, locals)
    ...
```

Αν το `fromlist` δεν είναι κενό και το module είναι πακέτο (μόνο τα πακέτα έχουν το `__path__` ως χαρακτηριστικό), τότε καλείται η συνάρτηση `_handle_fromlist()` με ορίσματα το `module`, το tuple `fromlist`, το αντικείμενο τύπου συνάρτηση `_analyze_import` και το λεξικό `locals`, αντίστοιχα. Στο τέλος, επιστρέφεται το module που προέρχεται από την κλήση της `_handle_fromlist()`.

```
    ...
    else:
        return module
```

Εάν τίποτα δεν είναι *True*, που σημαίνει ότι το `fromlist` δεν είναι άδειο ούτε το `module` δεν είναι πακέτο, αλλά είναι ένα module, απλώς επιστρέφεται αυτό το `module`.

_analyze_import

```
def _analyze_import(name, locals_):
    try:
        return sys.modules[name]
    except KeyError:
        pass
    ...
```

Τώρα, θα δημιουργηθεί η συνάρτηση `_analyze_import()`, η οποία εκτελεί περίπου τις ίδιες λειτουργίες εισαγωγής όπως κάνει η αρχική εισαγωγή, αλλά με κάποια πρόσθετη δουλειά πριν την εισαγωγή ενός module.

Αυτή η συνάρτηση παίρνει 2 ορίσματα, το όνομα (name) του module που θα εισαχθεί και το `locals_` που είναι η παράμετρος `locals` της `__import__()`. Στις πρώτες γραμμές του κώδικα, σε μία δήλωση `try..except`, το module που θα εισαχθεί φορτώνεται από τη μνήμη `sys.modules` μέσω του όνοματός του. Εάν εμφανιστεί `KeyError` και δεν υπάρχει ένα module στην μνήμη με αυτό το όνομα, η συνάρτηση συνεχίζεται. Διαφορετικά, το module που ήρθε από τη μνήμη επιστρέφεται και η συνάρτηση τερματίζεται. Αυτή η διαδικασία χρησιμοποιείται για να αποφευχθεί η εισαγωγή και η ανάλυση του ίδιου module πάνω από μία φορά. Οι επόμενοι κώδικες της συνάρτησης εκτελούνται μόνο όταν το module δεν υπάρχει ήδη στη μνήμη.

```
1       ...
2       path = None
3       if '.' in name:
4           pname, _, cname = name.rpartition('.')
5           if pname not in sys.modules:
6               _call_with_frames_removed(_analyze_import, pname, locals_)
7           pmodule = sys.modules[pname]
8           path = pmodule.__spec__.submodule_search_locations
9       for finder in sys.meta_path:
10          if 'find_spec' not in dir(finder):
11              # find and load module differently
12              finder = finder.find_module(name, path)
13              if finder is not None:
14                  module = finder.load_module(name)
15                  break
16          else:
17              ex_spec = finder.find_spec(name, path)
18              if ex_spec is not None:
19                  module = module_from_spec(ex_spec)
20                  sys.modules[name] = module
21                  break
22      else:
23          msg = f'No module named {name!r}'
24          raise ModuleNotFoundError(msg, name=name)
25      ...
```

Όπως δείχνει ο παραπάνω κώδικας, αρχικά το `path` του module αρχικοποιείται με *None*. Μετά, αν ένα module εισαχθεί απευθείας με την τελεία (.) χωρίς να εισαχθεί από ένα πακέτο, παραλείπεται ο κώδικας μέχρι και τη γραμμή 8. Διαφορετικά, η ανάλυση συνεχίζεται από τη γραμμή 4. Εδώ, με το όνομα στη μορφή `package.module`, χρησιμοποιώντας τη μέθοδο της κλάσης συμβολοσειρά (string) `rpartition()`, το όνομα χωρίζεται σε `pname` (parent name) και `cname` (child name). Έπειτα, ελέγχεται αν το `pname` δεν βρίσκεται στη μνήμη `sys.modules` για να αποφευχθούν περίεργες συμπεριφορές και μετά εισάγεται και αναλύεται πρώτα το `pmodule` (parent module) χρησιμοποιώντας το `pname` και την `_analyze_import()`. Αργότερα, το μονοπάτι του `pmodule` αποθηκεύεται στο `path`. Καθώς η ανάλυση συνεχίζεται στη γραμμή 9, το `spec` του module λαμβάνεται χρησιμοποιώντας έναν `finder`, ο οποίος βρίσκεται από το `name` και το `path`. Τέλος, χρησιμοποιώντας κάποιον loader, φορτώνεται το module (ένας loader βρίσκεται μέσα στη συναρτήσεις `module_from_spec()` και `load_module()`).

```
1       ...
2       if locals_:
3           if '__file__' in locals_:  # Ignore block when we import from interpreter (REPL)
4               caller_path = locals_['__file__']
5
```

```
6               if caller_path not in access.keys():
7                   """if it's the first time a function is called
8                   create a new dictionary item for access
9                   and put the message inside of it
10                  """
11                  access[caller_path] = {"import": "allow"}
12              else:
13                  """else, add the message as element in the list
14                  with the path of the module imported as key
15                  """
16                  access[caller_path]["import"] = "allow"
17      ...
```

Εδώ παρουσιάζεται ένα από τα πιο σημαντικά μέρη της ανάλυσης. Το μονοπάτι του module από όπου εισήχθη το module που αναλύεται, δηλαδή του module από το οποίο κλήθηκε η ανάλυση, δίνεται ως κλειδί του λεξικού access. Μία συμβολοσειρά με την τιμή import δίνεται ως τιμή αυτού του λεξικού.

```
1       ...
2       prolObj = Prologue()
3       prologue = prolObj.prologue # Get the function wrapping code
4       ex_source = module.__spec__.loader.get_source(name) # Get an example code
5
6       if ex_source:
7           ex_source = ex_source.replace('str.maketrans', 'str.wrapped.maketrans')
8       else:
9           ex_source = ""
10
11      if "from __future__ import" in ex_source:
12          codeProlObj = compile(prologue, module.__spec__.origin, 'exec')
13          exec(codeProlObj, module.__dict__)
14          codeExObj = compile(ex_source, module.__spec__.origin, 'exec')
15          exec(codeExObj, module.__dict__)
16      else:
17          source = prologue + ex_source
18          codeObj = compile(source, module.__spec__.origin, 'exec')
19          exec(codeObj, module.__dict__)
20      ...
```

Στη συνέχεια δημιουργείται ένα αντικείμενο Prologue και ο κώδικας του προλόγου φορτώνεται απο εκεί. Μετά, φορτώνεται ο κώδικας του module (`ex_source`). Στη γραμμή 17, δημιουργείται ένας νέος κώδικας από το `prologue` και το `ex_source`. Αμέσως μετά, όπως δείχνει το παραπάνω απόσπασμα κώδικα, αυτός ο νέος κώδικας μεταγλωττίζεται και εκτελείται. Αλλά, εάν υπάρχει δήλωση `from __future__ import` στην αρχή του κώδικα προς ανάλυση και γραφτεί κώδικας πριν από αυτή τη δήλωση, θα εμφανιστεί ένα `SyntaxError`. Για να αντιμετωπιστεί αυτό το πρόβλημα, εναλλακτικά, πρώτα μεταγλωττίζεται και εκτελείται ο κώδικας του προλόγου και όταν τελειώσει, μεταγλωττίζεται και εκτελείται ο αναλύσιμος κώδικας, αλλά και τα δύο με το λεξικό του ίδιου module ως το `locals` του.

```
1       ...
2       _wrap_module_members(module)
3
4       if path is not None:
5           setattr(pmodule, cname, module)
6
7       return module
```

Στο τέλος της εκτέλεσης του κώδικα, όλα τα ορισμένα από το χρήστη Python χαρακτηριστικά του module περιτυλίγονται μέσω της συνάρτησης `_wrap_module_members()` και στη συνέχεια

το `module` τοποθετείται ως παιδί του προηγούμενου module γονέα. Τέλος, επιστρέφεται ένα τροποποιημένο `module`, του οποίου τα καλούμενα χαρακτηριστικά είναι περιτυλιγμένα.

Get builtins

```
import builtins

print(dir(builtins))
```

Για να ληφθούν όλα τα ενσωματωμένα χαρακτηριστικά, χρησιμοποιήθηκε η ενσωματωμένη συνάρτηση `dir()`. Αυτή η συνάρτηση επιστρέφει τα ονόματά τους σε μία λίστα από συμβολοσειρές. Μετά, αυτή η λίστα εκτυπώθηκε και προστέθηκε με hard-code στον πρόλογο, επειδή η κλήση `dir(builtins)` δεν λειτουργούσε σωστά όταν γραφόταν κατευθείαν εκεί.

Prologue

```
class Prologue:
    def __init__(self):
        self.prologue = " \
        \nimport builtins\n \
        from analysis import Wrapper\n\n \
        builtin1 = ['ArithmeticError', 'BlockingIOError', 'BrokenPipeError', \
        'BufferError', 'BytesWarning', 'ChildProcessError', 'ConnectionAbortedError', \
        'ConnectionError', 'ConnectionRefusedError', 'ConnectionResetError', \
        'EOFError', 'Ellipsis', 'EnvironmentError', 'FileNotFoundError', \
        'FloatingPointError', 'FutureWarning', 'GeneratorExit', 'ImportWarning', \
        'IndentationError', 'InterruptedError', 'IsADirectoryError', \
        'KeyboardInterrupt', 'LookupError', 'MemoryError', 'ModuleNotFoundError', \
        'None', 'NotADirectoryError', 'NotImplementedError', \
        'OverflowError', 'PendingDeprecationWarning', 'PermissionError', \
        'ProcessLookupError', 'RecursionError', 'ReferenceError', 'ResourceWarning', \
        'RuntimeWarning', 'StopAsyncIteration', 'SyntaxError', \
        'SyntaxWarning', 'SystemError', 'SystemExit', 'TabError', 'TimeoutError', \
        'UnboundLocalError', 'UnicodeDecodeError', 'UnicodeEncodeError', \
        'UnicodeTranslateError', 'UnicodeWarning', 'ZeroDivisionError', \
        '__build_class__', '__debug__', '__import__', '__loader__', 'abs', 'all', \
        'any', 'ascii', 'bin', 'breakpoint', 'callable', 'chr', 'classmethod', \
        'compile', 'complex', 'copyright', 'credits', 'delattr', 'dir', 'divmod', \
        'enumerate', 'eval', 'exec', 'exit', 'filter', 'format', 'frozenset', \
        'hasattr', 'hash', 'help', 'hex', 'id', 'input', 'isinstance', \
        'issubclass', 'iter', 'len', 'license', 'list', 'map', 'max', \
        'memoryview', 'min', 'next', 'object', 'oct', 'open', 'ord', 'pow', \
        'print', 'property', 'quit', 'range', 'repr', 'reversed', 'round', \
        'set', 'setattr', 'sorted', 'staticmethod', 'str', 'sum', 'super', \
        'tuple', 'type', 'vars', 'zip', 'getattr', 'locals', 'dict', 'bytes', \
        'bool', 'int', 'slice', 'bytearray', 'float', 'globals']\n\n \
        # Wrap all built-in functions\n \
        for b in builtin1:\n \
        \tlocals()[b] = Wrapper(getattr(builtins, b), 'builtins')\n"

    def getPrologue(self):
        return self.prologue
```

Μια ιδέα, για να υλοποιηθεί το εργαλείο ανάλυσης είναι να περιτυλιχθεί κάθε ενσωματωμένο και ορισμένο από το χρήστη χαρακτηριστικό της Python. Χαρακτηριστικό εννοούμε μια συνάρτηση, μια κλάση, μια συνάρτηση κλάσης ή μια μέθοδο κλάσης. Μπορεί να γίνει αυτή η διαδικασία, έτσι ώστε το εργαλείο να παρεμβαίνει σε κάθε κλήση αυτών των χαρακτηριστικών, αφού πρέπει να είναι καλούμενα, και να αποθηκεύει τη χρήση τους στο λεξικό access.

Για να γίνει το περιτύλιγμα των ενσωματωμένων χαρακτηριστικών, δημιουργείται μια κλάση με όνομα Prologue η οποία περιέχει μια μεταβλητή μέλος με το όνομα prologue. Ο πρόλογος, ή αλλιώς prologue, είναι μια συμβολοσειρά που περιέχει τον κώδικα της περιτύλιξης αυτών των χαρακτηριστικών. Αργότερα, δημιουργείται μια μέθοδο getter για να επιστραφεί αυτός ο πρόλογος και να μπορεί ο χρήστης να τον χρησιμοποιεί.

_wrap_module_members

```
1   def _wrap_module_members(module):
2       name = module.__name__
3
4       all_func = set([fn[0] for fn in getmembers(module, isfunction)])
5       all_class = set([c[0] for c in getmembers(module, isclass)])
6
7       an_func = set([fn[0] for fn in getmembers(sys.modules[__name__], isfunction)])
8       an_class = set([c[0] for c in getmembers(sys.modules[__name__], isclass)])
9
10      func = all_func - an_func
11      clas = all_class - an_class
12      ...
```

Η παραπάνω συνάρτηση παίρνει ένα module ως όρισμα. Στη γραμμή 2, παίρνει το όνομα του module από το όρισμα `module`. Στη συνέχεια, όλες οι συναρτήσεις και οι κλάσεις του module αποθηκεύονται ως σύνολα (sets) ονομάτων (τα ονόματα είναι συμβολοσειρές) στις μεταβλητές `all_func` και `all_class`, αντίστοιχα. Εκεί συμπεριλαμβάνονται και οι συναρτήσεις και οι κλάσεις της ανάλυσης. Αμέσως μετά, λαμβάνονται αποκλειστικά όλες οι συναρτήσεις και οι κλάσεις από το εργαλείο ανάλυσης. Αφού αφαιρέθηκαν τα χαρακτηριστικά αυτά της ανάλυσης από τις συνολικές συναρτήσεις και κλάσεις, δίνονται οι συναρτήσεις και οι κλάσεις αποκλειστικά του εισαγόμενου module. Αυτές αποθηκεύονται στη `func` και στη `clas` αντίστοιχα.

```
1       ...
2
3       for fn in func:
4           current_function = getattr(module, fn)
5           # Don't wrap functions from C extensions (CPython)
6           if isinstance(current_function, Wrapper):
7               is_function_or_lambda = isinstance(current_function.wrapped, \
8                                                  (types.FunctionType, types.LambdaType))
9           else:
10              is_function_or_lambda = isinstance(current_function, \
11                                                 (types.FunctionType, types.LambdaType))
12          if not is_function_or_lambda:
13              continue
14          # This way we can avoid rewrapping functions
15          if current_function.__module__ != module.__name__:
16              continue
17          wrap = Wrapper(current_function, name)
18          setattr(module, fn, wrap)
19
20      for c in clas:
21          if c not in INIT_MOD_ATTR:
22              caller_class = getattr(module, c)
23
24              # Don't wrap classes from C extensions
25              if '__dict__' not in dir(caller_class):
26                  # Emulate hasattr built-in function for wrapped builtins
27                  try:
28                      _ = caller_class.__slots__
```

```
29                      found = True
30                  except AttributeError:
31                      found = False
32                  if not found:
33                      continue
34
35              # This way we can avoid rewrapping classes
36              if caller_class.__module__ != module.__name__:
37                  continue
38
39              # Wrap class's methods and functions first
40
41              caller_class_methods = set([m[0] for m in getmembers(caller_class, ismethod)])
42
43              caller_class_functions = set([f[0] for f in getmembers(caller_class, isfunction)])
44
45              for m in caller_class_methods:
46                  wrap = Wrapper(getattr(caller_class, m), name, caller_class=caller_class)
47                  setattr(caller_class, m, wrap)
48              for f in caller_class_functions:
49                  wrap = Wrapper(getattr(caller_class, f), name, caller_class=caller_class)
50                  setattr(caller_class, f, wrap)
51
52              # Wrap the class
53
54              wrap = Wrapper(caller_class, name)
55              setattr(module, c, wrap)
```

Στη γραμμή 3 του παραπάνω αποσπάσματος κώδικα στην ίδια συνάρτηση, για κάθε όνομα συνάρτησης στο σύνολο func, δίνεται το αντικείμενο τύπου συνάρτηση από το module μέσω της ενσωματωμένης συνάρτησης getattr(). Αν αυτό το αντικείμενο προέλθει από μια επέκταση C η συνάρτηση παραλείπεται και η ανάλυση συνεχίζεται με τις επόμενες συναρτήσεις. Το ίδιο συμβαίνει όταν προέρχεται από διαφορετικό module και δεν ορίζεται στο τρέχον module. Σε άλλες περιπτώσεις, το αντικείμενο τύπου συνάρτηση (`current_function`) προστίθεται ως όρισμα στο Wrapper με το όνομα του module (name) στην οποία ανήκει. Μετά από αυτό, δημιουργείται ένα αντικείμενο τύπου Wrapper με το όνομα wrap και η συνάρτηση του αντίστοιχου module αντικαθίσταται με αυτό το αντικείμενο χρησιμοποιώντας την ενσωματωμένη συνάρτηση setattr().

Στη γραμμή 20, κάνουμε το ίδιο πράγμα με τις κλάσεις. Για κάθε όνομα κλάσης στο σύνολο clas, βεβαιωνόμαστε ότι η κλάση δεν είναι ένα από τα χαρακτηριστικά που συνοδεύουν κάθε module που εισάγεται. Αν είναι ένα από αυτά, τότε δεν θα τυλιχτεί. Διαφορετικά, συνεχίζουμε από τη γραμμή 22. Σε αυτή τη γραμμή, παίρνουμε την ίδια την κλάση από τη μονάδα που αντιστοιχεί στο όνομα της κλάσης και την αποθηκεύουμε στη μεταβλητή caller_class. Και πάλι, αν το module όπου ορίζεται αυτή η κλάση δεν είναι το module που εισάγουμε, άρα αυτό σημαίνει ότι πήραμε την κλάση από διαφορετικό module και όχι από αυτό που κάναμε εισαγωγή αυτήν τη στιγμή, τότε δεν τυλίγουμε αυτήν την κλάση. Επίσης, δεν τυλίγουμε τις C κλάσεις που έρχονται από modules επέκτασης C. Διαφορετικά, συνεχίζουμε από τη γραμμή 41. Εδώ, δημιουργούμε ένα σύνολο από όλες τις μεθόδους (strings) αυτής της κλάσης και το αποθηκεύουμε στη μεταβλητή caller_class_methods. Κάνουμε το ίδιο πράγμα για όλες τις συναρτήσεις της και αποθηκεύουμε το αντίστοιχο σύνολο στην caller_class_functions. Στις επόμενες γραμμές, για κάθε όνομα μεθόδου ή συνάρτησης στην caller_class_methods ή στην caller_class_functions αντίστοιχα, παίρνουμε την αντίστοιχη μέθοδο ή συνάρτηση από την αντίστοιχη κλάση χρησιμοποιώντας τη `getattr()` και την προσθέτουμε στο αντικείμενο τύπου Wrapper. Δηλαδή, δημιουργούμε αυτό το αντικείμενο παραθέτοντάς του ως όρισμα τη συγκεκριμένη μέθοδο ή συνάρτηση. Τα υπόλοιπα ορίσματα του Wrapper είναι το όνομα του module και η αντίστοιχη κλάση. Στη συνέχεια, αντικαθιστούμε αυτήν τη μέθοδο ή συνάρτηση με αυτό το αντικείμενο Wrapper.

Ως αποτέλεσμα, όλες οι μέθοδοι και οι συναρτήσεις κλάσης είναι τυλιγμένες. Τέλος τυλίγουμε την αντίστοιχη κλάση όπως κάναμε πριν με τις μεθόδους και τις συναρτήσεις αλλά με το 3ο όρισμα ως None, γιατί είναι κλάση.

Wrapper

```
1  class Wrapper:
2      ...
```

Σε αυτό το απόσπασμα κώδικα, ορίζουμε μια κλάση με το όνομα `Wrapper`, η οποία, όπως δηλώνει το όνομα, λειτουργεί ως ένα περιτύλιγμα καλέσιμων χαρακτηριστικών και καλείται κάθε φορά που καλείται ένα χαρακτηριστικό.

__init__ της κλάσης Wrapper:

```
1  def __init__(self, wrapped, mod_name, caller_class=None):
2      self.wrap_init = True
3      self.wrapped = wrapped
4      self.mod_name = mod_name
5      Wrapper.__module__ = mod_name
6      self.caller_class = caller_class
```

Αυτή η dunder μέθοδος της κλάσης Wrapper αρχικοποιεί τις μεταβλητές μέλους του Wrapper. Κάθε φορά που τυλίγουμε τα ενσωματωμένα χαρακτηριστικά και τα χαρακτηριστικά ενός module πριν και μετά την εκτέλεση του κώδικά του, αντίστοιχα, η __init__() καλείται με τη wrapped ως το 1ο όρισμα, τη mod_name ως το 2ο όρισμα και την caller_class ως το 3ο όρισμα. Η wrapped είναι το αρχικό χαρακτηριστικό στο οποίο η Wrapper αντιστοιχεί και η mod_name είναι το όνομα του module όπου αυτό το αρχικό χαρακτηριστικό είναι ορισμένο. Η caller_class είναι η κλάση όπου το χαρακτηριστικό είναι ορισμένο, αν και μόνο αν το χαρακτηριστικό είναι μέθοδος ή συνάρτηση αυτής. Διαφορετικά, είναι None. Δηλαδή, στην περίπτωση που το αρχικό χαρακτηριστικό είναι συνάρτηση ή κλάση, είναι None.

Στην __init__(), οι αντίστοιχες μεταβλητές μέλη, οι οποίες επικαλούνται από τη self, αρχικοποιούνται με αυτά τα ορίσματα. Υπάρχει επίσης μία ακόμα μεταβλητή μέλος με όνομα wrap_init, η οποία δημιουργείται και αρχικοποιείται με την τιμή True. Χρησιμοποιείται για να αντιμετωπίσουμε την ενσωματωμένη συνάρτηση getattr(), η οποία χρησιμοποιείται για να αρχικοποιήσουμε το locals με τυλιγμένες ενσωματωμένες ή ορισμένες από το χρήστη συναρτήσεις ή κλάσεις ή μεθόδους. Μόνο αυτή τη φορά, η getattr() θα αποκλειστεί και δε θα εμφανιστεί στο λεξικό access. Παρατηρούμε ότι η wrap_init επίσης βοηθάει τη setattr() να μην εμφανίζεται στο access, όταν χρησιμοποιείται εσωτερικά στην __init__() (πχ self.wrap_init = True), όπως θα δούμε στη setattr() αργότερα. Επίσης, στη γραμμή 5, το χαρακτηριστικό __module__ της κλάσης Wrapper αντικαθίσταται από το όνομα του module του αρχικού χαρακτηριστικού, δηλαδή το wrapped.__module__ ή mod_name.

getoriginalclass της κλάσης Wrapper:

```
1  def getoriginalclass(self):
2      return self.caller_class
```

Αυτή η μέθοδος απλά επιστρέφει τη μεταβλητή μέλος `caller_class` για να είναι ευκολότερη η πρόσβαση του χρήστη σε αυτή.

__call__ της κλάσης Wrapper:

```
def __call__(self, *args, **kwargs):
    if not (self.wrap_init and self.wrapped.__name__ == "getattr"):
        ...
    else:
        self.wrap_init = False
    ...
```

Αυτή η dunder μέθοδος καλείται κάθε φορά που ένα καλέσιμο αντικείμενο καλείται. Αυτή τη φορά, αυτή η μέθοδος είναι βασικά ένα άγκιστρο για την __call__ του αρχικού Python χαρακτηριστικού, δηλαδή της self.wrapped.

Πρώτα, ελέγχουμε αν η wrap_init είναι True και το όνομα του αρχικού χαρακτηριστικού είναι getattr ταυτόχρονα. Αυτή η δήλωση χρησιμοποιείται για να αποφευχθεί η προσθήκη της ενσωματωμένης συνάρτησης getattr() μέσα στο λεξικό access ή η επιβολή της στην ανάλυσή μας από ατύχημα. Αυτά συμβαίνουν όταν καλούμε αυτή τη συνάρτηση για να πάρουμε ένα χαρακτηριστικό από το module builtins και μετά να το περιτυλίξουμε όταν το βάζουμε στο locals (Prologue). Αλλά την ίδια στιγμή, μπορούμε να προσθέσουμε την getattr() στο access ή να την επιβάλουμε σε ένα αναλυόμενο πρόγραμμα, όταν την καλούμε από διαφορετικό κώδικα.

Αν δεν είναι και τα δύο True, συνεχίζουμε με τον κώδικα μέσα στη δήλωση if. Διαφορετικά, αλλάζουμε την τιμή της wrap_init σε False για την επόμενη φορά που θα καλέσουμε την τυλιγμένη getattr().

```
        ...
        if self.caller_class:
            key = self.mod_name + "." + self.caller_class.__name__ + "." + self.wrapped.__name__
        else:
            if self.mod_name == "builtins":
                key = self.wrapped.__name__
            else:
                key = self.mod_name + "." + self.wrapped.__name__
        ...
```

Μέσα στη δήλωση if, αρχικοποιούμε το key για να το τοποθετήσουμε στις προσβάσεις/άδειες ή να το χρησιμοποιήσουμε για να επιβάλουμε τις άδειες στο πρόγραμμα. Το να έχει η self.caller_class None, σημαίνει ότι χειριζόμαστε μία κλάση ή μία κανονική συνάρτηση, αλλιώς το να έχει τιμή σημαίνει ότι χειριζόμαστε μία μέθοδο ή μία συνάρτηση μέσα σε μία κλάση. Αν η self.wrapped, δηλαδή το αρχικό χαρακτηριστικό το οποίο αντιστοιχεί στο περιτύλιγμα, είναι μία μέθοδος ή συνάρτηση κλάσης εκτελούμε την εξής λειτουργία. Κρατούμε το όνομα του module και της κλάσης όπου αυτό ορίστηκε καθώς και το όνομα αυτού του χαρακτηριστικού, ως το key. Αν είναι συνάρτηση ή κλάση, στην περίπτωση που είναι ενσωματωμένη, κρατούμε μόνο το όνομα του χαρακτηριστικού ως key, αφού ξέρουμε ήδη ότι είναι ενσωματωμένη και δεν ενδιαφερόμαστε για το module του. Στην περίπτωση που είναι μία κανονική συνάρτηση ή κλάση, αυτή τη φορά κρατούμε επίσης το όνομα του module μαζί με το όνομά του χωρίς το όνομα της κλάσης αφού εδώ δεν έχουμε μία κλάση ιδιοκτήτη.

```
        ...
        if _SAVE_ACCESSES:
            val = "allow"
            f = list(sys._current_frames().values())[0]
            if '__file__' in f.f_back.f_globals:
                path = f.f_back.f_globals['__file__']
                if path not in access.keys():
                    access[path] = {key: val}
                else:
                    access[path][key] = val
        ...
```

Στην πρώτη γραμμή, χρησιμοποιώντας μία boolean καθολική μεταβλητή με όνομα _SAVE_ACCESSES, ελέγχουμε αν έχουμε την πρόθεση να αποθηκεύσουμε τις προσβάσεις σε ένα λεξικό και μετά σε

αρχείο ή αν έχουμε την πρόθεση να επιβάλουμε τις ήδη υπάρχουσες προσβάσεις στο πρόγραμμα. Αν αυτό είναι True, που σημαίνει ότι θα αποθηκεύσουμε τις προσβάσεις σε ένα αρχείο, αυτό το μπλοκ κώδικα θα εκτελεστεί. Διαφορετικά, αν θα επιβάλουμε τις προσβάσεις στο πρόγραμμα ελέγχοντάς τις το block κώδικα κάτω από τη δήλωση else θα εκτελεστεί, το οποίο είναι στο παρακάτω απόσπασμα κώδικα.

Στις γραμμές 3-10, όπου η _SAVE_ACCESSES είναι True, δημιουργούμε μία μεταβλητή val με τιμή τη συμβολοσειρά allow και μετά παίρνουμε το μονοπάτι του module, όπου το περιτυλιγμένο χαρακτηριστικό κλήθηκε. Για να το πετύχουμε αυτό, αποκτούμε το πρώτο από τα τρέχοντα frames που παίρνουμε από τη στοίβα κλήσεων (call stack) όταν καλούμε το συγκεκριμένο περιτυλιγμένο χαρακτηριστικό αφού μετατρέψαμε τα frames σε λίστα. Στη συνέχεια, παίρνουμε την τιμή του χαρακτηριστικού __file__ από το globals αυτού του frame και την αποθηκεύουμε στη μεταβλητή path. Αφού αποκτήσαμε το μονοπάτι του module, προσθέτουμε τη χρήση, δηλαδή το key και το val, σε ένα υπολεξικό μέσα στο λεξικό access. Στην περίπτωση που το υπολεξικό (η πρόσβαση) δεν υπάρχει, το δημιουργούμε και το προσθέτουμε εκεί. Το κλειδί του υπολεξικού είναι το όνομα (key) που αρχικοποιήσαμε.

```
1           ...
2           else:
3               enforce(access, key)
4           ...
```

Αν η _SAVE_ACCESSES είναι False καλούμε τη συνάρτηση `enforce()` με ορίσματα το λεξικό προσβάσεων και το key, που βασικά είναι το όνομα της άδειας, για να επιβάλουμε τις άδειες στο πρόγραμμα και στη συνέχεια δεν κάνουμε τίποτα άλλο εκεί, σχετικά με την ανάλυση.

Μετά την ολοκλήρωση ενός από τους βασικούς κώδικες της ανάλυσης, συνεχίζουμε με το χειρισμό των ορισμάτων του περιτυλιγμένου χαρακτηριστικού μέχρι να καλέσουμε το αρχικό. Αργότερα, επιστρέφουμε το αποτέλεσμά του και βγαίνουμε από τη μέθοδο `__call__()`.

enforce

```
1   def enforce(accesses, key):
2       for elem in accesses.values():
3           for access_name, val in elem.items():
4               if key == access_name:
5                   if val == "deny":
6                       raise AttributeError("Usage of '" + key + "' is not allowed!")
7                   if val == "allow":
8                       return
9       raise AttributeError("Usage of '" + key + "' is not allowed!")
```

Στην παραπάνω συνάρτηση, χρησιμοποιώντας ένα βρόχο for, ελέγχουμε αν η καταγραφή του ονόματος του συστατικού (το key) βρίσκεται μέσα στις προσβάσεις ή αν έχει την τιμή deny. Αν ο βρόχος for τερματίσει χωρίς να έχει βρεθεί η καταγραφή του ή βρεθεί με την τιμή deny, σηκώνουμε ένα AttributeError και σταματάμε το πρόγραμμα. Διαφορετικά, αν βρεθεί με την τιμή allow, επιστρέφουμε από τη συνάρτηση και συνεχίζουμε με τα επόμενα καλέσιμα συστατικά στον αναλυόμενο κώδικα μέχρι να ολοκληρωθεί η ανάλυση.

get built-in modules names

```
1   def generateBuiltinsNames():
2       modules = {
3               module
4               for _, module, package in list(pkgutil.iter_modules())
5               if package is False
```

```
6               }
7
8       site_packages = glob.iglob(os.path.join(os.path.dirname(os.__file__) \
9                       + '/site-packages', '*-info', 'top_level.txt'))
10
11      modules -= {open(txt).read().strip() for txt in site_packages}
12      ...
```

Εδώ, θα πάρουμε όλα τα ενσωματωμένα και της τυπικής βιβλιοθήκης modules. Πρώτα, παίρνουμε τα modules από τη λίστα pkgutil.iter_modules(), των οποίων το package είναι False, αφού τα modules έχουν το __package__ τους ως None και τα αποθηκεύουμε στο modules ως σύνολο (set). Μετά, κάνουμε glob όλα τα top_level.txt αρχεία που εγκαταστάθηκαν κάτω από τα site-packages και τα αποθηκεύουμε στη μεταβλητή site_packages.

Στη γραμμή 11, διαβάζουμε τα αρχεία για τα import ονόματα, τα οποία είναι στη site_packages ανοίγοντας κάθε αρχείο, διαβάζοντας τα περιεχόμενά του και αφαιρώντας κενά διαστήματα από αυτά με τη χρήση της μεθόδου συμβολοσειράς strip(). Αποθηκεύουμε αυτά τα αρχεία σε ένα σύνολο, το οποίο το αφαιρούμε από το σύνολο modules.

```
1       ...
2       system_modules = set(sys.builtin_module_names)
3
4       python_root = sysconfig.get_python_lib(standard_lib=True)
5       _, top_level_libs, _ = list(os.walk(python_root))[0]
6
7       saved = sorted(top_level_libs + list(modules | system_modules))
8
9       with open("saved.json", "w") as f:
10          json.dump(saved, f)
11
12      return saved
```

Στο παραπάνω απόσπασμα κώδικα, παίρνουμε σε ένα σύνολο τα ονόματα όλων των ενσωματωμένων module και τα αποθηκεύουμε στη system_modules. Μετά, παίρνουμε μόνο τα top-level πακέτα από την εγκατάσταση της Python και τα αποθηκεύουμε στην top_level_libs.

Στο τέλος, προσθέτουμε όλα τα top-level πακέτα, τα πηγαία modules και τα πακέτα συστήματος σε μία ταξινομημένη λίστα. Αποθηκεύουμε αυτή τη λίστα σε ένα αρχείο json με όνομα saved.json και το επιστρέφουμε. Τώρα, μπορούμε να πάρουμε αυτή τη λίστα και να τη χρησιμοποιήσουμε στη συνάρτηση get_python_library().

get_python_library

```
1   def get_python_library():
2       """Get all Python's native packages.
3
4       These are built-in or standard library packages.
5       Return them in a list of strings.
6       """
7       return [
8           "urllib3", "pkg_resources", "requests", "gi",
9           "__future__", "__pycache__", "_abc", "_ast", "_asyncio", "_bisect",
10          "_blake2", "_bootlocale", "_bz2", "_codecs", "_codecs_cn",
11          ...
12          "xml", "xmlrpc", "xxlimited", "xxsubtype", "zipapp", "zipfile",
13          "zipimport", "zlib"
14      ]
15
16
17  modules = get_python_library()
```

Στην παραπάνω συνάρτηση, απλά επιστρέφουμε τη λίστα με όλα τα ενσωματωμένα και τυπικής βιβλιοθήκης πακέτα, που περιγράψαμε στο get built-in modules names. Μετά, στη γραμμή 17, καλούμε αυτή τη συνάρτηση get_python_library() και αποθηκεύουμε τη λίστα στην καθολική μεταβλητή modules.

build_class_hook

```
1  build_class_orig = builtins.__build_class__
2
3
4  def build_class_hook(func, className, *baseClasses, **kw):
5      baseClasses = list(baseClasses)
6      for i in range(len(baseClasses)):
7          if isinstance(baseClasses[i], Wrapper):
8              baseClasses[i] = baseClasses[i].wrapped
9      baseClasses = tuple(baseClasses)
10     return build_class_orig(func, className, *baseClasses, **kw)
11
12
13 build_class_hook.__module__ = build_class_orig.__module__
14 builtins.__build_class__ = build_class_hook
```

Υπήρχε ένα πρόβλημα όπου μία ορισμένη από το χρήστη κλάση κληρονομούσε από μία ήδη περιτυλιγμένη κλάση, δηλαδή η βασική κλάση (base class) ήταν ήδη του τύπου Wrapper. Αποφασίσαμε να το λύσουμε δημιουργώντας άλλο ένα άγκιστρο, αυτή τη φορά για την ενσωματωμένη συνάρτηση __build_class__(). Η __build_class__() καλείται εσωτερικά από την Python κάθε φορά που δημιουργείται μία κλάση με τη δήλωση class. Μέσα σε αυτό το άγκιστρο, στις γραμμές 5-9, ακριβώς πριν καλέσουμε την αρχική __build_class__(), για να δημιουργήσουμε την κλάση, αφαιρούμε το περιτύλιγμα πιθανώς περιτυλιγμένων βασικών κλάσεων από τα ορίσματα baseClasses. Επειδή το baseClasses είναι tuple (πλειάδα), το μετατρέπουμε σε λίστα πριν την εκτύλιξη και στο τέλος το μετατρέπουμε πίσω σε tuple. Εξωτερικά της συνάρτησης του αγκίστρου, αλλάζουμε επίσης το χαρακτηριστικό __module__ της στο __module__ της αρχικής __build_class__(), για να δίνει το σωστό όνομα του module κάθε φορά που ο χρήστης επικαλείται το __module__ του αγκίστρου. Αυτή τη φορά, επειδή μιλάμε για τη __build_class__(), αυτό θα δείξει builtins.

exit_handler

```
1  def exit_handler():
2      print(access)
3      if _SAVE_ACCESSES:
4          save_accesses(sys.argv[2])
5
6
7  atexit.register(exit_handler)
```

Θα δημιουργήσουμε έναν χειριστή (handler), μία συνάρτηση την οποία θα καλούμε κάθε φορά που το πρόγραμμα τερματίζει, για να σώσουμε το καθολικό λεξικό access σε ένα αρχείο με όνομα πχ. access.json.

Μέσα σε αυτή τη συνάρτηση, πρώτα τυπώνουμε το λεξικό access απλά για λόγους testing. Ακολουθώντας αυτό, ελέγχουμε με μία δήλωση if εάν το πρόγραμμα αναλύθηκε για να αποθηκεύσει τις προσβάσεις σε ένα αρχείο καταγραφής (κατά προτίμηση json). Το όνομα και το μονοπάτι αυτού του αρχείου καταγραφής δίνεται από το χρήστη ο οποίος τρέχει την ανάλυση. Για να το κάνουμε αυτό, δημιουργήσαμε μία σταθερά (boolean καθολική μεταβλητή) με όνομα _SAVE_ACCESSES. Αν αυτή είναι True αποθηκεύουμε τις προσβάσεις (αλληλεπιδράσεις) σε ένα αρχείο και το πρόγραμμα

τερματίζει, αλλιώς αν είναι False το πρόγραμμα τερματίζει κατευθείαν χωρίς καμμία περαιτέρω ενέργεια.

Μετά από αυτό, ανοίγουμε το αρχείο access.json με τη λειτουργία w και παίρνουμε τη ροή από αυτήν την κλήση της ενσωματωμένης συνάρτησης open(). Ονομάζουμε αυτή τη ροή ως f. Μέσα στο ανοικτό αρχείο, καλώντας τη συνάρτηση dump() του πακέτου json με όρισμα αυτή τη ροή, γράφουμε το λεξικό access σε pretty mode (με indents) με τα κλειδιά του ταξινομημένα. Χάρη στη δήλωση with, το αρχείο κλείνει αυτόματα (η ροή f καθαρίζεται – gets flushed) και δε χρειάζεται να καλέσουμε την ενσωματωμένη συνάρτηση close() στο τέλος.

Κεφάλαιο 5

Παραδείγματα εφαρμογής του PySecu

Σε αυτό το κεφάλαιο, παρουσιάζονται μερικά παραδείγματα εφαρμογής με ή χωρίς τη χρήση του εργαλείου δυναμικής ανάλυσης στις βιβλιοθήκες left-pad και pythonbenchmark. Αφού επιδείξαμε την κανονική εκτέλεση και των δύο βιβλιοθηκών χωρίς την ανάλυση, θα επιδείξουμε την εκτέλεση του καθενός με τη χρήση της ανάλυσης δείχνοντας την λειτουργικότητα της εξαγωγής και της επιβολής αδειών σε καθένα από αυτά τα δύο παραδείγματα.

5.1 Εφαρμογή στη βιβλιοθήκη left-pad

Η συγκεκριμένη βιβλιοθήκη λήφθηκε από το PyPI [29] και είναι ένα μεμονωμένο module, το οποίο περιέχει μία μόνο συνάρτηση που ονομάζεται left_pad().

Ένας χρήστης μπορεί να προσθέσει έναν αριθμό από συγκεκριμένες συμβολοσειρές σε μία άλλη συμβολοσειρά αποφασίζοντας και δίνοντας αυτόν τον αριθμό με έναν ευκολότερο τρόπο από το να χρησιμοποιεί κατευθείαν μεθόδους συμβολοσειράς της Python όπως την rjust(). Για να χρησιμοποιήσει τη συνάρτηση left_pad(), ο χρήστης πρέπει πρώτα να την εισάγει από τη βιβλιοθήκη left_pad.

5.1.1 Normal test

left_pad.py

```
1  def left_pad(s, n, c):
2      return s.rjust(len(s) + n, c)
```

Η παραπάνω συνάρτηση προσθέτει έναν αριθμό από συμβολοσειρές, οι οποίες προσδιορίζονται από το 3ο όρισμα (c), στην αριστερή πλευρά της συμβολοσειράς, η οποία προσδιορίζεται από το 1ο όρισμα (s). Ο αριθμός των συμβολοσειρών για να προστεθούν στη συμβολοσειρά προσδιορίζεται από το 2ο όρισμα (n).

Στη συνέχεια, περιγράφουμε την εσωτερική υλοποίηση αυτής της συνάρτησης, προκειμένου να γίνει κατανοητή η λειτουργία της συνάρτησης και να εξεταστεί η αποτελεσματικότητα της ανάλυσης. Μέσα στη συνάρτηση, καλείται η μέθοδος rjust() της συμβολοσειράς s με όρισμα το μήκος της s συν πόσες συμβολοσειρές c θα προστεθούν. Αυτό είναι το μήκος του αποτελέσματος και η συμβολοσειρά η οποία θα προστεθεί. Στη συνέχεια, το αποτέλεσμα της rjust() επιστρέφεται από τη συνάρτηση left_pad().

testing.py

```
1  from left_pad import left_pad
2
3  s = "abc"
4  s = left_pad(s, 2, "+")
5  print(s)
```

Όπως φαίνεται στο παραπάνω απόσπασμα κώδικα, εισάγουμε τη συνάρτηση left_pad() από τη βιβλιοθήκη left_pad και από εδώ και πέρα μπορούμε να τη χρησιμοποιήσουμε χωρίς να χρειάζεται να ξέρουμε λεπτομέρειες για τον κώδικα της βιβλιοθήκης. Αυτή τη φορά, δημιουργούμε την κεντρική συμβολοσειρά abc και την αποθηκεύουμε στη μεταβλητή s. Μετά, καλούμε τη συνάρτηση left_pad() με 1ο όρισμα την s, με 2ο όρισμα έναν ακέραιο 2 και με 3ο όρισμα μία συμβολοσειρά +. Αναμένουμε ότι δύο + θα προστεθούν στην αριστερή πλευρά της συμβολοσειράς abc. Το αποτέλεσμα της left_pad() θα σωθεί στη μεταβλητή s. Στο τέλος, η s αντικαθίσταται με τη νέα συμβολοσειρά και τυπώνεται.

Εισάγουμε τον κώδικα του testing.py, που περιγράψαμε παραπάνω, για να τον τρέξουμε από διαφορετικό module.

test.py

```
1  import testing
```

Output

```
++abc
```

Η εκτύπωση της μεταβλητής δείχνει το αποτέλεσμα που αναμέναμε.

5.1.2 Εξαγωγή αδειών

Για να χρησιμοποιήσουμε το PySecu θα το εισάγουμε πριν τρέξουμε τον κώδικα του παραδείγματος. Αυτή τη φορά, ο κώδικας του παραδείγματος εκτελείται εισάγοντας το module testing. Ο σωστός τρόπος, για να τρέξει η ανάλυση, φαίνεται στον παρακάτω κώδικα.

test_analysis.py

```
1  import pysecu
2  import testing
```

Θα αναλύσουμε τον κώδικα του παραδείγματος, τον οποίο περιγράψαμε πριν. Έχουμε δύο επιλογές για να τρέξουμε την ανάλυση. Οι επιλογές είναι η -s που αντιστοιχεί στο save και η -e που αντιστοιχεί στο enforce. Αυτή τη φορά, θα χρησιμοποιθεί -s στο terminal, στην εντολή python. Η -s επιλογή αποθηκεύει τις άδειες, δηλαδή τις καταγραφές συστατικών της Python, που μπορούν να κληθούν, από τους χρήστες της ανάλυσης, μέσα στο αρχείο αδειών του οποίου το όνομα και το μονοπάτι δίνεται από αυτούς.

```
python3 test_analysis.py -s access.json
```

Output

```
++abc
```

Τα αποτελέσματα της ανάλυσης αποθηκεύονται στο access.json στο ίδιο directory όπου έτρεξε η ανάλυση. Το αρχείο access.json μπορεί να χρησιμοποιηθεί στο μέλλον με την άλλη επιλογή της ανάλυσης για να ελέγξουμε αν η βιβλιοθήκη που αναλύσαμε έχει αλλάξει χωρίς την έγκρισή μας. Τα περιεχόμενα αυτού του αρχείου json παρουσιάζονται παρακάτω.

access.json

```
{
    "/home/user/.local/lib/python3.8/site-packages/left_pad.py": {
        "getattr": "allow",
        "len": "allow",
        "locals": "allow"
    },
    "/home/user/python/PySecu_package/PySecu/tests/micro-benchmarks/left-pad/testing.py": {
        "getattr": "allow",
        "import": "allow",
        "left_pad.left_pad": "allow",
        "locals": "allow",
        "print": "allow"
    },
    "test_analysis.py": {
        "import": "allow"
    }
}
```

Όπως βλέπουμε από το αρχείο json, στο αρχείο testing.py (3ο στοιχείο του λεξικού) αντιλαμβανόμαστε ότι η συνάρτηση left_pad() από τη βιβλιοθήκη left_pad καλείται από το module testing (αρχείο testing.py). Το όνομα του κάθε στοιχείου εμφανίζεται με μία ένδειξη, αν επιτρέπεται η κλήση του ή όχι. Αντιλαμβανόμαστε επίσης τι συμβαίνει μέσα σε αυτή τη βιβλιοθήκη left_pad (1ο στοιχείο του λεξικού). Η καταγραφή της δήλωσης import σημειώνεται στο αρχείο json κάθε φορά που εισάγουμε ένα module από ένα άλλο module.

5.1.3 Επιβολή αδειών

Θα ελέγξουμε τι θα συμβεί αν ο κώδικας της βιβλιοθήκης αλλάξει, πχ. αν προστεθεί επιπλέον συνάρτηση χωρίς την έγκρισή μας. Αυτή τη φορά, θα χρησιμοποιήσουμε την επιλογή -e στο terminal, στην εντολή python. Η επιλογή -e, φορτώνει το αρχείο αδειών του οποίου το όνομα και η διαδρομή δίνονται από τον χρήστη της ανάλυσης και το χρησιμοποιεί, για να επιβάλει τις συγκεκριμένες άδειες στη βιβλιοθήκη. Το αρχείο αδειών είναι το αρχείο json, το οποίο δημιουργούμε και εκεί αποθηκεύουμε τις άδειες (προσβάσεις) όταν τρέχουμε την ανάλυση με την επιλογή -s. Αυτή τη φορά, το αρχείο json έχει όνομα access.json.

```
python3 test_analysis.py -e access.json
```

Είμαστε στο ίδιο directory όπου σώθηκε το αρχείο access.json και βρίσκεται ακόμη εκεί.

Output

```
AttributeError: Usage of 'left_pad.do_not_run_me' is not allowed!
```

Η πρόσθετη συνάρτηση do_not_run_me() είναι απλά μία συνάρτηση που τυπώνει ένα μήνυμα χαιρετισμού. Όπως μπορούμε να δούμε από το output της οθόνης, ακριβώς πριν εκτελεστεί αυτή η συνάρτηση, το PySecu τη μπλόκαρε, τύπωσε ένα μήνυμα σφάλματος και σταμάτησε αμέσως το πρόγραμμα, επειδή το όνομα αυτής της συνάρτησης έλειπε από το αρχείο αδειών (προσβάσεων).

Άλλο ένα παράδειγμα της επιλογής enforce (-e) είναι να αλλάξουμε μερικές τιμές από το αρχείο json σε deny, επειδή δεν θέλουμε οι αντίστοιχές τους συναρτήσεις να εκτελεστούν από μία ομάδα ανθρώπων ή από οποιονδήποτε. Για παράδειγμα, θα αλλάξουμε την τιμή της print σε deny. Το αρχείο access.json άλλαξε σε αυτό:

access.json

```
{
    "/home/user/.local/lib/python3.8/site-packages/left_pad.py": {
        "getattr": "allow",
        "len": "allow",
        "locals": "allow"
    },
    "/home/user/python/PySecu_package/PySecu/tests/micro-benchmarks/left-pad/testing.py": {
        "getattr": "allow",
        "import": "allow",
        "left_pad.left_pad": "allow",
        "locals": "allow",
        "print": "deny"
    },
    "test_analysis.py": {
        "import": "allow"
    }
}
```

Ας τρέξουμε τώρα την ανάλυση με την εντολή που ακολουθεί.

```
python3 test_analysis.py -e access.json
```

Output

```
AttributeError: Usage of 'print' is not allowed!
```

Βλέπουμε ότι το PySecu μπλόκαρε την εκτέλεση της print(), επειδή έχει την τιμή deny στο αρχείο json, άρα δε μπορούμε να εκτελέσουμε αυτή τη συνάρτηση.

5.2 Εφαρμογή στη βιβλιοθήκη pythonbenchmark

Αυτή η βιβλιοθήκη λήφθηκε από τη σελίδα του Karlheinzniebuhr στο GitHub [23]. Το ακόλουθο παράδειγμα είναι πιο πολύπλοκο από το προηγούμενο και χρησιμοποιεί πιο ρεαλιστικό πακέτο με ένα αρχείο __init__.py. Αυτό το πακέτο μπορεί να συγκρίνει τους χρόνους δύο συναρτήσεων ή μεθόδων και να πει ποιο είναι γρηγορότερο ή αν έχουν τους ίδιους χρόνους εκτέλεσης. Μπορεί επίσης να μετρήσει το χρόνο μίας συγκεκριμένης συνάρτησης ή μεθόδου και να τον τυπώσει στο output. Στο ακόλουθο παράδειγμα θα συγκρίνουμε δύο συναρτήσεις.

5.2.1 Δοκιμή χωρίς την ανάλυση

συνάρτηση compare() στο pythonbenchmark/pythonbenchmark.py

```
def compare(functionA, functionB, times_average, *args, **kwargs):
    # loop n times
    i = 0
    totalA = 0.0
    totalB = 0.0

    '''
```

```
8       Normalizer
9       We intercalate the function calls because python tends to execute
10      the second function faster if we first loop over functionA and
11      then over functionB
12
13      '''
14      for i in range(times_average):
15          totalA += run_time(functionA,*args, **kwargs)
16          totalB += run_time(functionB,*args, **kwargs)
17
18      totalA = totalA/times_average
19      totalB = totalB/times_average
20
21      '''
22      1.001 or 0.1% is the lowest granularity time.clock() can distinguish I've tested
23      it calling compare and passing the same function twice. So we assume that if the difference
24      of speed is less than 0.1% the functions take the same time to execute
25      '''
26
27      if totalA < totalB and (totalB/totalA > 1.001):
28          print("Result: " + functionA.__name__ + " is "
29              + str(totalB/totalA) + " times faster than " + functionB.__name__)
30      elif totalB < totalA and (totalA/totalB > 1.001):
31          print("Result: " + functionB.__name__+ " is "
32              + str(totalA/totalB) + " times faster than " + functionA.__name__)
33      else:
34          print(functionA.__name__+" and " + functionB.__name__+" have the same speed!")
```

Το παραπάνω απόσπασμα κώδικα είναι ο εσωτερικός κώδικας της συνάρτησης compare(). Πρώτα υπολογίζει τους μέσους χρόνους των δύο συναρτήσεων χρησιμοποιώντας έναν βρόχο for για τα αθροίσματα των χρόνων και μετά διαιρώντας τα αθροίσματα με τον αριθμό των φορών που κάθε συνάρτηση μετρήθηκε. Αυτοί οι κανονικοποιημένοι μέσοι χρόνοι αποθηκεύονται στις μεταβλητές totalA και totalB, αντίστοιχα.

Τέλος, χρησιμοποιώντας μία δήλωση if, στην οποία επίσης συγκρίνουν τα κλάσματά τους με το 1.001 ως το χαμηλότερο βαθμό λεπτομέρειας που η συνάρτηση time.clock() μπορεί να συγκρίνει, ένα αντίστοιχο μήνυμα σύγκρισης τυπώνεται.

__init__.py

```
1   from .pythonbenchmark import compare, measure
```

Το αρχείο __init__.py χρησιμοποιείται για την Python για να αναγνωρίζει τον φάκελο pythonbenchmark ως ένα πακέτο και απλώς εισάγουμε τις συναρτήσεις compare() και measure(). Ως αποτέλεσμα, ο χρήστης μπορεί να σηκώσει αυτές τις συναρτήσεις εύκολα απλά εισάγοντας το πακέτο pythonbenchmark.

testing.py

```
1   import pythonbenchmark
2   import time
3
4   a,b,c,d,e = 10,10,10,10,10
5   something = [a,b,c,d,e]
6
7   def myFunction(something):
8       time.sleep(0.4)
9
10  def myOptimizedFunction(something):
```

```
11      time.sleep(0.2)
12
13  # comparing test
14  pythonbenchmark.compare(myFunction, myOptimizedFunction, 10, input)
```

Είναι ένα παράδειγμα χρήσης του πακέτου pythonbenchmark. Εισάγουμε αυτό το πακέτο και το πακέτο time της Python για να το χρησιμοποιήσουμε απλά για δοκιμή. Μετά, δημιουργούμε πέντε μεταβλητές με την τιμή 10 και τις βάζουμε σε μία λίστα απλά για λόγους επίδειξης. Αργότερα, δημιουργούμε δύο συναρτήσεις δείγματα των οποίων οι χρόνοι θα συγκριθούν. Τέλος, καλούμε την συνάρτηση compare() για να συγκρίνουμε τους χρόνους τους.

test.py

```
1  import testing
```

Εισάγουμε πάλι το παραπάνω παράδειγμα κώδικα για να το τρέξουμε από διαφορετικό module.

Output

```
Result: myOptimizedFunction is 1.9997234297943856 times faster than myFunction
```

5.2.2 Εξαγωγή αδειών

test_analysis.py

```
1  import pysecu
2  import testing
```

Για να τρέξουμε τη βιβλιοθήκη της ανάλυσης, πρέπει να την εισάγουμε ακριβώς πριν το παράδειγμά μας, όπως κάναμε στο προηγούμενο παράδειγμα με τη βιβλιοθήκη left_pad(). Αυτή τη φορά, εκτελούμε την ακόλουθη εντολή για να δημιουργήσουμε το αρχείο json των προσβάσεων.

```
python3 test_analysis.py -s access.json
```

Output

```
Result: myOptimizedFunction is 1.9994693672385881 times faster than myFunction
```

access.json

```
1  {
2      "/home/user/.local/lib/python3.8/site-packages/pythonbenchmark/__init__.py": {
3          "getattr": "allow",
4          "import": "allow",
5          "locals": "allow"
6      },
7      "/home/user/.local/lib/python3.8/site-packages/pythonbenchmark/pythonbenchmark.py": {
8          "getattr": "allow",
9          "locals": "allow",
10         "print": "allow",
11         "pythonbenchmark.pythonbenchmark.run_time": "allow",
12         "range": "allow",
13         "str": "allow"
14     },
15     "/home/user/python/PySecu_package/PySecu/tests/micro-benchmarks/pythonbenchmark/testing.py": {
16         "getattr": "allow",
17         "import": "allow",
```

```
18          "locals": "allow",
19          "pythonbenchmark.pythonbenchmark.compare": "allow"
20      },
21      "test_analysis.py": {
22          "import": "allow"
23      }
24  }
```

Στο αρχείο access.json, το οποίο δημιουργήθηκε από το εργαλείο δυναμικής ανάλυσης, μπορούμε πραγματικά να δούμε από τα κλειδιά του λεξικού ότι χρησιμοποιήσαμε τα αρχεία __init__.py, pythonbenchmark.py και testing.py.

Στο __init__.py απλώς χρησιμοποιήσαμε τη δήλωση import, άρα τίποτα άλλο δεν εμφανίστηκε στις τιμές. Στο pythonbenchmark.py, όπου ορίζονται και υλοποιούνται οι συναρτήσεις compare(), measure() και run_time(), βλέπουμε ότι στη συνάρτηση compare() η run_time() και η ενσωματωμένη συνάρτηση range() καλούνται. Αυτές οι συναρτήσεις χρησιμοποιήθηκαν μέσα στο βρόχο for στην compare(). Η κλάση str χρησιμοποιήθηκε στην κλήση της συνάρτησης print() στο τέλος, τα οποία και τα δύο εμφανίστηκαν στις τιμές του λεξικού. Μπορούμε να δούμε ότι η συνάρτηση compare() κλήθηκε μέσα στο αρχείο testing.py και ότι εισάγαμε κάθε module χρησιμοποιώντας τη δήλωση import.

5.2.3 Επιβολή αδειών

Ας τρέξουμε την ακόλουθη εντολή, για να φορτώσουμε τις ίδιες προσβάσεις και να τις επιβάλουμε στο πρόγραμμα. Προσθέσαμε μία κλήση στην ενσωματωμένη συνάρτηση max() στον κώδικα της βιβλιοθήκης που πρόκειται να αναλυθεί. Το όνομα αυτής της συνάρτησης λείπει από το αρχείο accesses.json.

```
python3 test_analysis.py -e access.json
```

Output

```
AttributeError: Usage of 'max' is not allowed!
```

Όπως βλέπουμε από το output της οθόνης, ακριβώς πριν εκτελεστεί η ενσωματωμένη συνάρτηση max(), το PySecu την μπλόκαρε, εκτύπωσε ένα μήνυμα σφάλματος και σταμάτησε αμέσως το πρόγραμμα, επειδή το όνομα αυτής της συνάρτησης έλειπε από το αρχείο προσβάσεων.

Άλλο ένα παράδειγμα της επιλογής enforce είναι να αλλάξουμε τις τιμές μερικών συναρτήσεων στο json αρχείο προσβάσεων σε deny. Πράγματι, αφού αλλάξαμε την τιμή της συνάρτησης run_time() σε deny και τρέξαμε την παραπάνω εντολή, το PySecu ομοίως τη μπλόκαρε και εκτύπωσε το παρακάτω μήνυμα σφάλματος:

Output

```
AttributeError: Usage of 'pythonbenchmark.pythonbenchmark.run_time' is not allowed!
```

Και στα δύο παραδείγματα, στο left-pad και στο pythonbenchmark, αν κρατήσουμε τη βιβλιοθήκη που θα αναλυθεί αμετάβλητη ή όλες τις προσβάσεις με την τιμή allow το πρόγραμμα θα εκτελεστεί κανονικά.

Κεφάλαιο 6

Αξιολόγηση

Σε αυτό το κεφάλαιο θα απαντηθούν τα παρακάτω ερωτήματα:

1. Ποια είναι η αποτελεσματικότητα εφαρμογής του εργαλείου PySecu όσον αφορά τη δυνατότητα ανίχνευσης προβλημάτων και την ως εκ τούτου αξιοπιστία του ως εργαλείου ανάλυσης ασφαλείας; (§6.2)

2. Ποιο είναι το χρονικό κόστος εκτέλεσης (χρόνος ανάλυσης) του εργαλείου PySecu που επιβάλλεται στην εκτέλεση των βιβλιοθηκών, που αναλύθηκαν; Ποια η σύγκρισή του με το αντίστοιχο χρονικό κόστος εκτέλεσης του ενσωματωμένου εργαλείου ανάλυσης API που παρέχεται από την Python, sys.settrace, που επιβάλλεται στην ανάλυση των ίδιων βιβλιοθηκών; (§6.3)

3. Ποια είναι η αποτελεσματικότητα της διαδικασίας μετασχηματισμού και ανάλυσης και πόσο αξιόπιστη παραμένει στην original -χωρίς ανάλυση- εκτέλεση του προγράμματος; (§6.4.1) Ποια η ταυτότητα των χαρακτηριστικών που μετασχηματίστηκαν; (§6.4.2)

6.1 Εξοπλισμός μετρήσεων

Το μηχάνημα που χρησιμοποιήθηκε για να τρέξουν οι μετρήσεις ήταν ένας φορητός υπολογιστής (laptop), ο οποίος έχει τα εξής χαρακτηριστικά:

- Intel Core i5-4210M CPU @ 2.60GHz × 4.
- 5.7GB μνήμης.
- Λειτουργικό σύστημα Ubuntu Linux 20.04.5 LTS

6.2 Ανίχνευση προβλημάτων - Ασφάλεια

Η δυνατότητα ανίχνευσης προβλημάτων, που προκαλείται από τη χρήση βιβλιοθηκών τρίτων, αντιμετωπίζεται από την ανάλυση με το PySecu η οποία αποδεικνύεται αποτελεσματική στην περίπτωση, που μια μη επιτρεπόμενη ενέργεια έχει προστεθεί στη βιβλιοθήκη, όταν αυτή είναι δημόσια. Στην περίπτωση αυτή, το PySecu σταματά αμέσως την εκτέλεση της βιβλιοθήκης. Η διαδικασία υλοποίησης της ανάλυσης έχει αναλυτικά αναφερθεί στο κεφάλαιο 3. Η βασική λειτουργία του αφορά στην απαγόρευση πρόσβασης που γίνεται κατά τη χρήση της enforce (-e) επιλογής της ανάλυσης, όταν

κάποιο καλούμενο χαρακτηριστικό (πχ. συνάρτηση) δεν υπάρχει στο αρχείο προσβάσεων ή υπάρχει με την ένδειξη deny. Αντίθετα, όταν το χαρακτηριστικό υπάρχει στο αρχείο προσβάσεων με την ένδειξη allow, η εκτέλεση συνεχίζεται μέχρι την ολοκλήρωση της βιβλιοθήκης. Η απόδοση της ανάλυσης (ανάλυση ασφαλείας) του εργαλείου εξαρτάται σε μεγάλο βαθμό από το πλήθος των περιτυλιγμένων χαρακτηριστικών και το ποσοστό τους σε σχέση με το σύνολο των χαρακτηριστικών της βιβλιοθήκης εφόσον κατά βάση και σύμφωνα με τη δομή του εργαλείου μόνο αυτά μπορούν να κληθούν και να αναλυθούν. Στοιχεία για τα περιτυλιγμένα χαρακτηριστικά που αναλύθηκαν, καταγράφονται στην παράγραφο 6.4.2 και στο παράρτημα C.2.

6.3 Επιβολή χρονικού κόστους εκτέλεσης

Χρησιμοποιήθηκαν 25 μικρο-πακέτα (micro-benchmarks) από το PyPI [32] ή από το GitHub [12].

Μετρήθηκαν οι χρόνοι για την ανάλυση κάθε μικρο-πακέτου, με και χωρίς τη χρήση του εργαλείου αδρομερούς δυναμικής ανάλυσης PySecu, καθώς και με το built-in εργαλείο API sys.settrace, εκτελώντας 100 επαναλήψεις κάθε φορά, οι μέσοι όροι των οποίων καταγράφονται στον Πίνακα 6.1., οι συγκρίσεις στους Πίνακες 6.2 και 6.3. Πρέπει να ληφθεί υπόψη ότι η διαφορά χρόνου μεταξύ της επιλογής save και enforce είναι αμελητέα.

Πίνακας 6.1: **Μέσοι χρόνοι ανάλυσης και απλής εκτέλεσης ανά βιβλιοθήκη.**

AVERAGE (ms)			
	original	PySecu	settrace
validus	28.70	101.57	76.89
bay-adder	20.74	65.73	24.35
binary_search	25.83	66.17	50.06
bit_array	22.20	73.83	26.18
split	27.42	58.50	88.81
first	24.30	54.19	42.00
first_dup	22.91	54.90	23.73
left-pad	21.43	53.63	23.77
max-ordered-diff	21.00	58.29	22.84
pad-on-left	21.61	55.00	26.10
pyfancy	22.34	110.41	27.82
value-lookup	21.67	55.02	26.27
fibonacci_index	27.49	67.16	61.18
ordered-set-stubs	27.64	66.22	55.85
tuple-flatten	21.11	58.25	24.35
flatten-xyz	21.23	73.90	26.38
json-flatten	24.25	71.86	46.01
range-regex	21.62	80.67	25.38
range-digit	24.39	83.26	36.50
pybite	24.43	84.07	45.37
chunky	25.82	68.03	57.71
length	22.85	50.46	25.71
set-algebra	23.72	162.57	32.07
boxing	247.18	438.85	1,929.07
right-triangle	22.38	54.13	25.67
AVERAGE (or,an)	32.57	86.67	114.00
MIN (or, an)	20.74	50.46	22.84
MAX (or, an)	247.18	438.85	1,929.07

Πίνακας 6.2: **Ποσοστιαία χρονική καθυστέρηση ανάλυσης.**

DIFFERENCE (%)		
	PySecu	settrace
validus	253.88	167.89
bay-adder	216.90	17.37
binary_search	156.15	93.79
bit_array	232.60	17.93
split	113.36	223.94
first	123.02	72.86
first_dup	139.63	3.59
left-pad	150.32	10.95
max-ordered-diff	177.56	8.77
pad-on-left	154.47	20.75
pyfancy	394.28	24.56
value-lookup	153.86	21.20
fibonacci_index	144.34	122.59
ordered-set-stubs	139.54	102.04
tuple-flatten	175.94	15.34
flatten-xyz	248.13	24.25
json-flatten	196.37	89.75
range-regex	273.08	17.35
range-digit	241.41	49.65
pybite	244.18	85.73
chunky	163.46	123.50
length	120.80	12.52
set-algebra	585.31	35.18
boxing	77.54	680.44
right-triangle	141.90	14.73
AVERAGE (%)	200.72	82.27
MIN (%)	77.54	3.59
MAX (%)	585.31	680.44
MEDIAN (%)	163.46	24.56

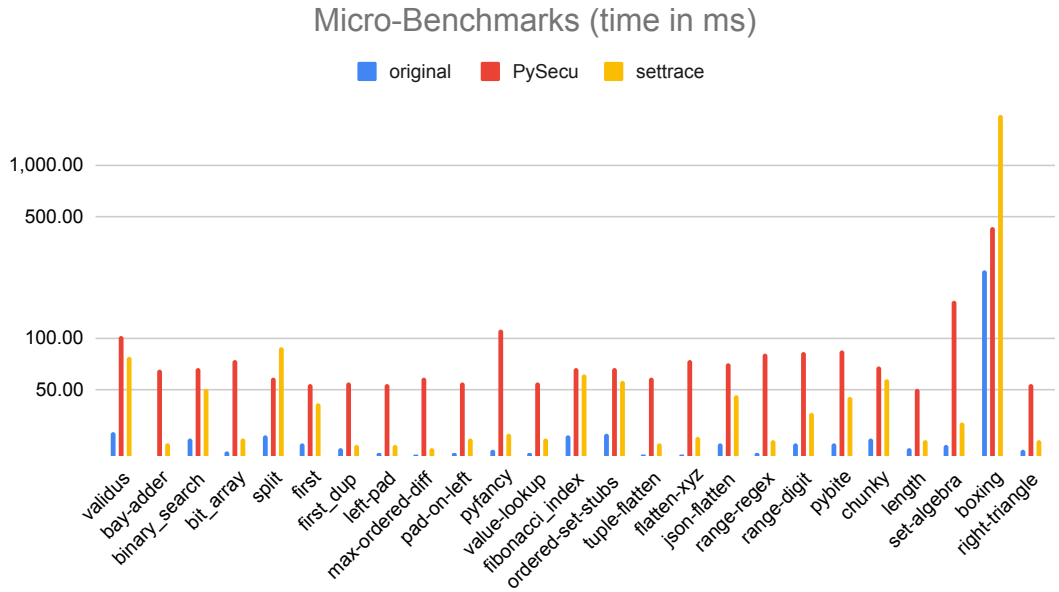
Πίνακας 6.3: **Συντελεστής χρόνου ανάλυσης επί της απλής εκτέλεσης.**

Συντελεστής χρόνου ανάλυσης (x)		
	PySecu	sys.settrace
validus	3,5	2,7
bay-adder	3,2	1,2
binary_search	2,6	1,9
bit_array	3,3	1,2
split	2,1	3,2
first	2,2	1,7
first_dup	2,4	1
left-pad	2,5	1,1
max-ordered-diff	2,8	1,1
pad-on-left	2,5	1,2
pyfancy	4,9	1,2
value-lookup	2,5	1,2
fibonacci_index	2,4	2,2
ordered-set-stubs	2,4	2
tuple-flatten	2,8	1,2
flatten-xyz	3,5	1,2
json-flatten	3	1,9
range-regex	3,7	1,2
range-digit	3,4	1,5
pybite	3,4	1,9
chunky	2,6	2,2
length	2,2	1,1
set_algebra	6,8	1,4
boxing	1,8	7,8
right-triangle	2,4	1,1
Average	3	1,8
Min	1,8	1
Max	6,8	7,8

Με τη χρήση του PySecu ο μέσος χρόνος ανάλυσης κυμαίνεται από 1,8x – 6,8x σχεδόν τριπλασιάζεται (3x), εμφανίζοντας μέση ποσοστιαία χρονική καθυστέρηση από την original εκτέλεση 200,72%. Ο μέγιστος χρόνος είναι 6,8x στο μικρο-πακέτο set-algebra και ο ελάχιστος είναι 1,8x στο μικρο-πακέτο boxing. Αυτό γίνεται επειδή κάθε φορά που εισάγουμε ένα στοιχείο, εκτελούμε τον κώδικα του προλόγου μας, πριν από τον κύριο κώδικα του στοιχείου και μετά την εκτέλεση του κώδικα του στοιχείου (της πηγής) περιτυλίγουμε όλα τα στοιχεία της βιβλιοθήκης με άγκιστρα Wrapper. Το τελευταίο περιτύλιγμα παίρνει αρκετό χρόνο για να ανακτήσει όλα τα στοιχεία της βιβλιοθήκης (συναρτήσεις, κλάσεις) και στη συνέχεια να τα αντικαταστήσει μέσα στους βρόχους. Επίσης, προστίθεται χρόνος για πιθανό ξετύλιγμα ορισμάτων, όποτε ο χρήστης καλεί αυτά τα άγκιστρα, και για πιθανό ξετύλιγμα των βασικών κλάσεων κάθε φορά που υπάρχει κληρονομικότητα κλάσης.

Αντίστοιχα η εφαρμογή του built-in εργαλείου API sys.settrace επιφέρει μέσο χρόνο ανάλυσης από 1x-7,8x περίπου 1,8x της original ανάλυσης και ποσοστιαία χρονική καθυστέρηση 82% από την original, με ελάχιστη χρονική επιβολή 1x στο πακέτο first_dup και μέγιστη 7,8x στο πακέτο boxing.

Στο παρακάτω Γράφημα 6.1, αποτυπώνονται σε ιστόγραμμα οι μέσοι χρόνοι ανάλυσης για το `PySecu` (κόκκινο χρώμα) και το `sys.settrace` (κίτρινο χρώμα) καθώς και ο original χρόνος εκτέλεσης των 25 βιβλιοθηκών (μπλε χρώμα).



Γράφημα 6.1: Σύγκριση χρόνων ανάλυσης βιβλιοθηκών με την original εκτέλεση.

Συγκρίνοντας τα δύο εργαλεία ανάλυσης PySecu και sys.settrace παρατηρούμε ότι στο πακέτο boxing το οποίο έχει υψηλό χρόνο original εκτέλεσης, ο χρόνος ανάλυσης του PySecu είναι 1,8x πολύ μικρότερος του sys.settrace που ανέρχεται στο 7,8x και στο split το PySecu δίνει 2,1x ενώ το sys.settrace 3,2x. Στα άλλα πακέτα όπως binary_search, fibonacci_index, ordered-set-stubs, chunky ο χρόνος ανάλυσης είναι περίπου στα ίδια επίπεδα, ενώ στα υπόλοιπα το PySecu καταγράφει μεγαλύτερη χρονική καθυστέρηση από το sys.settrace.

Συμπερασματικά καταλήγουμε ότι το PySecu έχει σε γενικές γραμμές το ίδιο χρονικό κόστος με το sys.settrace στα μικρο-πακέτα όπου ο χρόνος της ανάλυσης είναι πολύ μικρός. Στα πακέτα όμως

που ο χρόνος εκτέλεσης χωρίς ανάλυση είναι μεγάλος το PySecu φαίνεται να υπερτερεί σαφώς του sys.settrace και να δίνει χρόνο ανάλυσης πολύ μικρότερο. Επισημαίνεται ότι η χρονική επιβάρυνση μιας ανάλυσης δεν εξαρτάται μόνο από το εργαλείο αλλά και από τη διαδρομή της ανάλυσης ανάλογα με την εκάστοτε διεργασία εκτέλεσης του προγράμματος.

6.4 Στοιχεία ανάλυσης

6.4.1 Αποτελεσματικότητα - Αξιοπιστία ανάλυσης

Πραγματοποιήθηκαν δοκιμές των ενσωματωμένων στοιχείων της `Python` προκειμένου να διασφαλιστεί ότι με τη λειτουργία του `PySecu`, όλες οι αλληλεπιδράσεις των ενσωματωμένων στοιχείων της `Python`, ή (τουλάχιστον) ένας ικανοποιητικός αριθμός από αυτές, εμφανίζονται στο access.json. Οι δοκιμές που πραγματοποιήθηκαν αφορούσαν δοκιμές όλων των στοιχείων προκειμένου να αποδειχθεί ότι δεν υπάρχουν σφάλματα στα παραδείγματα του προγράμματος ή ότι τα προγράμματα δεν *κολλάνε*. Για να πραγματοποιηθούν οι δοκιμές, έγινε χρήση ενός αρχείου .py για testing του εργαλείου, για κάθε στοιχείο εκτός από μεμονωμένες περιπτώσεις όπου ένα αρχείο .py χρησιμοποιήθηκε για δοκιμή δύο στοιχείων. Όπως φαίνεται στο παράρτημα (§C.1), το αποτέλεσμα ήταν ενθαρρυντικό αφού σχεδόν όλα τα στοιχεία εμφανίζονταν από το `PySecu`. Τα μόνα στοιχεία που δεν λειτουργούν όταν είναι περιτυλιγμένα είναι τα ακόλουθα:

- **Exceptions/Warnings/Interrupts**: Αυτά τα στοιχεία, όταν είναι τυλιγμένα με το άγκιστρο Wrapper και χρησιμοποιούνται σε μια πρόταση `try..except` ή υψώνονται από τον προγραμματιστή, εμφανίζεται ένα `TypeError` με ένα μήνυμα ότι όλες οι εξαιρέσεις πρέπει να προέρχονται από τη `BaseException`. Δεν έχει (ακόμη) βρεθεί τρόπος να αντιμετωπιστεί αυτό.
- **int/float/bytes/bool/dict/list/set/str/slice/tuple/frozenset/bytearray στη δημιουργία τους και οι μέθοδοί τους**: Επειδή αυτά τα στοιχεία και οι μέθοδοί τους δημιουργούνται εσωτερικά στην `Python` χρησιμοποιώντας τη γλώσσα προγραμματισμού `C`, δεν είναι δυνατό να περιτυλιχθούν εντελώς σε `Python`. Οι μέθοδοί τους είναι επίσης αμετάβλητες και η `Python` δεν επιτρέπει την αντικατάστασή τους. Άρα, δεν μπορούμε να τα περιτυλίξουμε με τα αντίστοιχα άγκιστρά τους.
- **Printers/Quitters/Helpers**: Τα στοιχεία αυτών των τύπων, για παράδειγμα, `copyright`, `credits`, `help`, `license`, `quit` και `exit` δεν εμφανίζονται στο access.json και, όταν καλούνται, εμφανίζεται ένα `AttributeError`, επειδή αυτές οι κλάσεις δεν διαθέτουν `__name__` χαρακτηριστικό.

Μετά τη δοκιμή όλων των ενσωματωμένων στοιχείων της Python, από ένα σύνολο 147 στοιχείων περιτυλίγουμε τα 78, από τα οποία προκύπτουν 64 που εκτελούν την ανάλυση και 14 που δεν λειτουργούν. Δηλαδή, με το 18% των στοιχείων να μη λειτουργούν, γιατί πιθανόν το `PySecu` ορισμένες ακραίες περιπτώσεις αδυνατεί τις χειριστεί, θεωρούμε ότι στα περισσότερα micro-benchmarks θα εκτελείται η ανάλυση και ενδεχομένως και σε ακόμη μεγαλύτερα και δημοφιλή πακέτα. Αυτά τα θέματα είναι υπό διερεύνηση και ενδεχομένως να διορθωθούν σε μελλοντικές εκδόσεις του `PySecu`.

6.4.2 Ταυτότητα μετασχηματισμένων χαρακτηριστικών

Το `PySecu` που βασίζεται στην αδρομερή δυναμική ανάλυση στηρίζεται στην περιτύλιξη των περισσοτέρων στοιχείων της `Python`. Από τις συναρτήσεις που περιτυλίχθηκαν, το 5,6% των συναρτήσεων όλων των μικρο-πακέτων που δοκιμάσαμε (βλ. προηγούμενη ενότητα (§6.3)) δημιουργήθηκαν από

τον χρήστη, δηλαδή τον προγραμματιστή που δημιούργησε την αναλυόμενη βιβλιοθήκη και το 94,4% από τις συναρτήσεις ήταν οι ενσωματωμένες συναρτήσεις της Python. Επίσης, σχετικά με τις κλάσεις που περιτυλίχθηκαν, μόνο το 0,4% δημιουργήθηκε από τον χρήστη και το 99,6% ήταν οι ενσωματωμένες κλάσεις της Python. Τα ποσοστά αυτών των συναρτήσεων και κλάσεων περιγράφονται πιο ξεκάθαρα στο παράρτημα C.2.

Τα ποσοστά δείχνουν ότι στη περίπτωση που έχουμε χρησιμοποιήσει μικροπακέτα, οι περιτυλιγμένες συναρτήσεις χρήστη είναι πολύ λιγότερες από τις περιτυλιγμένες ενσωματωμένες συναρτήσεις. Άρα, συνάγεται ότι ο προγραμματιστής κάθε βιβλιοθήκης ορίζει πολύ λιγότερες συναρτήσεις από αυτές που ήδη υπάρχουν στην `Python` ως ενσωματωμένες. Το ίδιο ισχύει και για τις κλάσεις.

Επιπλέον, είναι λιγότερο πιθανό για τον προγραμματιστή να ορίσει περισσότερες κλάσεις από συναρτήσεις και έτσι η `Python` έχει περισσότερες ενσωματωμένες κλάσεις παρά συναρτήσεις. Έτσι, η διαφορά μεταξύ των κλάσεων χρήστη και των ενσωματωμένων κλάσεων είναι μεγαλύτερη από τη διαφορά μεταξύ των συναρτήσεων χρήστη και των ενσωματωμένων συναρτήσεων. Υπενθυμίζεται ότι χάρη στο `PySecu` καταφέραμε να πάρουμε καθορισμένες από τον χρήστη και ενσωματωμένες συναρτήσεις ή κλάσεις και να μετρήσουμε τον αριθμό τους. Για να γίνει αυτό, προσθέσαμε μετρητές για τις συναρτήσεις ή κλάσεις και partial logic μέσα στο βρόχο for στο στάδιο περιτύλιξης.

Αντιθέτως, αν αναλύσουμε μεγάλες βιβλιοθήκες, δηλαδή μακρο-πακέτα, θα δούμε ότι το ποσοστό των συναρτήσεων ή κλάσεων που ορίζει ο χρήστης υπερβαίνει το ποσοστό των ενσωματωμένων συναρτήσεων ή κλάσεων.

Συνοψίζοντας, θεωρούμε ότι ο ad-hoc επιλεκτικός μετασχηματισμός των στοιχείων μιας βιβλιοθήκης ενδεχομένως να επηρεάζει το χρόνο ανάλυσης, γεγονός που μένει να επαληθευτεί σε μελλοντικές εργασίες σχετικά με το `PySecu`. Στη συνέχεια, μετρήσαμε τις προσβάσεις (αλληλεπιδράσεις) σε κάθε αρχείο access.json και αποθηκεύσαμε τα αποτελέσματα στον πίνακα 6.4.

Πίνακας 6.4: **Αριθμός των καταχωρημένων προσβάσεων στο αρχείο.**

Package Name	Accesses
validus	22
bay-adder	8
binary_search	10
bit_array	17
split	15
first	8
first_dup	6
left-pad	9
max-ordered-diff	6
pad-on-left	11
pyfancy	19
value-lookup	16
fibonacci_index	12
ordered-set-stubs	11
tuple-flatten	10
flatten-xyz	15
json-flatten	17
range-regex	20
range-digit	16
pybite	23
chunky	11
length	12
set-algebra	50
boxing	21
right-triangle	15

Γίνεται αντιληπτό ότι, παρόλο που το μικροπακέτο boxing χρειάζεται τον περισσότερο χρόνο για να ολοκληρώσει την εκτέλεσή του σε σύγκριση με τα άλλα μικροπακέτα, στο αρχείο του, access.json δεν είναι αυτό με τις περισσότερες προσβάσεις. Το set-algebra έχει το access.json με τις περισσότερες προσβάσεις και χρειάζεται λιγότερο χρόνο από το boxing.

Κεφάλαιο 7

Σύγκριση με άλλα εργαλεία

Στο κεφάλαιο αυτό εξετάζεται η σύγκριση του εργαλείου `PySecu` με άλλα εργαλεία που αναπτύσσονται για δυναμική ανάλυση προγραμμάτων σε Python (§7.1) και με εργαλεία που αναπτύσσονται για αδρομερή δυναμική ανάλυση προγραμμάτων σε άλλες σύγχρονες δυναμικές γλώσσες προγραμματισμού (πχ το Lya για τη JavaScript) (§7.2).

7.1 DynaPyt για Python

Όπως προαναφέρθηκε η δυναμική ανάλυση προγραμμάτων σε Python βρίσκεται σε πρωταρχικό στάδιο με αποτέλεσμα να μην έχουν αναπτυχθεί και εφαρμοστεί ακόμα ευρέως αντίστοιχα εργαλεία δυναμικής ανάλυσης. Παρότι αρκετές προτάσεις αναλύσεων έχουν κατά καιρούς υπάρξει π.χ. για τον εντοπισμό σφαλμάτων (debugging) [45], για την επιβολή differential privacy [1] ή για το slicing προγραμμάτων [8] δεν υπάρχει όμως αξιόλογη πρόοδος σε σύγκριση με άλλες δημοφιλείς δυναμικές γλώσσες, π.χ. JavaScript, η οποία έχει δει ένα κύμα δυναμικών αναλύσεων τα τελευταία χρόνια [4]. Ως εκ τούτου σύγκριση του `PySecu` με κάποιο δοκιμασμένο εργαλείο σε Python είναι σχεδόν αδύνατη και η όποια απόπειρα αποτελεί μια έμμεση προσέγγιση και με προϋποθέσεις. Τελευταία, αξιόλογη προσπάθεια αποτελεί η ανάπτυξη ενός τέτοιου εργαλείου, του `DynaPyt` [10]. Το `DynaPyt`, είναι το πρώτο εργαλείο γενικής χρήσης για δυναμική ανάλυση μεγάλου μεγέθους προγραμμάτων σε Python και επιτρέπει στους προγραμματιστές να εφαρμόσουν συνοπτικές αναλύσεις. Το `DynaPyt` διαθέτει τη δυνατότητα επιλεκτικής οργάνωσης και τροποποίησης της εκτέλεσης, ακολουθεί πιστά την original εκτέλεση χωρίς να αλλάζει όμως τις σημαντικές διαδικασίες του προγράμματος. Στον παρακάτω Πίνακα 7.1 φαίνεται η αξιολόγηση του `PySecu` με το `DynaPyt` και με ad-hoc λύσεις καθώς και με την ενσωματωμένη API της Python, sys.settrace, όσον αφορά την πολυπλοκότητα του σχεδιασμού και τον επιπλέον χρόνο εκτέλεσης που επιβάλλει η ανάλυση.

Πίνακας 7.1: **Αξιολόγηση PySecu με DynaPyt και άλλες εφαρμογές.**

	Πολυπλοκότητα σχεδιασμού	Χρόνος Ανάλυσης
Ad-hoc	Υψηλή	Χαμηλό (1x)
sys.settrace	Μέτρια	Υψηλό (1,6x-13x)
DynaPyt/TraceAll	Χαμηλή	Υψηλό (1,2x-16x)
DynaPyt/BranchCoverage	Χαμηλή	Χαμηλό (1x-2x)
DynaPyt/OnlyAdd	Χαμηλή	Χαμηλό (1x-2x)
PySecu	Χαμηλή	Χαμηλό (3x)

Η ανάλυση, TraceAll, είναι η πιο χρονοβόρα αφού ανιχνεύονται και αναλύονται όλες οι εκτελούμενες διαδικασίες του προγράμματος, ταχύτερη όμως από αναλύσεις με εργαλεία για άλλες γλώσσες, όπως το Jalangi για τη JavaScript και το RoadRunner για τη Java. Η επιβάρυνση που επιβάλλεται από την ανάλυση TraceAll είναι σχετικά υψηλή, συγκεκριμένα μεταξύ 1,2× και 16×, ενώ σε σχέση με το sys.settrace στις περισσότερες περιπτώσεις είναι αισθητά βραδύτερη. Η ανάλυση BranchCoverage ανιχνεύει τον έλεγχο εισόδων ροής του προγράμματος ορίζοντας μόνο τις εισόδους ροής. Η τρίτη ανάλυση OnlyAdd, εξαιρετικά χαμηλού χρόνου ανάλυσης, ανιχνεύει μόνο τις δυαδικές διαδικασίες μέσω του τελεστή +. Οι δυο τελευταίες ελαφριές αναλύσεις, BranchCoverage και OnlyAdd, επιβάλλουν μέτρια χρονική καθυστέρηση, συγκεκριμένα μεταξύ 1,0× και 2,0×. Όσον αφορά τη σύγκριση με το sys.settrace, οι BranchCoverage και OnlyAdd, είναι 5,6%-88,6% ταχύτερες από το sys.settrace, δηλαδή κατά μέσο όρο 1,6x-13x της απλής εκτέλεσης.

Λαμβάνοντας υπόψη τα παραπάνω και επιχειρώντας μια έμμεση σύγκριση, όσον αφορά το PySecu σε σχέση με τις τρείς αναλύσεις του DynaPyt, παρόλο που δοκιμάστηκαν σε διαφορετικές βιβλιοθήκες, συμπεραίνουμε ότι το PySecu επιβάλλει ως επί το πλείστον χρόνο ανάλυσης μικρότερο του TraceAll και λίγο μεγαλύτερο των BranchCoverage και OnlyAdd.

7.2 Εργαλεία δυναμικής ανάλυσης άλλων γλωσσών

Σε αντίθεση με την Python, για διευκόλυνση της εφαρμογής δυναμικών αναλύσεων, γενικής χρήσης εργαλεία δυναμικής ανάλυσης έχουν δημιουργηθεί για άλλες γλώσσες, π.χ., Pin [18], Valgrind [22] για έμφυτα δυαδικά αρχεία, DiSL [19] και RoadRunner [11] για Java, Jalangi [33] για JavaScript και Wasabi [17] για WebAssembly. Αυτά τα εργαλεία ανάλυσης έχουν αναπτυχθεί στην ίδια βασική ιδέα μιας εύκολης εφαρμογής κατά την εκτέλεση ενός προγράμματος, συνήθως με τη μορφή επανάκλησης ή αγκίστρων. Πρόσφατα αναπτύχθηκε το Lya [40], εργαλείο αδρομερούς δυναμικής ανάλυσης για JavaScript.

Αν και στις παραπάνω περιπτώσεις η οποιαδήποτε σύγκριση με το PySecu είναι επισφαλής, ωστόσο η σύγκριση με το Lya δίνει κάποια χρήσιμα γενικά συμπεράσματα, αφού πρόκειται επίσης για ισοδύναμο με το PySecu εργαλείο αδρομερούς δυναμικής ανάλυσης με χρήση της τεχνικής της Επαναπλαισιοποίησης Βιβλιοθηκών, για μια επίσης σύγχρονη δυναμική γλώσσα προγραμματισμού τη JavaScript. Συνοψίζοντας, το Lya εμφανίζει την ίδια τάξη μεγέθους μέση χρονική επιβάρυνση με το PySecu με το fine-grained εργαλείο Jalangi για JavaScript κατά μέσο όρο 87x της original εκτέλεσης.

Κεφάλαιο 8

Σχετικές εργασίες και Μελλοντικές πρωτοβουλίες

Όπως προαναφέρθηκε στην παράγραφο (§2.4) έχουν αναπτυχθεί αρκετά εργαλεία δυναμικής ανάλυσης σε σύγχρονες δυναμικές γλώσσες προγραμματισμού (πχ. `JavaScript`, `WebAssembly`, `Java`, κλπ.). Ελάχιστες προσπάθειες έχουν γίνει στην `Python`. Μια τέτοια προσπάθεια αφορά το πλαίσιο δυναμικής ανάλυσης γενικού σκοπού `DynaPyt` [10], με το οποίο σύγκριση του `PySecu` έχει εξεταστεί στην παράγραφο 7.1.

Προτάσεις για μελλοντική βελτίωση και επέκταση του `PySecu` καθώς και αξιοποίηση της τεχνικής της αδρομερούς δυναμικής ανάλυσης, αφορούν τις παρακάτω κατευθύνσεις:

- Μετασχηματισμός των χαρακτηριστικών που εισάγονται από έμφυτα πακέτα της `Python`, όπως πακέτα της τυπικής βιβλιοθήκης και ενσωματωμένα πακέτα ούτως ώστε όταν αυτά τα χαρακτηριστικά περιτυλίγονται, να λειτουργούν κανονικά, όταν ο κώδικας της αναλύσιμης βιβλιοθήκης εκτελείται. Δηλαδή να μειωθεί δραστικά το ποσοστό των περιτυλιγμένων στοιχείων που δεν λειτουργούν και να αυξηθεί η αξιοπιστία της ανάλυσης.

- Εξέλιξη του `PySecu` ώστε να μειωθεί ο χρόνος ανάλυσης που επιφέρει, επιλέγοντας ενδεχομένως το επιθυμητό επίπεδο και είδος μετασχηματισμού, χωρίς να διακυβεύεται η αξιοπιστία της ανάλυσης ασφαλείας.

- Εφαρμογή της τεχνικής της αδρομερούς δυναμικής ανάλυσης σε άλλες σύγχρονες δυναμικές γλώσσες προγραμματισμού, όπου δεν έχει υλοποιηθεί μέχρι σήμερα (πχ. `Lua`, `Ruby`).

- Δεδομένου ότι η γλώσσα προγραμματισμού `Python` πρώτα δημιουργήθηκε μέσω της `CPython`, που σημαίνει ότι αναπτύχθηκε στη γλώσσα προγραμματισμού `C`, και η `C` είναι πολύ γρηγορότερη από την `Python`, πολλές βιβλιοθήκες περιέχουν modules επέκτασης `C`. Αυτά τα modules και τα χαρακτηριστικά που εισάγονται από αυτά αγνοούνται από το `PySecu`. Προτείνεται να εξεταστεί η ανάλυση τέτοιων modules στην `Python`, αφού η `C` δεν είναι δυναμική γλώσσα και δεν είναι δυνατό να τα αναλύσει δυναμικά.

Κεφάλαιο 9

Συμπεράσματα

Σε αυτή τη διπλωματική εργασία, παρουσιάζεται το (allow/deny) εργαλείο δυναμικής ανάλυσης PySecu, συνταγμένο πλήρως σε Python και εύκολα εφαρμόσιμο. Αποτελεί το πρώτο εργαλείο που βασίζεται στην τεχνική της αδρομερούς δυναμικής ανάλυσης, για ανάλυση ασφαλείας προγραμμάτων Python, το οποίο ακολουθεί τον κώδικα εκτέλεσης των βιβλιοθηκών. Τροποποιεί χαρακτηριστικά (συναρτήσεις ή κλάσεις) των βιβλιοθηκών και κατά την εκτέλεσή τους τα αποθηκεύει ως προσβάσεις στο αρχείο προσβάσεων για μελλοντική χρήση. Η εφαρμογή του PySecu κατέδειξε ότι επιβάλλει μέσο χρόνο ανάλυσης με συντελεστή 3x στην χωρίς ανάλυση εκτέλεση των βιβλιοθηκών, που είναι στην ίδια περίπου τάξη μεγέθους με του ενσωματωμένου εργαλείου ανάλυσης API που παρέχεται από την Python, με πολύ ταχύτερα αποτελέσματα σε μεγαλύτερες βιβλιοθήκες. Τέλος, αρκετά ενθαρρυντικά είναι τα αποτελέσματά του σε σχέση με αυτά του πρόσφατα ανεπτυγμένου εργαλείου δυναμικής ανάλυσης γενικού σκοπού για Python, καθώς και αντίστοιχων εργαλείων άλλων σύγχρονων δυναμικών γλωσσών.

Διαθεσιμότητα: Το PySecu είναι δημόσια διαθέσιμο ως πακέτο της Python στο PyPI. Οι κώδικες των βιβλιοθηκών που αναλύθηκαν από το PySecu, είναι διαθέσιμοι στο GitHub μέσω του παραρτήματος C.3. Το PySecu δουλεύει και με τα scripts της Python και με το διερμηνέα της Python, γνωστό και ως REPL.

Βιβλιογραφία

[1] C. Abuah, A. Silence, D. Darais, and J. P. Near, "Dduo: General-purpose dynamic analysis for differential privacy," in *2021 IEEE 34th Computer Security Foundations Symposium (CSF)*, 2021, pp. 1–15.

[2] S. K. Aggarwal and M. S. Kumar, "Debuggers for programming languages," in *The Compiler Design Handbook: Optimizations and Machine Code Generation*, Y. Srikant and P. Shankar, Eds. Boca Raton, Florida, USA: CRC Press, 2003. [Online]. Available: https://www.taylorfrancis.com/chapters/mono/10.1201/9781420040579-13/debuggers-programming-languages-sanjeev-kumar-aggarwal-sarath-kumar-srikant-priti-shankar?context=ubx

[3] P. Amidon, E. Davis, S. Sidiroglou-Douskos, and M. Rinard, "Program fracture and recombination for efficient automatic code reuse," *2015 IEEE High Performance Extreme Computing Conference (HPEC)*, pp. 1–6, 2015.

[4] E. Andreasen, L. Gong, A. Møller, M. Pradel, M. Selakovic, K. Sen, and C.-A. Staicu, "A survey of dynamic analysis and test generation for javascript," in *ACM Computing Surveys*, vol. 50, no. 66, Sep. 2018, pp. 1–36. [Online]. Available: https://doi.org/10.1145/3106739

[5] I. Arvanitis, G. Ntousakis, S. Ioannidis, and N. Vasilakis, "A systematic analysis of the event-stream incident," in *15th European Workshop on Systems Security (EUROSEC '22)*. New York, NY, USA: ACM, Apr. 2022, pp. 22–28. [Online]. Available: https://doi.org/10.1145/3517208.3523753

[6] AV-TEST - The Independent IT-Security Institute. Total amount of malware and pua. [Online]. Available: https://www.av-test.org/en/statistics/malware/

[7] Calltutor. What is hook in programming and why is it important. [Online]. Available: https://www.calltutors.com/blog/what-is-hook/

[8] Z. Chen, L. Chen, Y. Zhou, Z. Xu, W. C. Chu, and B. Xu, "Dynamic slicing of python programs," in *2014 IEEE 38th Annual Computer Software and Applications Conference*, 2014, pp. 219–228.

[9] L. Christophe, C. D. Roover, and W. D. Meuter, "Dynamic analysis using javascript proxies," vol. 2. Piscataway, NJ, USA: IEEE Press, May 2015, pp. 813–814. [Online]. Available: https://dl.acm.org/doi/10.5555/2819009.2819180#sec-ref

[10] A. Eghbali and M. Pradel, "DynaPyt: A dynamic analysis framework for Python," in *ESEC/FSE '22: 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. New York, NY, USA: ACM, 2022.

[11] C. Flanagan and S. N. Freund, "The roadrunner dynamic analysis framework for concurrent programs," in *Proceedings of the 9th ACM SIGPLAN-SIGSOFT workshop on Program analysis for software tools and engineering*. New York, NY, USA: ACM, Jun. 2010, pp. 1–8. [Online]. Available: https://doi.org/10.1145/1806672.1806674

[12] GitHub. (2022). [Online]. Available: https://github.com/

[13] intel. (2022) Dynamic analysis vs. static analysis. [Online]. Available: https://www.intel.com/content/www/us/en/develop/documentation/inspector-user-guide-windows/top/getting-started/dynamic-analysis-vs-static-analysis.html

[14] M. Keil and P. Thiemann, "Efficient dynamic access analysis using javascript proxies," in *Proceedings of the 9th Symposium on Dynamic Languages*, vol. 2. New York, NY, USA: ACM, May 2015, pp. 49–60. [Online]. Available: https://dl.acm.org/doi/10.1145/2578856.2508176

[15] J. Kreindl, D. Bonetta, L. Stadler, D. Leopoldseder, and H. Mössenböck, "Multi-language dynamic taint analysis in a polyglot virtual machine," in *Proceedings of the 17th International Conference on Managed Programming Languages and Runtimes*. New York, NY, USA: ACM, Nov. 2020, pp. 15–29. [Online]. Available: https://doi.org/10.1145/3426182.3426184

[16] T. Lauinger, A. Chaabane, S. Arshad, W. Robertson, C. Wilson, and E. Kirda, "Thou shalt not depend on me: Analysing the use of outdated javascript libraries on the web," 2017.

[17] D. Lehmann and M. Pradel, "Wasabi: A framework for dynamically analyzing webassembly," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*. New York, NY, USA: ACM, Apr. 2019, pp. 1045–1058. [Online]. Available: https://doi.org/10.1145/3297858.3304068

[18] C.-K. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S. Wallace, V. J. Reddi, and K. Hazelwood, "Pin: Building customized program analysis tools with dynamic instrumentation," in *Proceedings of the 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation*, ser. PLDI '05. New York, NY, USA: Association for Computing Machinery, Jun. 2005, p. 190–200. [Online]. Available: https://doi.org/10.1145/1065010.1065034

[19] L. Marek, A. Villazón, Y. Zheng, D. Ansaloni, W. Binder, and Z. Qi, "Disl: a domain-specific language for bytecode instrumentation," in *Proceedings of the 11th annual international conference on Aspect-oriented Software Development*. New York, NY, USA: ACM, Mar. 2012, pp. 239–250. [Online]. Available: https://doi.org/10.1145/2162049.2162077

[20] MBA. Modularity – definition and advantages. [Online]. Available: https://www.mbaknol.com/operations-management/modularity/

[21] J. Mickens, J. Elson, and J. Howell, "Mugshot: Deterministic capture and replay for javascript applications," in *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation*. USA: USENIX Association, 2010, p. 11.

[22] N. Nethercote and J. Seward, "Valgrind: A framework for heavyweight dynamic binary instrumentation," in *Proceedings of the 28th ACM SIGPLAN Conference on Programming Language Design and Implementation*, ser. PLDI '07. New York, NY,

USA: Association for Computing Machinery, 2007, p. 89–100. [Online]. Available: https://doi.org/10.1145/1250734.1250746

[23] K. Niebuhr. (2016) Learn to write faster code by experimenting with this library. [Online]. Available: https://github.com/Karlheinzniebuhr/pythonbenchmark

[24] N. Nikiforakis, L. Invernizzi, A. Kapravelos, S. V. Acker, W. Joosen, C. Kruegel, F. Piessens, and G. Vigna, "You are what you include: large-scale evaluation of remote javascript inclusions," in *Proceedings of the 2012 ACM conference on Computer and Communications Security*. New York, NY, USA: ACM, 2012, pp. 736–747.

[25] G. Ntousakis, S. Ioannidis, and N. Vasilakis, "Demo: Detecting third-party library problems with combined program analysis," in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*. New York, NY, USA: ACM, Nov. 2021, pp. 2429–2431. [Online]. Available: https://doi.org/10.1145/3460120.3485351

[26] D. L. Parnas. (1972) On the criteria to be used in decomposing systems into modules. [Online]. Available: https://dl.acm.org/doi/pdf/10.1145/361598.361623

[27] S. Pastrana and G. Suarez-Tangil. (Jul. 2019) A first look at the crypto-mining malware ecosystem: A decade of unrestricted wealth. [Online]. Available: https://www.slideshare.net/eraser/a-first-look-at-the-cryptomining-malware-ecosystem-a-decade-of-unrestricted-wealth

[28] C. Perlton. How to escape python script hell with modules packages. [Online]. Available: https://blog.inedo.com/python/modularization-and-packages#:~:text=Modularization%20is%20the%20technique%20of,functions%2C%20modules%2C%20and%20packages.

[29] M. Perry. (2016) Pad a string to the left with any number of characters. [Online]. Available: https://pypi.org/project/left-pad/

[30] A. Petukhov and D. Kozlov. (2008) Detecting security vulnerabilities in web applications using dynamic analysis with penetration testing. [Online]. Available: https://owasp.org/www-pdf-archive/OWASP-AppSecEU08-Petukhov.pdf

[31] Python. (2022) Python 3.8.13 documentation. [Online]. Available: http://docs.python.org/3.8/

[32] ——. (2022) Python package index. [Online]. Available: https://pypi.org/

[33] K. Sen, S. Kalasapur, T. Brutch, and S. Gibbs, "Jalangi: a selective record-replay and dynamic analysis framework for javascript," in *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*. New York, NY, USA: ACM, Aug. 2013, pp. 488–498. [Online]. Available: https://doi.org/10.1145/2491411.2491447

[34] S. Sidiroglou-Douskos, E. Lahtinen, F. Long, and M. Rinard, "Automatic error elimination by horizontal code transfer across multiple applications," *SIGPLAN Notices*, vol. 50, no. 6, pp. 43–54, June 2015. [Online]. Available: https://dl.acm.org/doi/10.1145/2813885.2737988

[35] H. Sun, D. Bonetta, C. Humer, and W. Binder, "Efficient dynamic analysis for node.js," in *Proceedings of the 27th International Conference on Compiler Construction*. New York, NY, USA: ACM, Feb. 2018, pp. 196–206. [Online]. Available: https://doi.org/10.1145/3178372.3179527

[36] M. Szydlowski, M. Egele, C. Kruegel, and G. Vigna, "Challenges for dynamic analysis of ios applications," in *Open Problems in Network Security*, vol. 7039. Lucerne, Switzerland: Springer, Jun. 2011, pp. 65–77. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-642-27585-2_6

[37] Y. Tanabe, T. Aotani, and H. Masuhara, "A context-oriented programming approach to dependency hell," 07 2018, pp. 8–14.

[38] N. Vasilakis, B. Karel, N. Roessler, N. Dautenhahn, A. DeHon, and J. M. Smith, "Towards fine-grained, automated application compartmentalization," in *Proceedings of the 9th Workshop on Programming Languages and Operating Systems*. New York, NY, USA: ACM, Oct. 2017, pp. 43–50. [Online]. Available: https://doi.org/10.1145/3144555.3144563

[39] ——, "Breakapp: Automated, flexible application compartmentalization," in *Networked and Distributed Systems Security*. San Diego, CA, USA: NDSS, Feb. 2018. [Online]. Available: http://dx.doi.org/10.14722/ndss.2018.23131

[40] N. Vasilakis, G. Ntousakis, V. Heller, and M. C. Rinard. (2021) Efficient module-level dynamic analysis for dynamic languages with module recontextualization (lya artifact). [Online]. Available: https://dl.acm.org/do/10.5281/zenodo.4902806/full/

[41] N. Vasilakis, C.-A. Staicu, G. Ntousakis, K. Kallas, B. Karel, A. DeHon, and M. Pradel, "Preventing dynamic library compromise on node.js via rwx-based privilege reduction," in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*. New York, NY, USA: ACM, Nov. 2021, pp. 1821–1838. [Online]. Available: https://dl.acm.org/doi/10.1145/3460120.3484535

[42] E. Westra, "Modular programming with python." Packt Publishing, May 2016. [Online]. Available: https://www.oreilly.com/library/view/modular-programming-with/9781785884481/ch05s04.html

[43] T. Würthinger, C. Wimmer, C. Humer, A. Wöß, L. Stadler, C. Seaton, G. Duboscq, D. Simon, and M. Grimmer, "Practical partial evaluation for high-performance dynamic language runtimes," vol. 52, no. 6. New York, NY, USA: ACM, Jun. 2017, pp. 662–676. [Online]. Available: https://doi.org/10.1145/3140587.3062381

[44] T. Würthinger, C. Wimmer, A. Wöß, L. Stadler, G. Duboscq, C. Humer, G. Richards, D. Simon, and M. Wolczko, "One vm to rule them all," in *Proceedings of the 2013 ACM international symposium on New ideas, new paradigms, and reflections on programming software*. New York, NY, USA: ACM, Oct. 2013, pp. 187–204. [Online]. Available: https://doi.org/10.1145/2509578.2509581

[45] Z. Xu, P. Liu, X. Zhang, and B. Xu, "Python predictive analysis for bug detection," in *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*. New York, NY, USA: ACM, Nov. 2016, pp. 121–132. [Online]. Available: https://doi.org/10.1145/2950290.2950357

Παραρτήματα

Παράρτημα Α

A Python Guide

A.1 General Information

Python is designed by Guido van Rossum and it is a high-level, interpreted, general-purpose programming language. It supports structured (particularly procedural), object-oriented and functional programming.

A.2 Install Python

It is recommended to install the latest version of Python (Python 3.8 and upper).

- For Linux:
  Just type in the terminal:

  ```
  sudo apt-get install python3.8
  ```

- For Windows:
  Go to Python's website and in Downloads select Windows. Then, select one of the Python releases (preferably stable) that suit your machine and follow the instructions in the installer.

After the installation, try to run `python -V` or just `python` on Linux terminal or Windows command prompt to check if the installation was successful.

A.3 Basic Python Commands

After installing Python and making sure that it is indeed installed, you can use the interpreter of the latest installed Python version by typing `python3`. You can type `help()` or `help(object)` inside this interpreter or you can access the Python docs [31].

Inside this interpreter, you can also run the following commands for example. But, it is better to run multiple commands as .py scripts just in case they need multiple lines of code.

A.3.1 A Hello World Example

```
1  print("Hello World!")
```

This command calls built-in function `print`, which takes a string as an argument (this time a *Hello World!*) and prints it out on the screen.

Output

```
Hello World!
```

A.3.2 Python Program to Check if a Number is Odd or Even

```
1  num = int(input("Enter a number: "))
2  if (num % 2) == 0:
3     print("{0} is Even".format(num))
4  else:
5     print("{0} is Odd".format(num))
```

The code above first prints a *Enter a number:* on the screen and waits for the user to type a line with the keyboard and press enter. We suppose that he indeed gives a number. This number is returned from the `input` built-in function as a `str` (string). Then, this number, because we want it to be an integer, gets converted to an `int` and gets stored in *variable* `num`.

Now, in lines 2-5, using the *%* operator, which is the operator of the remainder, we check with an `if` statement if the remainder of the operation *num % 2* is equal to zero, that is if the `num` is even. If it is equal to zero, then we `print` a message that it is even (with the `format` string method we just add the value of `num` into the string). Otherwise, we `print` a message that it is odd.

Output 1

```
Enter a number: 43
43 is Odd
```

Output 2

```
Enter a number: 18
18 is Even
```

Παράρτημα Β

Installing PySecu

B.1 Install PySecu from pip

First, you need to check if you have `pip` installed. To do this, type the following command:

```
python3 -m pip --version
```

or just:

```
pip3 --version
```

If `pip` isn't already installed, then first try to bootstrap it from the standard library:

```
python3 -m ensurepip --default-pip
```

Finally, to install our `PySecu`, type the following command:

```
pip3 install pysecu
```

B.2 Install PySecu from GitHub

From terminal or command prompt type the following command:

```
git clone https://github.com/andromeda/PySecu.git
```

This will create a `PySecu` folder with all the contents of the analysis tool inside of it. These contents are code, tests, license, etc.

Attention!! To use `PySecu` after installing it from GitHub, you should go to its directory:

```
cd PySecu
```

and type the following command:

```
pip3 install -e .
```

This will install the `PySecu` package locally in your computer in editable mode. After this, you can type `import pysecu`, **before** your analyzed code.

Παράρτημα C

Πληροφορίες Βιβλιοθηκών

C.1 Ενσωματωμένα (Built-in) στοιχεία

Σε αυτήν την ενότητα, παρουσιάζεται ο αναλυτικός Πίνακας C.1 με τα αποτελέσματα ανάλυσης με το PySecu κάθε ενσωματωμένου στοιχείου των βιβλιοθηκών που αναλύθηκαν, τα συμπεράσματα της οποίας συνοπτικά αναφέρθηκαν στο κεφάλαιο 6 (§6.4.1). Στον Πίνακα αυτόν καταγράφεται για κάθε στοιχείο η κατηγορία που ανήκει, ενδείξεις (Yes/No) για το αν είναι περιτυλιγμένο και για το αν λειτουργεί με το PySecu, καθώς και τον λόγο για τον οποίο δεν λειτουργεί.

Πίνακας C.1: **Ανάλυση ενσωματωμένων στοιχείων Python με το PySecu**

Native Name	Wrapped	Working	Reason	Category
print	Yes	Yes	-	function
abs	Yes	Yes	-	function
all	Yes	Yes	-	function
any	Yes	Yes	-	function
ascii	Yes	Yes	-	function
bin	Yes	Yes	-	function
breakpoint	Yes	Yes	-	function
callable	Yes	Yes	-	function
chr	Yes	Yes	-	function
classmethod	Yes	Yes	-	class
compile	Yes	Yes	-	function
complex	Yes	Yes	-	class
copyright	Yes	No	Same as license	printer
credits	Yes	No	Same as license	printer
delattr	Yes	Yes	-	function
dir	Yes	Yes	-	function
divmod	Yes	Yes	-	function
enumerate	Yes	Yes	-	function
eval	Yes	Yes	-	function
exec	Yes	Yes	-	function
exit	Yes	No	Attribute error in line 97: _Quitter has no attribute __name__	quitter
filter	Yes	Yes	-	class
format	Yes	Yes	-	function
frozenset	Yes	Yes	(frozenset methods are not showing up)	class
hasattr	Yes	Yes	-	function
hash	Yes	Yes	-	function
help	Yes	No	Attribute error in line 97: _Helper has no attribute __name__	helper
hex	Yes	Yes	-	function
id	Yes	Yes	-	function
input	Yes	Yes	-	function
isinstance	Yes	Yes	-	function
issubclass	Yes	Yes	-	function
iter	Yes	Yes	-	function
len	Yes	Yes	-	function
license	Yes	No	Attribute error in line 97: _Printer has no attribute __name__	printer
list	Yes	Yes	(list is not shown up when it is created with [] nor its methods)	class
map	Yes	Yes	-	class
max	Yes	Yes	-	function
memoryview	Yes	Yes	-	class
min	Yes	Yes	-	function
next	Yes	Yes	-	function
oct	Yes	Yes	-	function
open	Yes	Yes	-	function
ord	Yes	Yes	-	function
pow	Yes	Yes	-	function
property	Yes	Yes	-	class
quit	Yes	No	Same as exit	quitter
range	Yes	Yes	-	class
repr	Yes	Yes	-	function

reversed	Yes	Yes	-	class
round	Yes	Yes	-	function
set	Yes	Yes	-	class
setattr	Yes	Yes	-	function
sorted	Yes	Yes	-	function
staticmethod	Yes	Yes	-	class
str	Yes	Yes	(str is not shown up when it is created with "" nor its methods)	class
sum	Yes	Yes	-	function
super	Yes	Yes	-	class
tuple	Yes	Yes	(tuple is not shown up when it is created with () nor its methods)	class
type	Yes	Yes	-	class
vars	Yes	Yes	-	function
zip	Yes	Yes	-	class
getattr	Yes	Yes	-	function
locals	Yes	Yes	-	function
NameError	No			Errors/Warning
ImportError	No			Errors/Warning
Exception	No			Errors/Warning
object	Yes	Yes	-	class
FileExistsError	No			Errors/Warning
OSError	No			Errors/Warning
DeprecationWarning	No			Errors/Warning
UserWarning	No			Errors/Warning
ValueError	No			Errors/Warning
Warning	No			Errors/Warning
IOError	No			Errors/Warning
TypeError	No			Errors/Warning
RuntimeError	No			Errors/Warning
KeyError	No			Errors/Warning
AttributeError	No			Errors/Warning
dict	Yes	No	(dict is not showing up)	class
UnicodeError	No			Errors/Warning
bytes	Yes	No	(bytes is not showing up)	class
bool	Yes	No	(bool is not showing up)	class
int	Yes	No	(int is not showing up)	class
slice	Yes	No	(slice is not showing up)	class
IndexError	No			Errors/Warning
bytearray	Yes	No	(bytearray is not showing up)	class
float	Yes	No	(float is not showing up)	class
AssertionError	No			Errors/Warning
globals	Yes	No	(globals is not showing up)	function
ArithmeticError	No			Errors/Warning
BaseException	No			Errors/Warning
BlockingIOError	No			Errors/Warning
BrokenPipeError	No			Errors/Warning
BufferError	No			Errors/Warning
BytesWarning	No			Errors/Warning
ChildProcessError	No			Errors/Warning
ConnectionAbortedError	No			Errors/Warning
ConnectionError	No			Errors/Warning

ConnectionRefusedError	No			Errors/Warning
ConnectionResetError	No			Errors/Warning
EOFError	No			Errors/Warning
Ellipsis	No			ellipsis
EnvironmentError	No			Errors/Warning
FALSE	Yes	Yes	-	bool
FileNotFoundError	No			Errors/Warning
FloatingPointError	No			Errors/Warning
FutureWarning	No			Errors/Warning
GeneratorExit	No			Interrupt
ImportWarning	No			Errors/Warning
IndentationError	No			Errors/Warning
InterruptedError	No			Errors/Warning
IsADirectoryError	No			Errors/Warning
KeyboardInterrupt	No			Errors/Warning
LookupError	No			Errors/Warning
MemoryError	No			Errors/Warning
ModuleNotFoundError	No			Errors/Warning
None	Yes	Yes	-	NoneType
NotADirectoryError	No			Errors/Warning
NotImplemented	No			Errors/Warning
NotImplementedError	No			Errors/Warning
OverflowError	No			Errors/Warning
PendingDeprecationWarning	No			Errors/Warning
PermissionError	No			Errors/Warning
ProcessLookupError	No			Errors/Warning
RecursionError	No			Errors/Warning
ReferenceError	No			Errors/Warning
ResourceWarning	No			Errors/Warning
RuntimeWarning	No			Errors/Warning
StopAsyncIteration	No			Errors/Warning
StopIteration	No			Errors/Warning
SyntaxError	No			Errors/Warning
SyntaxWarning	No			Errors/Warning
SystemError	No			Errors/Warning
SystemExit	No			Errors/Warning
TabError	No			Errors/Warning
TimeoutError	No			Errors/Warning
TRUE	Yes	Yes	-	bool
UnboundLocalError	No			Errors/Warning
UnicodeDecodeError	No			Errors/Warning
UnicodeEncodeError	No			Errors/Warning
UnicodeTranslateError	No			Errors/Warning
UnicodeWarning	No			Errors/Warning
ZeroDivisionError	No			Errors/Warning
__build_class__	Yes	Yes	-	__<>__
__import__	Yes	Yes	-	__<>__
__loader__	No		(__loader__'s methods are not showing up)	__<>__

C.2 Περιτυλιγμένα (wrapped) στοιχεία

Σε αυτή την ενότητα καταγράφονται αναλυτικά στον Πίνακα C.2 για τα περιτυλιγμένα στοιχεία κάθε βιβλιοθήκης που αναλύθηκε, τα ποσοστά των συναρτήσεων (functions) και κλάσεων (classes) χρήστη (user) καθώς και τα αντίστοιχα ποσοστά για τα ενσωματωμένα (built-in). Συνολική προσέγγιση των παραπάνω αποτελεσμάτων φαίνεται σχηματικά (πίτα) στο παρακάτω Γράφημα C.4.

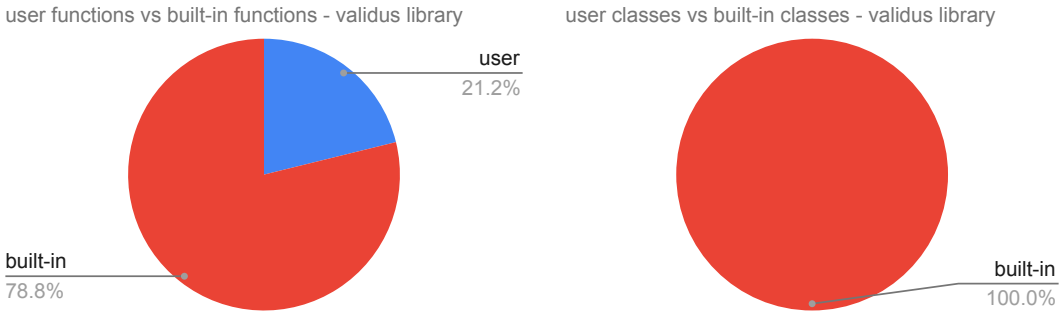
Επίσης, στα Γραφήματα C.1, C.2, C.3 φαίνονται σχηματικά (πίτα) τα ίδια με παραπάνω ποσοστά για τις μεγαλύτερες βιβλιοθήκες validus, pybite, και set-algebra αντίστοιχα.

Πίνακας C.2: **Περιτυλιγμένα στοιχεία χρήστη έναντι ενσωματωμένων ανά βιβλιοθήκη**.

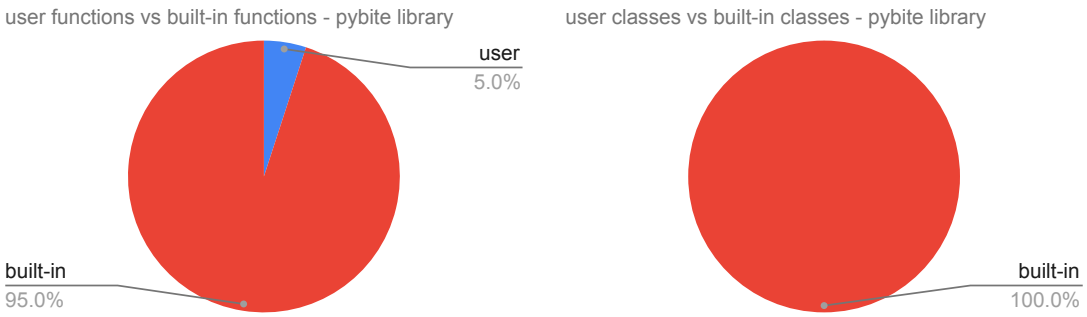
library	module	total functions	user made func	built-in func	user made func (%)	built-in func (%)
validus	validus.utils	42	1	41	2.38%	97.62%
	validus.phones	42	1	41	2.38%	97.62%
	validus.isbn	47	6	41	12.77%	87.23%
	validus.validators	88	47	41	53.41%	46.59%
	validus.__init__	41	0	41	0.00%	100.00%
	TOTAL	260	55	205	21.15%	78.85%
	module	total classes	user made classes	built-in classes	user made classes (%)	built-in classes (%)
	validus.utils	65	0	65	0.00%	100.00%
	validus.phones	65	0	65	0.00%	100.00%
	validus.isbn	65	0	65	0.00%	100.00%
	validus.validators	65	0	65	0.00%	100.00%
	validus.__init__	65	0	65	0.00%	100.00%
	TOTAL	325	0	325	0.00%	100.00%
library	module	total functions	user made func	built-in func	user made func (%)	built-in func (%)
bay-adder	bay_adder	42	1	41	2.38%	97.62%
	TOTAL	42	1	41	2.38%	97.62%
	module	total classes	user made classes	built-in classes	user made classes (%)	built-in classes (%)
	bay_adder	65	0	65	0.00%	100.00%
	TOTAL	65	0	65	0.00%	100.00%
library	module	total functions	user made func	built-in func	user made func (%)	built-in func (%)
binary_search	binary_search.__init__	43	2	41	4.65%	95.35%
	TOTAL	43	2	41	4.65%	95.35%
	module	total classes	user made classes	built-in classes	user made classes (%)	built-in classes (%)
	binary_search.__init__	66	0	66	0.00%	100.00%
	TOTAL	66	0	66	0.00%	100.00%
library	module	total functions	user made func	built-in func	user made func (%)	built-in func (%)
bit_array	bit_array.bit_array	41	0	41	0.00%	100.00%
	bit_array.__init__	41	0	41	0.00%	100.00%
	TOTAL	82	0	82	0.00%	100.00%
	module	total classes	user made classes	built-in classes	user made classes (%)	built-in classes (%)
	bit_array.bit_array	66	1	65	1.52%	98.48%
	bit_array.__init__	65	0	65	0.00%	100.00%
	TOTAL	131	1	130	0.76%	99.24%
library	module	total functions	user made func	built-in func	user made func (%)	built-in func (%)
split	split	47	6	41	12.77%	87.23%
	TOTAL	47	6	41	12.77%	87.23%
	module	total classes	user made classes	built-in classes	user made classes (%)	built-in classes (%)
	split	68	1	67	1.47%	98.53%
	TOTAL	68	1	67	1.47%	98.53%
library	module	total functions	user made func	built-in func	user made func (%)	built-in func (%)
first	first	42	1	41	2.38%	97.62%
	TOTAL	42	1	41	2.38%	97.62%
	module	total classes	user made classes	built-in classes	user made classes (%)	built-in classes (%)
	first	65	0	65	0.00%	100.00%
	TOTAL	65	0	65	0.00%	100.00%
library	module	total functions	user made func	built-in func	user made func (%)	built-in func (%)
first_dup	testing	42	1	41	2.38%	97.62%
	TOTAL	42	1	41	2.38%	97.62%
	module	total classes	user made classes	built-in classes	user made classes (%)	built-in classes (%)
	testing	65	0	65	0.00%	100.00%
	TOTAL	65	0	65	0.00%	100.00%
library	module	total functions	user made func	built-in func	user made func (%)	built-in func (%)
left-pad	left_pad	42	1	41	2.38%	97.62%
	TOTAL	42	1	41	2.38%	97.62%

	module	total classes	user made classes	built-in classes	user made classes (%)	built-in classes (%)
	left_pad	65	0	65	0.00%	100.00%
	TOTAL	65	0	65	0.00%	100.00%
library	module	total functions	user made func	built-in func	user made func (%)	built-in func (%)
max-ordered-diff	max_diff	42	1	41	2.38%	97.62%
	TOTAL	42	1	41	2.38%	97.62%
	module	total classes	user made classes	built-in classes	user made classes (%)	built-in classes (%)
	max_diff	65	0	65	0.00%	100.00%
	TOTAL	65	0	65	0.00%	100.00%
library	module	total functions	user made func	built-in func	user made func (%)	built-in func (%)
pad-on-left	pad_on_left.__init__	41	0	41	0.00%	100.00%
	pad_on_left.left_pad	42	1	41	2.38%	97.62%
	TOTAL	83	1	82	1.20%	98.80%
	module	total classes	user made classes	built-in classes	user made classes (%)	built-in classes (%)
	pad_on_left.__init__	65	0	65	0.00%	100.00%
	pad_on_left.left_pad	65	0	65	0.00%	100.00%
	TOTAL	130	0	130	0.00%	100.00%
library	module	total functions	user made func	built-in func	user made func (%)	built-in func (%)
pyfancy	pyfancy.pyfancy	42	1	41	2.38%	97.62%
	pyfancy.__init__	41	0	41	0.00%	100.00%
	pyfancy.demo	41	0	41	0.00%	100.00%
	TOTAL	124	1	123	0.81%	99.19%
	module	total classes	user made classes	built-in classes	user made classes (%)	built-in classes (%)
	pyfancy.pyfancy	66	1	65	1.52%	98.48%
	pyfancy.__init__	65	0	65	0.00%	100.00%
	pyfancy.demo	65	0	65	0.00%	100.00%
	TOTAL	196	1	195	0.51%	99.49%
library	module	total functions	user made func	built-in func	user made func (%)	built-in func (%)
value-lookup	value_lookup.value_lookup	42	1	41	2.38%	97.62%
	value_lookup.__init__	41	0	41	0.00%	100.00%
	TOTAL	83	1	82	1.20%	98.80%
	module	total classes	user made classes	built-in classes	user made classes (%)	built-in classes (%)
	value_lookup.value_lookup	65	0	65	0.00%	100.00%
	value_lookup.__init__	65	0	65	0.00%	100.00%
	TOTAL	130	0	130	0.00%	100.00%
library	module	total functions	user made func	built-in func	user made func (%)	built-in func (%)
fibonacci_index	fibonacci_index.__init__	41	0	41	0.00%	100.00%
	fibonacci_index.fibonacci_index	41	1	40	2.44%	97.56%
	TOTAL	82	1	81	1.22%	98.78%
	module	total classes	user made classes	built-in classes	user made classes (%)	built-in classes (%)
	fibonacci_index.__init__	65	0	65	0.00%	100.00%
	fibonacci_index.fibonacci_index	65	0	65	0.00%	100.00%
	TOTAL	130	0	130	0.00%	100.00%
library	module	total functions	user made func	built-in func	user made func (%)	built-in func (%)
ordered-set-stubs	ordered_set.__init__	42	1	41	2.38%	97.62%
	TOTAL	42	1	41	2.38%	97.62%
	module	total classes	user made classes	built-in classes	user made classes (%)	built-in classes (%)
	ordered_set.__init__	66	1	65	1.52%	98.48%
	TOTAL	66	1	65	1.52%	98.48%
library	module	total functions	user made func	built-in func	user made func (%)	built-in func (%)
tuple-flatten	tuple_flatten.__init__	42	1	41	2.38%	97.62%
	TOTAL	42	1	41	2.38%	97.62%
	module	total classes	user made classes	built-in classes	user made classes (%)	built-in classes (%)
	tuple_flatten.__init__	65	0	65	0.00%	100.00%
	TOTAL	65	0	65	0.00%	100.00%
library	module	total functions	user made func	built-in func	user made func (%)	built-in func (%)
flattenxyz	flatten_xyz.exception	41	0	41	0.00%	100.00%
	flatten_xyz.flatten	41	0	41	0.00%	100.00%
	flatten_xyz.__init__	41	0	41	0.00%	100.00%
	TOTAL	123	0	123	0.00%	100.00%
	module	total classes	user made classes	built-in classes	user made classes (%)	built-in classes (%)
	flatten_xyz.exception	66	1	65	1.52%	98.48%
	flatten_xyz.flatten	66	1	65	1.52%	98.48%
	flatten_xyz.__init__	65	0	65	0.00%	100.00%
	TOTAL	197	2	195	1.02%	98.98%
library	module	total functions	user made func	built-in func	user made func (%)	built-in func (%)
json-flatten	json_flatten	44	3	41	6.82%	93.18%
	TOTAL	44	3	41	6.82%	93.18%
	module	total classes	user made classes	built-in classes	user made classes (%)	built-in classes (%)
	json_flatten	65	0	65	0.00%	100.00%
	TOTAL	65	0	65	0.00%	100.00%

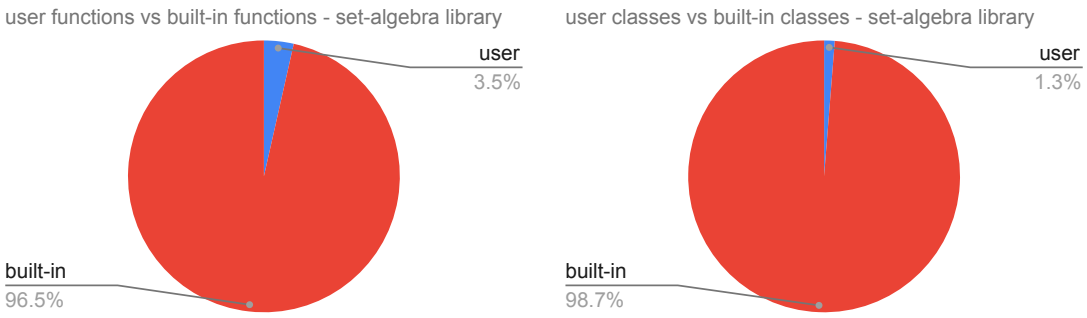
library	module	total functions	user made func	built-in func	user made func (%)	built-in func (%)
range-regex	range_regex.range_regex	47	6	41	12.77%	87.23%
	range_regex.__init__	41	0	41	0.00%	100.00%
	TOTAL	88	6	82	6.82%	93.18%
	module	total classes	user made classes	built-in classes	user made classes (%)	built-in classes (%)
	range_regex.range_regex	65	0	65	0.00%	100.00%
	range_regex.__init__	65	0	65	0.00%	100.00%
	TOTAL	130	0	130	0.00%	100.00%
library	module	total functions	user made func	built-in func	user made func (%)	built-in func (%)
range-digit	range_digit.__init__	45	4	41	8.89%	91.11%
	TOTAL	45	4	41	8.89%	91.11%
	module	total classes	user made classes	built-in classes	user made classes (%)	built-in classes (%)
	range_digit.__init__	66	1	65	1.52%	98.48%
	TOTAL	66	1	65	1.52%	98.48%
library	module	total functions	user made func	built-in func	user made func (%)	built-in func (%)
pybite	pybite.iterate	44	3	41	6.82%	93.18%
	pybite.file._open	42	1	41	2.38%	97.62%
	pybite.file.iterate	42	1	41	2.38%	97.62%
	pybite.file.chunk	49	8	41	16.33%	83.67%
	pybite.file.__init__	41	0	41	0.00%	100.00%
	pybite.__init__	41	0	41	0.00%	100.00%
	TOTAL	259	13	246	5.02%	94.98%
	module	total classes	user made classes	built-in classes	user made classes (%)	built-in classes (%)
	pybite.iterate	65	0	65	0.00%	100.00%
	pybite.file._open	65	0	65	0.00%	100.00%
	pybite.file.iterate	65	0	65	0.00%	100.00%
	pybite.file.chunk	65	0	65	0.00%	100.00%
	pybite.file.__init__	65	0	65	0.00%	100.00%
	pybite.__init__	65	0	65	0.00%	100.00%
	TOTAL	390	0	390	0.00%	100.00%
library	module	total functions	user made func	built-in func	user made func (%)	built-in func (%)
chunky	chunky.__init__	42	1	41	2.38%	97.62%
	TOTAL	42	1	41	2.38%	97.62%
	module	total classes	user made classes	built-in classes	user made classes (%)	built-in classes (%)
	chunky.__init__	66	1	65	1.52%	98.48%
	TOTAL	66	1	65	1.52%	98.48%
library	module	total functions	user made func	built-in func	user made func (%)	built-in func (%)
length	length.__init__	41	0	41	0.00%	100.00%
	length.length	43	2	41	4.65%	95.35%
	TOTAL	84	2	82	2.38%	97.62%
	module	total classes	user made classes	built-in classes	user made classes (%)	built-in classes (%)
	length.__init__	65	0	65	0.00%	100.00%
	length.length	65	0	65	0.00%	100.00%
	TOTAL	130	0	130	0.00%	100.00%
library	module	total functions	user made func	built-in func	user made func (%)	built-in func (%)
set-algebra	set_algebra.infinity	42	1	41	2.38%	97.62%
	set_algebra.parser	44	3	41	6.82%	93.18%
	set_algebra.endpoint	42	1	41	2.38%	97.62%
	set_algebra.interval	43	2	41	4.65%	95.35%
	set_algebra.set_	43	2	41	4.65%	95.35%
	set_algebra.__init__	41	0	41	0.00%	100.00%
	TOTAL	255	9	246	3.53%	96.47%
	module	total classes	user made classes	built-in classes	user made classes (%)	built-in classes (%)
	set_algebra.infinity	67	2	65	2.99%	97.01%
	set_algebra.parser	66	0	66	0.00%	100.00%
	set_algebra.endpoint	66	1	65	1.52%	98.48%
	set_algebra.interval	67	1	66	1.49%	98.51%
	set_algebra.set_	67	1	66	1.49%	98.51%
	set_algebra.__init__	65	0	65	0.00%	100.00%
	TOTAL	398	5	393	1.26%	98.74%
library	module	total functions	user made func	built-in func	user made func (%)	built-in func (%)
boxing	boxing.boxing	43	2	41	4.65%	95.35%
	boxing.__init__	41	0	41	0.00%	100.00%
	TOTAL	84	2	82	2.38%	97.62%
	module	total classes	user made classes	built-in classes	user made classes (%)	built-in classes (%)
	boxing.boxing	65	0	65	0.00%	100.00%
	boxing.__init__	65	0	65	0.00%	100.00%
	TOTAL	130	0	130	0.00%	100.00%
library	module	total functions	user made func	built-in func	user made func (%)	built-in func (%)
right-triangle	right_triangle	47	7	40	14.89%	85.11%
	TOTAL	47	7	40	14.89%	85.11%
	module	total classes	user made classes	built-in classes	user made classes (%)	built-in classes (%)
	right_triangle	66	1	65	1.52%	98.48%
	TOTAL	66	1	65	1.52%	98.48%



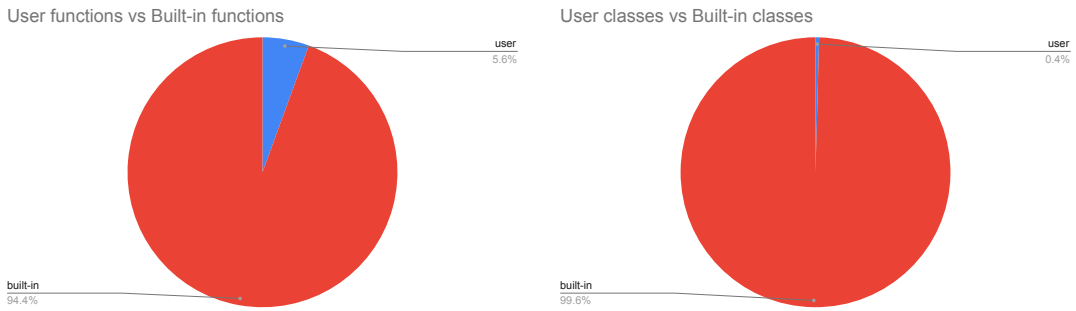
Γράφημα C.1: Περιτυλιγμένα στοιχεία χρήστη έναντι ενσωματωμένων στην Validus.



Γράφημα C.2: Περιτυλιγμένα στοιχεία χρήστη έναντι ενσωματωμένων στην pybite.



Γράφημα C.3: Περιτυλιγμένα στοιχεία χρήστη έναντι ενσωματωμένων στην set-algebra.



Γράφημα C.4: Συνολικά περιτυλιγμένα στοιχεία χρήστη έναντι ενσωματωμένων.

C.3 Προσδιορισμός αναλυθέντων βιβλιοθηκών

Σε αυτήν την ενότητα παρατίθεται ο σκοπός που επιτελεί κάθε βιβλιοθήκη, όπως ακριβώς προσδιορίζεται στο GitHub (προσδιορισμός στην αγγλική γλώσσα).

- validus: It is a simple string validation library. It takes a string and checks the type of it. e.g. email, url, json, etc. It usually uses regular expressions to succeed it.
- bay-adder: It just takes two numbers and returns their sum.
- binary_search: Binary Search class to traverse an ordered list and effectively populate the arrays with valid content. It takes an ordered list and applies the Binary Search algorithm to search for an item in the list. It returns this item with its corresponding index.
- bit_array: As the name indicates, it is an array whose elements are binary numbers, that is only numbers of 0 and 1, and acts as a list. You can pick up elements from there, you can initialize it with a list, you can access parts of the array with slice operations, etc.
- split: It is a library with functions that split and partition sequences.
- first: It is a package (library) with a simple function that returns the first true value from an iterable, or `None` if there is none. You can also supply a key function that is used to judge the truth value of the element or a default value if `None` doesn't fit your use case.
- first_dup: It is just a function that takes an array and checks if this array has duplicate elements, that is two or more elements with the same value. If it does, it stops searching and prints a message. It does not print this message if all its elements are unique.
- left-pad: As it was described in the Examples chapter (§5.1), it is a library (single module) that pads a string to the left with any number of characters. This number of characters is given by the user as an argument to the function `left_pad`.
- max-ordered-diff: It is a function that returns the largest ordered difference (between a value and another value with a smaller index) in the array.
- pad-on-left: It is a package (library) with a function called `left_pad`. This function returns a string, which will be padded on the left with characters if necessary. If the input string is longer than the specified length, it will be returned unchanged.

- pyfancy: It is a simple Python library that provides a mechanism for easily styling text in some terminal environments. Text is styled by chaining together methods that add escape codes for color modifiers to the text.

- value-lookup: It is a library that searches value inside an array. For example, if you use the optional `complete_analysis` parameter with the value of `True`, when you call the `vlookup` function of this library to search a value from a list, you will get more information about the searched value. Otherwise, if you don't use this parameter, you will just get a boolean of `True` or `False`, whether it is found or not, respectively.

- fibonacci_index: It is a library that computes the nth sum of the Fibonacci sequence after passing by the first two numbers which are the 0 and the 1. So the n starts with the value of 0 at the 3rd number of the Fibonacci sequence and it will be $n = k - 2$, with k the real index of the Fibonacci sequence.

- ordered-set-stubs: It is a library that returns stubs with type annotations for `ordered-set` Python library

- tuple-flatten: It is a library that keeps a single function that iterates through tuples and adds their values. Example: `(1,2)+(2,3)=(3,5)`. This function called `flatten(*argv)` takes an arbitrary amount of tuples and returns the sum of of each indices' value. It is not necessary for all tuples to be equally sized, as long as each tuple in the arguments is either shorter or equally sized as its predecessor.

- flatten-xyz: It is a library that flattens arrays projects.
For example, `[1, [2, [3, [4, 5]]]]` will be `[1, 2, 3, 4, 5]`.

- json-flatten: It is a library that contains functions for flattening a JSON object to a single dictionary of pairs, and unflattening that dictionary back to a JSON object.

- range-regex: This library generates regular expressions in numeric range. The user can give a range as arguments in a function and this function will generate a regular expression for this range.

- range-digit: It is library of ranges for rounding decimal digits, which have low and sup.

- pybite: It is a library that helps in chunk by chunk iteration. It creates an iterator of chunks from an iterable, it creates an iterator of file lines, it handles file's chunks, etc.

- chunky: It is a `Python` library that handles reading and writing of text files in chunks. It automatically creates files when the specified chunk size is reached. It provides a familiar interface of io.TextIOBase for easy integration with existing python IO facilities.

- length: It is a simple package (library) which is used to find the number of digits in the given number.

- set-algebra: This package provides classes representing math concepts, like `Infinity`, `Endpoint`, `Interval` and `Uncountable Infinite Set`.

- boxing: This package helps the user draw stylish boxes on the terminal with ease.

- right-triangle: This is a package that can be used to do calculations with right-angled triangles.