ΟΙΚΟΝΟΜΙΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΜΕΤΑΠΤΥΧΙΑΚΟ ΔΙΠΛΩΜΑ ΕΙΔΙΚΕΥΣΗΣ (MSc)
στα ΠΛΗΡΟΦΟΡΙΑΚΑ ΣΥΣΤΗΜΑΤΑ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

«ΑΝΑΠΤΥΞΗ ΜΕΘΟΔΟΥ ΕΥΡΕΣΗΣ ΛΟΓΙΚΩΝ ΑΔΥΝΑΜΙΩΝ ΣΤΙΣ ΔΟΚΙΜΕΣ ΔΙΕΙΣΔΥΣΗΣ»

ΣΤΕΡΓΙΟΠΟΥΛΟΣ ΓΕΩΡΓΙΟΣ

Μ3090004

ΑΘΗΝΑ, ΟΚΤΩΒΡΙΟΣ 2010

ΜΕΤΑΠΤΥΧΙΑΚΟ ΔΙΠΛΩΜΑ ΕΙΔΙΚΕΥΣΗΣ (MSc)
στα ΠΛΗΡΟΦΟΡΙΑΚΑ ΣΥΣΤΗΜΑΤΑ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

«ΑΝΑΠΤΥΞΗ ΜΕΘΟΔΟΥ ΕΥΡΕΣΗΣ ΛΟΓΙΚΩΝ ΑΔΥΝΑΜΙΩΝ ΣΤΙΣ ΔΟΚΙΜΕΣ ΔΙΕΙΣΔΥΣΗΣ»

ΣΤΕΡΓΙΟΠΟΥΛΟΣ ΓΕΩΡΓΙΟΣ

Μ3090004

Επιβλέπων Καθηγητής: ΔΗΜΗΤΡΙΟΣ ΓΚΡΙΤΖΑΛΗΣ
Εξωτερικός Κριτής: ΒΑΣΙΛΕΙΟΣ ΤΣΟΥΜΑΣ

ΟΙΚΟΝΟΜΙΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΑΘΗΝΑ, ΟΚΤΩΒΡΙΟΣ 2010

ΠΕΡΙΕΧΟΜΕΝΑ

ΕΥΡΕΤΗΡΙΟ ΠΙΝΑΚΩΝ 7
ΕΥΡΕΤΗΡΙΟ ΕΙΚΟΝΩΝ 8
ΕΥΡΕΤΗΡΙΟ ΣΧΗΜΑΤΩΝ 9
ΕΥΧΑΡΙΣΤΙΕΣ 10
ΠΕΡΙΛΗΨΗ 11
ABSTRACT 13
ΕΙΣΑΓΩΓΗ 14
1. ΓΕΝΙΚΕΣ ΚΑΤΗΓΟΡΙΕΣ ΕΛΕΓΧΟΥ 15
1.1 ΔΟΚΙΜΕΣ ΕΝΑΝΤΙΑ ΣΤΟ ΠΕΡΙΒΑΛΛΟΝ ΛΕΙΤΟΥΡΓΙΑΣ 16
1.2 ΔΟΚΙΜΕΣ ΕΙΣΑΓΩΓΗΣ ΠΛΗΡΟΦΟΡΙΩΝ ΣΤΗΝ ΕΦΑΡΜΟΓΗ 16
1.3 ΔΟΚΙΜΕΣ ΕΝΑΝΤΙΑ ΣΤΑ ΔΕΔΟΜΕΝΑ ΚΑΙ ΣΤΗ ΛΟΓΙΚΗ ΤΗΣ ΕΦΑΡΜΟΓΗΣ 17
2. BLACK-BOX TESTING 18
2.1 ΔΟΚΙΜΕΣ ΔΕΔΟΜΕΝΩΝ ΕΙΣΟΔΟΥ 19
2.1.1 ΔΟΚΙΜΈΣ ΔΥΝΑΤΏΝ ΣΥΝΔΥΑΣΜΏΝ (ALL-PAIRS TESTING) 19
2.1.2 ΔΟΚΙΜΈΣ ΑΣΑΦΕΙΑΣ (FUZZING TESTS) 20
2.1.3 ΔΟΚΙΜΈΣ ΑΚΡΑΙΩΝ ΤΙΜΏΝ ΕΙΣΌΔΟΥ 23
3. WHITE-BOX TESTING 24
3.1 ΕΛΕΓΧΟΙ ΜΕ ΕΙΣΑΓΩΓΗ ΔΕΔΟΜΕΝΩΝ (INPUT APPLICATION TESTING) 25
3.2 ΑΝΑΛΥΣΗ ΠΗΓΑΙΟΥ ΚΩΔΙΚΑ (CODE AUDIT) 25
3.2.1 ΛΕΙΤΟΥΡΓΙΚΟΙ ΕΛΕΓΧΟΙ (FUNCTION / UNIT TESTING) 25
3.2.2 ΕΛΕΓΧΟΙ ΣΧΕΔΙΑΣΜΟΥ (DESIGN TESTING) 25
3.2.3 ΕΛΕΓΧΟΙ ΤΗΣ ΡΟΗΣ ΕΚΤΕΛΕΣΗΣ (CONTROL FLOW TESTING) 26
3.2.4 ΕΛΕΓΧΟΙ ΡΟΗΣ ΔΕΔΟΜΕΝΩΝ (DATA FLOW TESTING) 26
3.3 BINARY EXECUTABLE FILE TESTING 27
3.3.1 PE HEADER TESTING 27
3.3.2 MACHINE-LEVEL TESTING 28
3.4 ΕΛΕΓΧΟΙ ΛΟΓΙΚΗΣ (APPLICATION LOGIC TESTS) 28
3.4.1 ΠΙΝΑΚΕΣ ΑΠΟΦΑΣΗΣ 29
3.4.2 ΠΙΝΑΚΕΣ ΜΕΤΑΒΑΣΗΣ ΚΑΤΑΣΤΑΣΕΩΝ 29
4. GRAY-BOX TESTING 31
5. ΜΕΘΟΔΟΙ ΚΑΙ ΕΡΓΑΛΕΙΑ ΣΤΙΣ ΔΟΚΙΜΕΣ ΔΙΕΙΣΔΥΣΗΣ 32
5.1 ΣΤΑΤΙΚΗ ΑΝΑΛΥΣΗ 33
5.2 ΜΕΘΟΔΟΙ ΣΤΑΤΙΚΗΣ ΑΝΑΛΥΣΗΣ 33
5.2.1 ΔΙΑΣΠΟΡΑ ΜΟΛΥΣΜΕΝΩΝ ΑΝΤΙΚΕΙΜΕΝΩΝ (TAINTED OBJECT PROPAGATION) 34
5.3 ΔΥΝΑΜΙΚΗ ΑΝΑΛΥΣΗ 35

5.4 ΜΕΘΟΔΟΙ ΔΥΝΑΜΙΚΗΣ ΑΝΑΛΥΣΗΣ 35

6. ΤΕΛΕΥΤΑΙΕΣ ΕΞΕΛΙΞΕΙΣ ΣΤΟ ΧΩΡΟ ΤΩΝ ΔΟΚΙΜΩΝ ΔΙΕΙΣΔΥΣΗΣ ΚΑΙ ΕΥΡΕΣΗΣ ΑΔΥΝΑΜΙΩΝ (STATE OF THE ART REVIEW) 37

6.1 ΣΤΑΤΙΚΟΣ ΕΝΤΟΠΙΣΜΟΣ ΑΔΥΝΑΜΙΩΝ ΣΕ X86 ΕΚΤΕΛΕΣΙΜΑ ΑΡΧΕΙΑ 37

6.2 ΔΟΚΙΜΕΣ ΑΣΑΦΕΙΑΣ ΓΙΑ ΕΛΕΓΧΟ ΑΣΦΑΛΕΙΑΣ ΚΑΙ ΠΟΙΟΤΗΤΑΣ ΛΟΓΙΣΜΙΚΟΥ (Takanen, DeMott, & Miller, 2008) 40

6.3 ΔΟΚΙΜΕΣ ΔΙΕΙΣΔΥΣΗΣ ΜΕ ΒΕΛΤΙΩΜΕΝΗ ΑΝΑΓΝΩΡΙΣΗ ΔΙΑΝΥΣΜΑΤΩΝ ΕΙΣΟΔΟΥ (William G.J. Halfond, Shauvik Roy Choudhary, and Alessandro Orso) 40

6.3.1 ΣΤΑΔΙΟ ΣΥΛΛΟΓΗΣ ΠΛΗΡΟΦΟΡΙΩΝ 41

6.3.2 ΦΑΣΗ ΔΗΜΙΟΥΡΓΙΑΣ ΔΕΔΟΜΕΝΩΝ 42

6.3.3 SDAPT 43

6.4 ΕΝΤΟΠΙΣΜΟΣ ΕΠΙΘΕΣΕΩΝ ΠΟΥ ΑΞΙΟΠΟΙΟΥΝ ΤΑ ΛΟΓΙΚΑ ΛΑΘΗ ΤΩΝ ΕΦΑΡΜΟΓΩΝ (Jingyu Zhou αnd Giovanni Vigna) 43

6.4.1 ΔΥΑΔΙΚΗ ΕΠΑΝΕΓΓΡΑΦΗ (BINARY REWRITING) 44

6.4.2 ΓΛΩΣΣΑ ΠΡΟΣΔΙΟΡΙΣΜΟΥ ΕΛΕΓΧΩΝ 45

6.5 ΕΝΑ ΠΡΩΤΟ ΒΗΜΑ ΠΡΟΣ ΤΟΝ ΑΥΤΟΜΑΤΟ ΕΝΤΟΠΙΣΜΟ ΑΔΥΝΑΜΙΩΝ ΥΠΕΡΧΕΙΛΙΣΗΣ ΜΝΗΜΗΣ (david wagner jeffrey s. foster eric a. brewer alexander aiken, university of california, berkeley) 46

6.6 ΔΥΝΑΜΙΚΗ ΑΝΑΛΥΣΗ ΜΟΛΥΣΜΕΝΩΝ ΔΕΔΟΜΕΝΩΝ ΓΙΑ ΤΟΝ ΑΥΤΟΜΑΤΟ ΕΝΤΟΠΙΣΜΟ, ΑΝΑΛΥΣΗ ΚΑΙ ΚΑΤΑΓΡΑΦΗ ΑΔΥΝΑΜΙΩΝ ΣΕ ΕΜΠΟΡΙΚΟ ΛΟΓΙΣΜΙΚΟ (James Newsome, Dawn Song, July 2005, School of Computer Science, Carnegie Mellon University. Pittsburgh) 47

6.7 ITS4: ΕΝΑΣ ΣΤΑΤΙΚΟΣ ΣΑΡΩΤΗΣ ΑΔΥΝΑΜΙΩΝ ΓΙΑ ΠΗΓΑΙΟ ΚΩΔΙΚΑ ΤΩΝ ΓΛΩΣΣΩΝ C ΚΑΙ C++ Code (John Viega, J.T. Bloch, Yoshi Kohno, Gary McGraw. Reliable Software Technologies, Dulles, Virginia) 48

6.8 ΜΙΑ ΕΥΡΙΣΤΙΚΗ ΠΡΟΣΕΓΓΙΣΗ ΓΙΑ ΕΝΤΟΠΙΣΜΟ ΑΔΥΝΑΜΙΩΝ ΕΓΧΥΣΗΣ ΕΝΤΟΛΩΝ SQL ΣΕ ΔΙΑΔΙΚΤΥΑΚΕΣ ΕΦΑΡΜΟΓΕΣ (Angelo Ciampa, Corrado Aaron Visaggio, Massimiliano Di Penta, Dept. of Engineering-RCOST, Univ. Of Sannio, Italy) 48

6.9 ΜΕΘΟΔΟΙ ΓΙΑ ΤΗΝ ΑΠΟΤΡΕΨΗ, ΤΟΝ ΕΝΤΟΠΙΣΜΟ ΚΑΙ ΤΗΝ ΑΠΟΜΑΚΡΥΝΣΗ ΑΔΥΝΑΜΙΩΝ ΛΟΓΙΣΜΙΚΟΥ (Jay-Evan J. Tevis, John A. Hamilton, Jr. Department of Computer Science and Software Engineering, Auburn University, Auburn, Alabama) 50

6.10 ΣΤΑΤΙΚΗ ΑΝΑΛΥΣΗ ΑΝΩΜΑΛΙΩΝ ΚΑΙ ΑΔΥΝΑΜΙΩΝ ΑΣΦΑΛΕΙΑΣ ΣΕ ΕΚΤΕΛΕΣΙΜΑ ΑΡΧΕΙΑ (Jay-Evan J. Tevis, John A. Hamilton, Jr. Department of Computer Science, Western Illinois University, Macomb, Illinois) 51

6.11 ΡΙΧΥ: ΕΝΑ ΕΡΓΑΛΕΙΟ ΣΤΑΤΙΚΗΣ ΑΝΑΛΥΣΗΣ ΓΙΑ ΤΟΝ ΕΝΤΟΠΙΣΜΟ ΑΔΥΝΑΜΙΩΝ ΣΕ ΔΙΑΔΙΚΤΥΑΚΕΣ ΕΦΑΡΜΟΓΕΣ (Nenad Jovanovic, Christopher Kruegel, and Engin Kirda, Technical University of Vienna, Secure Systems Lab) 52

6.12 ΕΞΕΡΕΥΝΩΝΤΑΣ ΤΟ BLACK-BOX: ΚΑΝΟΝΤΑΣ ΧΡΗΣΗ ΔΥΝΑΜΙΚΗΣ ΔΙΑΧΕΙΡΙΣΗΣ ΓΙΑ ΤΗΝ ΔΗΜΙΟΥΡΓΙΑ ΑΔΥΝΑΜΙΩΝ ΑΣΦΑΛΕΙΑΣ (Barton P. Miller, Mihai Christodorescu, Robert Iverson, Tevfik Kosar, Alexander Mirgorodskii, Florentina Popovici. Computer Sciences Department, University of Wisconsin, Madison) 53

6.13 ΒΕΛΤΙΩΣΗ ΔΗΜΙΟΥΡΓΙΑΣ ΔΟΚΙΜΩΝ ΒΑΣΕΙ ΣΕΝΑΡΙΩΝ ΚΑΝΟΝΤΑΣ ΧΡΗΣΗ ΑΥΤΟΜΑΤΟΥ ΠΡΟΣΔΙΟΡΙΣΜΟΥ ΔΙΕΠΑΦΩΝ (William G.J. Halfond and Alessandro Orso. College of Computing. Georgia Institute of Technology) 54

6.14 SWADDLER: ΜΙΑ ΠΡΟΣΕΓΓΙΣΗ ΓΙΑ ΤΟΝ ΕΝΤΟΠΙΣΜΟ ΠΑΡΑΒΙΑΣΕΩΝ ΤΩΝ ΠΕΡΙΟΡΙΣΜΩΝ ΚΑΤΑΣΤΑΣΗΣ ΣΕ ΔΙΑΔΙΚΤΥΑΚΕΣ ΕΦΑΡΜΟΓΕΣ ΚΑΝΟΝΤΑΣ ΧΡΗΣΗ ΑΝΩΜΑΛΙΩΝ (marco cova, davide balzarotti, viktoria felmetsger, and giovanni vigna. department of computer science, university of california santa barbara) 55

7. Η ΜΕΘΟΔΟΣ LogGIC 59

7.1 ΕΙΣΑΓΩΓΗ 59

7.2 ΛΟΓΙΚΑ ΛΑΘΗ ΚΑΙ Η ΜΕΘΟΔΟΣ ΤΗΣ ΔΙΠΛΩΜΑΤΙΚΗΣ 61

7.3 Η ΜΕΘΟΔΟΣ ΠΡΟΣΕΓΓΙΣΗΣ ΛΟΓΙΚΩΝ ΕΠΙΛΟΓΩΝ 63

7.4 ΜΕΘΟΔΟΣ ΑΝΑΛΥΣΗΣ ΔΙΑΝΥΣΜΑΤΩΝ ΕΙΣΟΔΟΥ 65

7.5 Η ΜΕΘΟΔΟΣ ΑΝΑΛΥΣΗΣ ΛΟΓΙΚΩΝ ΚΑΝΟΝΩΝ 66

7.6 ΕΡΓΑΛΕΙΑ ΥΛΟΠΟΙΗΣΗΣ ΤΗΣ ΜΕΘΟΔΟΥ 71

7.6.1 ΕΞΑΓΩΓΗ ΚΑΝΟΝΩΝ ΜΕ ΤΟ DAIKON 71

7.6.2 ΠΑΡΑΓΩΓΗ ΜΟΝΟΠΑΤΙΩΝ ΕΚΤΕΛΕΣΗΣ ΤΗΣ ΣΤΑΤΙΚΗΣ ΑΝΑΛΥΣΗΣ ΚΑΙ ΤΟ ΕΡΓΑΛΕΙΟ NASA JAVA PATHFINDER (JPF) 72

7.7 ΠΡΟΣΠΕΛΑΣΗ ΔΕΝΤΡΟΥ ΜΕΤΑΓΛΩΤΤΙΣΗΣ ΚΑΙ Ο JAVAC 74

7.8 ΑΣΑΦΗΣ ΛΟΓΙΚΗ 76

7.8.1 ΑΠΟΣΑΦΗΝΙΣΗ ΑΠΟΤΕΛΕΣΜΑΤΩΝ ΑΣΑΦΟΥΣ ΛΟΓΙΚΗΣ 76

7.8.2 ΑΣΑΦΗΣ ΛΟΓΙΚΗ ΚΑΙ ΕΡΓΑΛΕΙΟ LOGGIC 77

7.9 ΠΑΡΟΥΣΙΑΣΗ ΤΩΝ ΑΠΟΤΕΛΕΣΜΑΤΩΝ ΛΕΙΤΟΥΡΓΙΑΣ ΤΟΥ ΕΡΓΑΛΕΙΟΥ ΚΑΙ ΤΩΝ ΠΙΘΑΝΩΝ ΑΔΥΝΑΜΙΩΝ 83

8. ΠΕΡΙΟΡΙΣΜΟΙ ΛΕΙΤΟΥΡΓΙΑΣ ΚΑΙ ΜΕΛΛΟΝΤΙΚΗ ΕΡΕΥΝΑ 85

9. ΧΡΗΣΗ ΤΗΣ ΜΕΘΟΔΟΥ ΜΕ ΤΟ ΕΡΓΑΛΕΙΟ LogGIC ΚΑΙ ΕΞΑΓΩΓΗ ΣΥΜΠΕΡΑΣΜΑΤΩΝ 88

9.1 ΚΟΙΝΑ ΚΡΙΤΗΡΙΑ ΑΞΙΟΛΟΓΗΣΗΣ ΚΑΙ Η ΚΛΙΜΑΚΑ CCWAPPS 57

9.1 ΑΝΑΠΤΥΞΗ ΕΦΑΡΜΟΓΗΣ LOGICBOMB ΓΙΑ ΕΛΕΓΧΟ ΜΕ ΤΟ LogGIC 89

9.2 ΑΠΟΤΕΛΕΣΜΑΤΑ ΑΝΑΛΥΣΗΣ ΕΦΑΡΜΟΓΩΝ 90

10. ΠΑΡΑΡΤΗΜΑΤΑ 93

10.1 ΠΑΡΑΡΤΗΜΑ Α - ΠΡΟΤΥΠΟ ΑΡΧΕΙΟ SITE.PROPERTIES ΓΙΑ ΤΗΝ ΕΓΚΑΤΑΣΤΑΣΗ ΤΟΥ JPF 94

10.2 ΠΑΡΑΡΤΗΜΑ Β – ΠΡΟΤΥΠΟ ΑΡΧΕΙΟ ΠΑΡΑΜΕΤΡΟΠΟΙΗΣΗΣ ΓΙΑ ΕΛΕΓΧΟ ΕΦΑΡΜΟΓΗΣ ΜΕ ΤΟ JPF 95

10.3 ΠΑΡΑΡΤΗΜΑ Γ - ΠΑΡΑΜΕΤΡΟΠΟΙΗΣΗ ΛΕΙΤΟΥΡΓΙΑΣ ΤΟΥ DAIKON 96

10.4 ΠΑΡΑΡΤΗΜΑ Δ - ΠΑΡΑΜΕΤΡΟΠΟΙΗΣΗ ΛΕΙΤΟΥΡΓΙΑΣ ΚΑΙ ΕΓΚΑΤΑΣΤΑΣΗ ΤΟΥ JAVA PATHFINDER 97

11. ΒΙΒΛΙΟΓΡΑΦΙΑ 100

ΕΥΡΕΤΗΡΙΟ ΠΙΝΑΚΩΝ

Πίνακας 1: Πιθανοι συνδυασμοι σε συνολο πεντε μεταβλητων [1] 19
Πίνακας 2: Τα τεσσερα τεταρτημορια λογικων αποφασεων [24] 29
Πίνακας 3: Παραδειγμα σχεσεων γεγονοτων καταστασεων 30

ΕΥΡΕΤΗΡΙΟ ΕΙΚΟΝΩΝ

Εικόνα 1: Ροη πληροφοριασ σε δοκιμεσ black-box 18
Εικόνα 2: Η αρχιτεκτονική του εργαλείου εύρεσης αδυναμιών των Cova, Felmetsger, Banks και Vigna. 38
Εικόνα 4: Μερος του ορισμου κριτηριων τησ CCWAPSS [27] 57

ΕΥΡΕΤΗΡΙΟ ΣΧΗΜΑΤΩΝ

Figure 1: ΔΙΑΓΡΑΜΜΑ ΡΟΗΣ ΜΕΘΟΔΟΥ Loggic 62

ΕΥΧΑΡΙΣΤΙΕΣ

Κατ' αρχάς, θα ήθελα να ευχαριστήσω τον επιβλέποντά μου, καθηγητή κ. Δημήτρη Α. Γκρίτζαλη και τον συνεπιβλέποντά μου κ. Βασίλη Τσούμα για την βοήθεια και τις κατευθυντήριες συμβουλές τους κατά την διάρκεια των περασμένων μηνών. Οφείλω το θέμα και την πορεία της διπλωματικής μου εργασίας στον κ. Τσούμα, καθώς ήταν εκείνος ο οποίος κατέθεσε την ακαδημαϊκή πρόταση της παρούσας διπλωματικής και ο οποίος με εισήγαγε στον τομέα των Δοκιμών Διείσδυσης υπολογιστικών εφαρμογών, σε μια περίοδο που είχα περιορισμένες γνώσεις γύρω από τους σύγχρονους τομείς της ασφάλειας. Ήξερα μόνο ότι ήθελα να εργαστώ πάνω στα γενικότερα θέματα ασφαλείας λογισμικού, χωρίς κάτι συγκεκριμένο υπ' όψιν μου.

Ευχαριστώ επίσης θερμά τον εκπονητή λογισμικού της NASA, Peter Mehlitz για την άριστη και διαρκή του υποστήριξη όσων αφορά το εργαλείο Java PathFinder της NASA. Η βοήθειά του υπήρξε καθοριστική για την ποιότητα των αποτελεσμάτων του εργαλείου JPF κατά την εκπόνηση της εργασίας μου, γεγονός που επηρέασε άμεσα και την ποιότητα της παρούσας διπλωματικής. Το Java PathFinder, όντας ένα δύσχρηστο εργαλείο, θα απαιτούσε πολύ περισσότερο χρόνο προκειμένου να δουλέψει ως έπρεπε, χρόνος που κατάφερα να διαθέσω σε άλλα καίρια σημεία της διπλωματικής μου, καθαρά λόγω της άμεσης υποστήριξης που παρείχε ο κ. Mehlitz.

Τέλος, θα ήθελα να ευχαριστήσω την οικογένειά μου για την στήριξη και βοήθεια που παρείχαν καθ' όλη τη διάρκεια εκπόνησης της διπλωματικής μου εργασίας. Η βοήθεια όλων απεδείχθη καταλυτική για την επιτυχία της παρούσας διπλωματικής.

ΠΕΡΙΛΗΨΗ

Η παρούσα διπλωματική αποτελείται από δύο μέρη. Το πρώτο μέρος είναι μια θεωρητική ανάλυση του ερευνητικού χώρου των αδυναμιών ασφαλείας και των τεχνικών εντοπισμού τους, καθώς επίσης και μια παρουσίαση διάφορων σημαντικών ερευνών γύρω από τον εντοπισμό λογικών αδυναμιών και γενικότερων αδυναμιών στα διάφορα είδη εφαρμογών. Το δεύτερο μέρος είναι μία προσπάθεια ανάπτυξης ενός εργαλείου αυτόματου εντοπισμού λογικών αδυναμιών σε εφαρμογές με γραφική διεπαφή, με το όνομα LogGIC. Το εργαλείο αναπτύχθηκε στη γλώσσα προγραμματισμού Java. Η μέθοδος που αναπτύχθηκε δεν αποτελεί μια ακόμα υλοποίηση συνηθισμένων δοκιμών διείσδυσης που ελέγχουν τεχνικές αδυναμίες (όπως έγχυση εντολών SQL, υπερχείλιση πινάκων κ.τ.λ.) αλλά στοχεύει τα λάθη λογικής στον προγραμματισμό των λειτουργιών των εφαρμογών.

Με τον όρο *λογικό λάθος* ή *λογική αδυναμία* εννοούμε έναν συγκεκριμένο, διαθέσιμο και επιτρεπτό συνδυασμό δράσεων ενός χρήστη πάνω σε μία εφαρμογή, ο οποίος οδηγεί της λειτουργίες της σε ανεπιθύμητα αποτελέσματα. Οι λογικές αδυναμίες είναι αποτέλεσμα λανθασμένου προγραμματισμού της λογικής διαδικασίας μιας λειτουργίας –και, γενικότερα, ενός προγράμματος- από τον εκπονητή που ανέλαβε να την υλοποιήσει. Τέτοιες αδυναμίες είναι επιβλαβείς για την ασφάλεια του λογισμικού και συχνά μπορούν να χρησιμοποιηθούν από κακόβουλους χρήστες, προκειμένου αυτοί να αλλοιώσουν τη προβλεπόμενη λειτουργία μιας εφαρμογής ή προκειμένου να αποκτήσουν πρόσβαση σε πληροφορίες και λειτουργίες τις οποίες δεν θα έπρεπε κανονικά να μπορούν να προσπελάσουν. Ο εντοπισμός και η αξιολόγηση λογικών αδυναμιών μέσα σε μια εφαρμογή αποτελεί μια επίπονη και χρονοβόρα διαδικασία, καθώς ο αναλυτής της εκάστοτε εφαρμογής πρέπει να έχει μακροχρόνια εμπειρία και μεγάλο εύρος γνώσεων γύρω από τα θέματα Ασφάλειας των υπολογιστικών συστημάτων. Η παρούσα διπλωματική αποτελεί μια προσπάθεια αυτοματοποίησης της διαδικασίας εύρεσης λογικών αδυναμιών σε εφαρμογές με γραφική διεπαφή. Προτείνεται και υλοποιείται μια μέθοδος εύρεσης λογικών αδυναμιών η οποία συνδυάζει τρεις υπάρχουσες τεχνικές εντοπισμού αδυναμιών: Η μέθοδος συνδυάζει διαδικασίες δυναμικής και στατικής ανάλυσης των εφαρμογών υπό έλεγχο, με μια ευριστική μέθοδο εντοπισμού κρίσιμων σημείων στο κώδικα -από πλευράς ασφάλειας- στα οποία γίνεται λήψη αποφάσεων, καθώς και με μία μέθοδο εντοπισμού αδυναμιών που στηρίζεται στον έλεγχο των διανυσμάτων εισόδου δεδομένων στην εφαρμογή. Τα αποτελέσματα των τριών ανωτέρω διαδικασιών τροφοδοτούνται σε ένα σύστημα Ασαφούς Λογικής προκειμένου να προσδιοριστεί

το επίπεδο κρισιμότητας των πιθανών λογικών αδυναμιών που εντοπίστηκαν. Η πρώτη μέθοδος εύρεσης λογικών αδυναμιών η οποία στηρίζεται στην δυναμική και στατική ανάλυση, αποτελεί μια υλοποίηση που βασίστηκε στην έρευνα της ομάδας ασφάλειας του Πανεπιστημίου της Καλιφόρνια [2].

Το εργαλείο που αναπτύχθηκε στη παρούσα διπλωματική είναι σχεδιασμένο να παρέχει χρέη συμβούλου στους αναλυτές εφαρμογών. Εντοπίζει, αξιολογεί τα αποτελέσματα και, στη συνέχεια, εφιστά την προσοχή στα σημεία του πηγαίου κώδικα των εφαρμογών τα οποία ενδέχεται να κρύβουν επικίνδυνες λογικές αδυναμίες, παραθέτοντάς μια μαθηματική αξιολόγηση της επικινδυνότητας της εκάστοτε λογικής αδυναμίας. Η αξιολόγηση υπολογίζεται συνδυάζοντας τη *συνολικής Δριμύτητας* (*final severity*) μιας αδυναμίας (η Δριμύτητα υπολογίζεται με βάση τη θέση και τον ρόλο του κώδικα που κρύβει την αδυναμία μέσα στο πηγαίο κώδικα της εφαρμογής), με το μέγεθος της ίδιας της αδυναμίας (κατά πόσον σίγουρο είναι ότι υπάρχει αδυναμία και είναι προσπελάσιμη. Η μεταβλητή υπολογίζεται με βάση τα αποτελέσματα της ευριστικής μεθόδου ανάλυσης καθώς επίσης και με βάση τα αποτελέσματα της Δυναμικής και Στατικής ανάλυσης).

ABSTRACT

This Msc thesis consists of two parts. The first part is a thorough analysis on the field of security vulnerabilities along with a presentation of some interesting papers and research documents that were written during the last few years. The second part is an attempt to develop an automatic tool that can find logic vulnerabilities in graphical user interface applications. The tool is written in Java and was given the name LogGIC.

Logic vulnerabilities are born from erroneous programming during the process of developing an application. This kind of vulnerabilities are dangerous and possibly harmful when it comes to security issues, since they can be exploited by maligned users in order to gain access to sensitive information or in order to change the intended operation of an application. Thus, the necessity of finding an automatic method that can track down and expose logic vulnerabilities was born. Tracking down logic vulnerabilities in computer programs can be a long and tedious process. The security analyst has to delve each time into complex source code and keep track of the latest progress in security and programming in order to be efficient. This thesis is an attempt to develop a tool that can automatize the process of finding logic vulnerabilities in applications with graphical user interfaces. I developed a method to track down and find logic vulnerabilities, which combines three known methods used in vulnerability analysis. The method developed in this thesis combines Dynamic program analysis and Static analysis with a heuristic method which finds critical decision points in the source code and, also, with a method which tracks down vulnerabilities created by the input Vectors of an application. The results of all three methods are then fed to a Fuzzy Logic system in order to compute the criticality level of each possible vulnerability. The first method deployed in the LogGIC tool, which uses Dynamic and Static analysis, is based on the research in logic vulnerabilities done by the security team of the University of California[2].

The tool that was developed during the course of this thesis is designed to fulfill the role of an advisor to a security analyst. It can track down and evaluate a possible logic vulnerability and then warn the security analyst about it, by providing a mathematical estimation of the criticality level of each one of its results. This estimation is computed by combining the severity level of a finding (based on its place and role inside the source code) with a vulnerability level of exposure (which is based on the heuristics and the dynamic analysis results).

ΕΙΣΑΓΩΓΗ

Η παρούσα διπλωματική ασχολείται με το πρόβλημα της αυτόματης εύρεσης λογικών αδυναμιών με χρήση μεθόδων και εργαλείων. Προσφέρει μια εκτενή θεωρητική ανάλυση του χώρου της Ασφάλειας που αφορά τις Δοκιμές Διείσδυσης και τις μεθόδους εύρεσης αδυναμιών. Παρουσιάζει μια πληθώρα μεθόδων και τεχνικών που καλύπτουν το ευρύτερο φάρμα της εύρεσης αδυναμιών, τόσο στις υπολογιστικές εφαρμογές όσο και στο πηγαίο κώδικα των προγραμμάτων.

Στη συνέχεια, παρουσιάζονται κάποιες ενδιαφέρουσες εξελίξεις στο χώρο της Ασφάλειας και, ειδικότερα, στις τεχνικές εύρεσης αδυναμιών. Οι έρευνες που αναλύονται αποτελούν δημοσιευμένη δουλειά ακαδημαϊκών ομάδων ασφάλειας από πανεπιστήμια όλου του κόσμου. Κάποιες δημοσιεύσεις αναλύονται σε μεγαλύτερο βάθος, καθώς αποτελούν ενδιαφέρουσες έρευνες στον τομέα εύρεσης αδυναμιών κι επίσης, επειδή αποτέλεσαν πηγή έμπνευσης για την δουλειά που πραγματοποιήθηκε στα πλαίσια της παρούσας διπλωματικής.

Τέλος, παρουσιάζεται η μέθοδος που αναπτύχθηκε στα πλαίσια της παρούσας διπλωματικής και αναλύεται η διαδικασία ανάπτυξης του εργαλείου LogGIC, το οποίο αποτελεί μια μερική υλοποίηση της θεωρητικής μεθόδου εύρεσης λογικών αδυναμιών, καθώς και των σχετικών τεχνικών που αναπτύχθηκαν και ενσωματώθηκαν στη μέθοδο που προτείνουμε.

1. ΓΕΝΙΚΕΣ ΚΑΤΗΓΟΡΙΕΣ ΕΛΕΓΧΟΥ

Στο κεφάλαιο γίνεται μια γενική επισκόπηση του χώρου των δοκιμών διείσδυσης. Περιγράφεται τί είναι μια δοκιμή διείσδυσης καθώς και τα σημεία στα οποία εστιάζουν αυτές οι διαδικασίες δοκιμών.

Οι *Δοκιμές Διείσδυσης* (στα αγγλικά *Penetration Testing,* χρήση αυτού του όρου από εδώ και εφεξής) είναι το σύνολο μεθοδικών διαδικασιών ανάλυσης *λογισμικού* (*software)*, οι οποίες εστιάζουν στον εντοπισμό ευπαθειών και αδυναμιών που μπορούν να φανούν χρήσιμες σε κακόβουλους χρήστες [1]. Δίχως να αποτελεί πανάκεια, το Penetration Testing αποτελεί ένα βήμα για την ανάπτυξη λογισμικού που πληροί τις προϋποθέσεις ασφάλειας. Σκοπό του Penetration Testing δεν αποτελεί ο έλεγχος της λειτουργικότητας ενός λογισμικού. Αντίθετα, σκοπός του είναι ο έλεγχος της εφαρμογής για ύπαρξη ευπαθειών και αδυναμιών.

Ένας καθοριστικός παράγοντας επιτυχίας δοκιμής διείσδυσης είναι να καταφέρει να εκτρέψει την λειτουργία της εφαρμογής υπό έλεγχο από τον προβλεπόμενο στόχο της: Η εύρεση σημείων του κώδικα, σημεία εισόδων ή σημεία διαδικασιών στην εφαρμογή που μπορούν να αλλοιωθούν και η προσπάθεια της δοκιμής διείσδυσης να αλλοιώσει τα ευρήματα αυτά προκειμένου να παρατηρηθεί πώς θα αντιδράσει το λογισμικό κάτω από τις νέες συνθήκες, αποτελεί το βασικό τρόπο λειτουργίας των δοκιμών αυτών. Ο σκοπός κάθε δοκιμής διείσδυσης είναι να αποδειχθεί ότι η εφαρμογή υπό έλεγχο λειτουργεί αξιόπιστα κάτω από αναμενόμενες αλλά και μη αναμενόμενες συνθήκες λειτουργίας. Οπότε, το σημαντικότερο βήμα κατά τη διαδικασία ενός Penetration Test είναι ο εντοπισμός του *τι* μπορεί να αλλοιωθεί στην εφαρμογή υπό έλεγχο και *με ποιούς* τρόπους.

Το *περιβάλλον λειτουργίας* της εφαρμογής *(Application Environment)*, η *εισαγωγή πληροφοριών* από τους χρήστες (*user input*, χρήση αγγλικού όρου εφεξής) και η *εσωτερική λογική και δεδομένα* του προγράμματος αποτελούν τα βασικά μέρη κάθε εφαρμογής. Τυχόν αλλοίωση της λειτουργίας τους μπορεί να εμφανίσει προβλήματα ασφάλειας [1]. Το περιβάλλον λειτουργίας αποτελείται από τα αρχεία, τις εξωτερικές εφαρμογές, τους πόρους συστήματος και άλλους τοπικούς ή δικτυακούς πόρους που χρησιμοποιεί η εφαρμογή υπό έλεγχο. Η εισαγωγή πληροφοριών είναι ουσιαστικά όλα τα δεδομένα που προέρχονται από εξωτερικές και –πιθανώς αναξιόπιστες- οντότητες του περιβάλλοντος της εφαρμογής. Τα δεδομένα αυτά εισέρχονται στην εφαρμογή από τις εισόδους της και, στην συνέχεια, αναλύονται. Ως εσωτερικά δεδομένα και λογική της εφαρμογής ορίζονται οι αποθηκευμένες μεταβλητές του προγράμματος και τα *λογικά μονοπάτια εκτέλεσης (logical execution paths)* της εφαρμογής.

Οι αλλοιώσεις πληροφοριών και δεδομένων στο περιβάλλον λειτουργίας της εφαρμογής, στις διαδικασίες εισαγωγής πληροφοριών και στα εσωτερικά δεδομένα και λειτουργίες του προγράμματος, αποτελούν τις κύριες μεθόδους των κακόβουλων χρηστών για επίθεση σε μια εφαρμογή. Ακολουθεί αναλυτικότερη παρουσίαση των τριών αυτών γενικών κατηγοριών.

1.1 ΔΟΚΙΜΕΣ ΕΝΑΝΤΙΑ ΣΤΟ ΠΕΡΙΒΑΛΛΟΝ ΛΕΙΤΟΥΡΓΙΑΣ

Η λειτουργία του λογισμικού στηρίζεται σε ένα πλήθος από εκτελέσιμα αρχεία, *επεκτάσεις λειτουργικότητας (plugins*) και *Δυναμικά συνδεδεμένων βιβλιοθηκών (Dynamically Linked Libraries – DLL* εφ' έξης). Το λογισμικό χρησιμοποιεί, εγγράφει και ανακτά πληροφορίες από περιοχές του συστήματος όπως αυτές των *μεταβλητών περιβάλλοντος (environmental variables)* και των *καταχωρήσεων συστήματος (registry)*. Καθένας από τους προαναφερθέντες πόρους μπορεί να είναι η πηγή κάποιας αδυναμίας στην ασφάλεια των εφαρμογών υπό δοκιμή.

Δημοφιλείς τρόποι επίθεσης αποτελούν, μεταξύ άλλων, η αλλοίωση ή ακόμη και η αντικατάσταση των *δυναμικών βιβλιοθηκών* (*DLL*) μιας εφαρμογής, των εκτελέσιμων αρχείων, των κοινών περιοχών μνήμης που κάνει χρήση η εφαρμογή αλλά και των προσωρινών αρχείων της. Τέλος, κάποιες τεχνικές επίθεσης βασίζονται στην επιβάρυνση της λειτουργίας του συστήματος σε ακραία σημεία, προκειμένου να μελετηθεί η ανταπόκριση και η λειτουργία του σε σχέση με το περιβάλλον. Τέτοιες τεχνικές μπορεί να εστιάσουν στην επιβάρυνση του δικτύου, στην ακραία κατανάλωση μνήμης κ.α.

1.2 ΔΟΚΙΜΕΣ ΕΙΣΑΓΩΓΗΣ ΠΛΗΡΟΦΟΡΙΩΝ ΣΤΗΝ ΕΦΑΡΜΟΓΗ

Στο penetration testing, όλες οι πληροφορίες που εισέρχονται στην εφαρμογή από μη αξιόπιστες πηγές είναι άξιες ανάλυσης. Οι *διαδρομές (paths)* διαδικτυακών πρωτοκόλλων, οι *υποδοχές πληροφοριών* (*sockets*, χρήση αγγλικού όρου εφεξής), οι διάφορες *απομακρυσμένες επικλήσεις διαδικασιών (remote procedure calls –RPCs* εφεξής), κάποιες διαδικτυακές υπηρεσίες, τα αρχεία δεδομένων που χρησιμεύουν ως είσοδοι στην εφαρμογή, τα προσωρινά αρχεία που δημιουργούνται κατά τη λειτουργία της εφαρμογής όπως και τα διάφορα *αρχεία παραμετροποίησης (configuration files)* και αρχεία XML, όλα τα ανωτέρω μπορούν να υποστούν αλλαγή και να χρησιμοποιηθούν για καταχώρηση ανεπιθύμητων πληροφοριών και αλλοίωση

της λειτουργίας του λογισμικού. Επίσης μπορεί να χρησιμοποιηθούν οι *επιφάνειες διεπαφές χρήστη-εφαρμογής* (*User Interfaces* εφεξής) οι οποίες επιτρέπουν άμεση εισαγωγή πληροφοριών στην εφαρμογή. Τέτοια interfaces αποτελούν οι *επιφάνειες αυθεντικοποίησης (login screens), οι διαδικτυακές επιφάνειες διεπαφές (Web front ends)* κλπ.

Αυτό που πρέπει να ελεγχθεί σε αυτό το είδος δοκιμών διείσδυσης, είναι αν οι εισαγόμενες πληροφορίες ελέγχονται σωστά: Οι «αναμενόμενες» πληροφορίες να γίνονται δεκτές και οι «μη-αναμενόμενες» να απορρίπτονται. Παραδείγματα μη-αναμενόμενης εισόδου είναι οποιαδήποτε εν δυνάμει κακόβουλη είσοδος όπως μεγάλες γραμμές συνεχόμενων χαρακτήρων, αλλοιωμένα πακέτα κίνησης κτλ.

1.3 ΔΟΚΙΜΕΣ ΕΝΑΝΤΙΑ ΣΤΑ ΔΕΔΟΜΕΝΑ ΚΑΙ ΣΤΗ ΛΟΓΙΚΗ ΤΗΣ ΕΦΑΡΜΟΓΗΣ

Κάποια λάθη είναι κωδικοποιημένα μέσα στην εσωτερική αποθήκη πληροφοριών της ίδιας της εφαρμογής. Σε αυτές τις περιπτώσεις, λάθη στον κώδικα και στην σχεδίαση της δομής του κώδικα μπορεί να χρησιμοποιηθούν από ένα κακόβουλο χρήστη προκειμένου η εφαρμογή να μην λειτουργήσει με τον τρόπο που σχεδιάστηκε. Τέτοια λάθη συχνά οδηγούν στις επιθέσεις που είναι ευρέως γνωστές ως *επιθέσεις άρνησης εξυπηρέτησης και υπερχείλισης (Denial of service attacks και Buffer overflow attacks),* οι οποίες συχνά είναι επιτυχείς όταν οι προγραμματιστές της εφαρμογής δεν έχουν ελέγξει τα σημεία στον κώδικα που χρησιμοποιούν αυξανόμενες θέσεις αποθήκευσης πληροφοριών.

Πέρα από αυτό όμως, σοβαρή απειλή αποτελούν και τα λογικά λάθη που μπορεί να έχουν ενσωματωθεί εν αγνοία ή από παράλειψη των κατασκευαστών της εφαρμογής. Για παράδειγμα, η αλλαγή μιας φαινομενικά (για το σύστημα) σωστής πληροφορίας εισόδου μπορεί να επιστρέψει πληροφορίες που κανονικά ο χρήστης δεν θα έπρεπε να έχει πρόσβαση.

Τα επόμενα τρία κεφάλαια παρουσιάζουν και αναλύουν μια διαφορετική ταξινόμηση των δοκιμών διείσδυσης και των μεθόδων εύρεσης αδυναμιών, ανάλογα με το ποσό πληροφορίας που είναι διαθέσιμο στην μέθοδο εύρεσης αδυναμιών για την εφαρμογή υπό έλεγχο.

2. BLACK-BOX TESTING

Ο όρος «τεχνικές *μαύρου-κουτιού»* (*Black-box testing*) αναφέρεται στο σύνολο των μεθόδων δοκιμής λογισμικού οι οποίες ελέγχουν τη λειτουργικότητα και τις αδυναμίες μιας εφαρμογής χωρίς να ξέρουν την εσωτερική της δομή και λειτουργία, σε αντίθεση με τον έλεγχο της εσωτερικής δομής και την ανάλυση κώδικα γνωστή ως *white-box testing*. Κατά τη χρήση μεθόδων ελέγχου black-box, δεν υπάρχει διαθέσιμη -σχεδόν- καμία πληροφορία για την εφαρμογή. Σκοπός είναι να προσομοιωθούν όσο το δυνατόν πιο πειστικά, οι επιθέσεις ενός απομακρυσμένου κακόβουλου χρήστη ο οποίος δεν έχει πρόσβαση σε σημαντικές πληροφορίες της εφαρμογής[6].

ΕΙΚΟΝΑ 1: ΡΟΗ ΠΛΗΡΟΦΟΡΙΑΣ ΣΕ ΔΟΚΙΜΕΣ BLACK-BOX

Ο κώδικας της εφαρμογής, η δομή και ο τρόπος που αυτή επεξεργάζεται τις πληροφορίες δεν είναι διαθέσιμα προς χρήση κατά τη διάρκεια των ελέγχων τύπου black-box. Οι μέθοδοι ανάλυσης που εφαρμόζονται σε αυτές τις δοκιμές βασίζονται στα δεδομένα εισόδου και εξόδου της εφαρμογής, καθώς και στο τρόπο αλληλεπίδρασης αυτής με το περιβάλλον της. Τέτοιου είδους έλεγχοι κάνουν χρήση επιθυμητών αλλά και κακόβουλων δεδομένων ως δεδομένα εισόδου στην εφαρμογή και εξάγουν συμπεράσματα από τα δεδομένα εξόδου αυτής, τεχνική που συχνά συναντάται με τον όρο “point-and-click penetration testing” [14]. Οι κυριότερες τεχνικές που χρησιμοποιούνται ευρέως στους διάφορους ελέγχους black-box testing είναι οι ακόλουθες :

- *Δοκιμές Δεδομένων Εισόδου (Input Testing)*
  - *Δοκιμές Δυνατών Συνδυασμών (All-pairs testing)*
  - *Δοκιμές Ασάφειας (Fuzzing Tests)*
  - Δοκιμές ακραίων τιμών εισόδου (*Extreme Input Testing*)

2.1 ΔΟΚΙΜΕΣ ΔΕΔΟΜΕΝΩΝ ΕΙΣΟΔΟΥ

Μια συνηθισμένη πρακτική ελέγχου εφαρμογών που εμπίπτει στην κατηγορία του black box δοκιμών, είναι η χρήση εργαλείων όπως *περιηγητές ιστοσελίδων (web crawlers)*. Αυτά τα εργαλεία απαριθμούν όλες τις προσπελάσιμες σελίδες και καταστάσεις λειτουργίας μιας εφαρμογής διαδικτύου, παράγουν ειδικά διαμορφωμένες πληροφορίες για να δοθούν ως είσοδος στην εφαρμογή υπό έλεγχο κι έπειτα παρατηρούν την λειτουργία των εφαρμογών αυτών, προσπαθώντας να εντοπίσουν πιθανές αδυναμίες και ευπάθειες [10].

2.1.1 ΔΟΚΙΜΈΣ ΔΥΝΑΤΏΝ ΣΥΝΔΥΑΣΜΏΝ (ALL-PAIRS TESTING)

Οι *δοκιμές δυνατών συνδυασμών* (*All-pairs testing* ή αλλιώς *pairwise testing*) είναι ένα είδος συνδυαστικής δοκιμής η οποία, για κάθε δυνατό ζεύγος πληροφοριών εισόδου σε μια εφαρμογή, ελέγχει πρακτικά όλους τους πιθανούς, διακριτούς συνδυασμούς αυτών. Ακολουθεί ένα τυπικό παράδειγμα για καλύτερη κατανόηση αυτής της τεχνικής ελέγχου.

Έστω η ύπαρξη πέντε μεταβλητών ως input στην εφαρμογή και κάθε μία αναπαρίσταται με ένα γράμμα της αλφαβήτου. Έστω επίσης ότι κάθε μεταβλητή μπορεί να πάρει τιμές από 1 έως 5.

Table 1: Varying column E only					
Combination Number	A	B	C	D	E
1	1	1	1	1	1
2	1	1	1	1	2
3	1	1	1	1	3
4	1	1	1	1	4
5	1	1	1	1	5

ΠΙΝΑΚΑΣ 1: ΠΙΘΑΝΟΙ ΣΥΝΔΥΑΣΜΟΙ ΣΕ ΣΥΝΟΛΟ ΠΕΝΤΕ ΜΕΤΑΒΛΗΤΩΝ [1]

Στη πράξη, πέντε μεταβλητές με πέντε πιθανές καταστάσεις μας οδηγούν στον έλεγχο 3125 συνδυασμών (5 εις την $5^{η}$). Προκειμένου να ελεγχθούν όλες οι πιθανές είσοδοι στην εφαρμογή, πρέπει να εισαχθούν στο λογισμικό υπό έλεγχο και οι 3125 συνδυασμοί και να μελετηθούν τα αποτελέσματα από την επεξεργασία κάθε

εισαγωγής. Τέτοιου είδους αναλύσεις καλύπτουν συχνά διάφορες προσπάθειες εύρεσης *συνθηματικών (password)* σε διαδικασίες αυθεντικοποίησης χρηστών. Κλασσικό παράδειγμα επίθεσης που αξιοποιεί την τεχνική δοκιμής δυνατών συνδυασμών είναι οι *επιθέσεις λεξικού* (*dictionary attacks*) και οι *επιθέσεις ωμής προσβολής συνθηματικών* (*brute force attacks*).

Παρόλο που η ανωτέρω τεχνική είναι εκτενής και πληρεί τις προϋποθέσεις κάλυψης όλων των πιθανών συνδυασμών, εντούτοις κοστίζει πολύ σε χρόνο και σε υπολογιστική ισχύ προκειμένου να εκτελεστεί, κόστος το οποίο ανεβαίνει εκθετικά όσο αυξάνονται οι πιθανές τιμές και ο αριθμός των μεταβλητών εισόδου. Για τον έλεγχο μεγάλων συνθηματικών που συνδυάζουν πολλούς διαφορετικούς χαρακτήρες της κατηγορίας ASCII, η μέθοδος αυτή μπορεί να αποφέρει αποτελέσματα αλλά είναι πιθανό να κάνει μέρες ή ακόμα και μήνες μέχρι να εκτελεστεί πλήρως οπότε δεν ενδείκνυται για όλες τις περιπτώσεις δοκιμών ελέγχου.

2.1.2 ΔΟΚΙΜΈΣ ΑΣΑΦΕΙΑΣ (FUZZING TESTS)

Οι δοκιμές ελέγχου με το όνομα *Δοκιμές Ασάφειας* (*Fuzzing Tests*) επιτρέπουν στους ελεγκτές λογισμικού να βρουν αδυναμίες της υπό έλεγχο εφαρμογής, κάνοντας χρήση μεθόδων εισαγωγής αλλοιωμένων ή κακόβουλων πληροφοριών ως είσοδο, μέσω υπαρχουσών διεπαφών αλληλεπίδρασης του χρήση με την εφαρμογή (User Interfaces)[3].

Με άλλα λόγια, αυτή η τεχνική επιτρέπει τη αποκάλυψη των πιο εκτεθειμένων και επικίνδυνων *διεπαφών αλληλεπίδρασης, διεπαφών κατάλληλων για επίθεση (attack surfaces*) μέσα σε μία εφαρμογή. Ο όρος Attack surface ορίζεται συχνά ως το τμήμα του πηγαίου κώδικα μιας εφαρμογής (*source code fingerprint*) στο οποίο έχει πρόσβαση κανείς μέσω εξωτερικών, προσπελάσιμων διεπαφών όπως φόρμες διαδικτύου, γραφικές διεπαφές εφαρμογών, επαφές πρόσβασης στις λειτουργίες λογισμικού κ.α.

Οι διασημότερες κατηγορίες επιθέσεων στις υπηρεσίες μιας εφαρμογής περιλαμβάνουν και τις επιθέσεις σε διαδικτυακές εφαρμογές. Οι πιο γνωστές μέθοδοι περιλαμβάνουν [10] :

- *Εκμετάλλευση παραμέτρων σε φόρμες HTML*. Ο πιο συνηθισμένος τρόπος για να δέχεται παραμέτρους μια εφαρμογή διαδικτύου είναι μέσω των φορμών HTML. Όταν υποβάλλεται μια φόρμα, οι εκάστοτε παράμετροι στέλνονται ως ένα τμήμα

ενός HTTP αιτήματος. Ένας επιτιθέμενος μπορεί να αλλοιώσει τις παραμέτρους που αποστέλλονται στο σύστημα, αλλάζοντας τις τιμές των πεδίων στις φόρμες αποστολής HTML.

- Εκμετάλλευση δικτυακών συνδέσμων *URL*. Στις αιτήσεις τύπου HTTP GET, η παράμετροι καθώς και οι πιθανές τιμές αυτών εμφανίζονται ως τμήμα του *δικτυακού συνδέσμου (link)* μόλις η αίτηση καταχωρηθεί. Ένας επιτιθέμενος μπορεί να αλλάξει τη συμβολοσειρά του συνδέσμου απευθείας και να χρησιμοποιήσει τη νέα μορφή του συνδέσμου για να αποκτήσει πρόσβαση που κανονικά δεν θα έπρεπε να έχει.
- *Εκμετάλλευση πεδίων τύπου HIDDEN σε κώδικα HTML*. Λόγω του ότι το πρωτόκολλο HTTP δεν διατηρεί την *κατάσταση λειτουργίας του (state)* - τουλάχιστον η έκδοση 1.0-, πολλές διαδικτυακές εφαρμογές χρησιμοποιούν κρυμμένα πεδία τιμών για να προσομοιάσουν αυτή τη διατήρηση πληροφοριών. Τα κρυμμένα πεδία είναι ουσιαστικά πεδία εισαγωγής πληροφοριών που δεν εμφανίζονται στο χρήστη. Από τη στιγμή που το κρυφό πεδίο είναι τμήμα της φόρμας HTML της σελίδας, αν ένας επιτιθέμενος αποθηκεύσει τη σελίδα, αλλοιώσει την τιμή ενός κρυμμένου πεδίου και φορτώσει ξανά την σελίδα, τότε θα αναγκάσει την εφαρμογή να παραλάβει τα αλλοιωμένα δεδομένα του κρυφού πεδίου. Αυτή η τεχνική χρησιμοποιείται ευρέως για επιθέσεις σε βάσεις δεδομένων με παραποίηση ερωτημάτων SQL.
- *Δηλητηρίαση μικρών αρχείων κειμένου (Cookie Poisoning):* Αυτή η τεχνική αναφέρεται στη τροποποίηση ενός μικρ*ού αρχείου κειμένου* (*cookie*, αγγλικά εφεξής) τα οποία αποθηκεύονται στον υπολογιστή κατά την πλοήγησή μας στο διαδίκτυο. Συνήθως περιγράφουν στοιχεία όπως όνομα χρήστη (user name) και συνθηματικό πρόσβασης (password) με σκοπό κατά την επίσκεψή στον ίδιο ιστότοπο αργότερα, ο υπολογιστής να "θυμάται" και να κάνει login ή άλλες εργασίες αυτόματα. Στο θέμα της ασφάλειας, μπορούν να εισαχθούν κακόβουλα δεδομένα εισόδου στην εφαρμογή μέσω τιμών που αποθηκεύονται στα cookies.
- *Έγχυση πληροφοριών SQL (SQL Injection)*: Η επίθεση αυτή αφορά την αποστολή πληροφοριών εισόδου που περιέχουν εντολές SQL για εκτέλεση ερήμην του προγράμματος. Επιθέσεις τέτοιου τύπου μπορεί να οδηγήσουν στη διαρροή και αλλοίωση πολλών πληροφοριών, από την δομή της βάσης δεδομένων μέχρι μεμονωμένες πληροφορίες συγκεκριμένων τιμών της βάσης. Οι επιθέσεις αυτές συνήθως υλοποιούνται στα διανύσματα εισόδου της εφαρμογής, όπως πεδία τιμών στις διεπαφές χρήστη κ.α.
- *Εκτέλεση σειράς εντολών μέσω διαφορετικών, εξωτερικών ιστοσελίδων (Cross-Site Scripting - XSS)*: Αυτός ο τρόπος επίθεσης βασίζεται στην εκμετάλλευση εφαρμογών που αποστέλλουν μη-ελεγχόμενη πληροφορία προκειμένου να

μπερδέψουν το σύστημα στο να εκτελέσει κακόβουλο κώδικα. Στις επιθέσεις XSS, ένας επιτιθέμενος μπορεί να προσθέσει κακόβουλες εντολές τύπου JavaScript μέσα σε δυναμικά παραγόμενο περιεχόμενο έμπιστων ιστοσελίδων. Όταν οι εντολές αυτές εκτελεστούν στον υπολογιστή του χρήστη που θα δει τη σελίδα, θα μπορέσουν να προχωρήσουν σε μια πληθώρα κακόβουλων πράξεων, από αλλαγή πληροφοριών μέχρι κλοπή συνθηματικών, ενέργειες που θα διεξαχθούν δίχως την συγκατάθεση των νόμιμων κατόχων των πληροφοριών.

- *Διαχωρισμός απαντήσεων τύπου HTTP (HTTP Response Splitting)*: Η επίθεση διαχωρισμού απαντήσεων HTTP στοχεύει στην εκμετάλλευση πληροφοριών που αποστέλλει η εφαρμογή προκειμένου να πραγματοποιήσει επιθέσεις τύπου *Δηλητηρίασης της προσωρινής μνήμης της ιστοσελίδας* (*Web Cache Poisoning*, αγγλικός όρος εφεξής). Με το να παρέχει ένας κακόβουλος χρήστης χαρακτήρες CR και LF (όπως οι χαρακτήρες αλλαγής γραμμής) μπορεί να προκαλέσει την δημιουργία δύο HTTP απαντήσεων για μία αίτηση. Με το να ελέγχει την δεύτερη απάντηση, ο επιτιθέμενος μπορεί να προκαλέσει πληθώρα προβλημάτων στην εφαρμογή.

Ένα παράδειγμα δοκιμής Ασάφειας θα ήταν η απομακρυσμένη αποστολή κακόβουλων αιτημάτων διαδικτύου σε μια διαδικτυακή φόρμα αλληλεπίδρασης χρήστη – εφαρμογής. Οι συνηθέστερες κατηγορίες *απομακρυσμένων επιφανειών (remote interfaces)* που αξιοποιούνται από τα fuzzing tests για να ψάξουν για αδυναμίες αξιοποιώντας την αποστολή κακόβουλων αιτημάτων, είναι οι ακόλουθες:

- Διαδικτυακές Εφαρμογές: Οι φόρμες των ιστοσελίδων του διαδικτύου αποτελούν μια από τις πιο γνωστές επιφάνειες αλληλεπίδρασης. Σχεδόν το 50% των κοινών αδυναμιών που παρουσιάζονται στους ελέγχους ασάφειας, προέρχονται από test που εκτελούνται πάνω σε εφαρμογές που είναι συνδεδεμένες σε δίκτυο και προσφέρουν κάποια υπηρεσία.
- Ψηφιακά Πολυμέσα: Οι τύποι των αρχείων πολυμέσων μεταφέρονται σαν ένθετες πληροφορίες μέσα στα πακέτα και διαμέσω του διαδικτύου όπως παραδείγματος χάρη μέσω του ηλεκτρονικού ταχυδρομείου. Διάφοροι έλεγχοι ασάφειας αποστέλλουν πληθώρα κακόβουλων πληροφοριών μέσω τέτοιων πακέτων προσπαθώντας να ενεργοποιήσουν τυχόν ευπάθειες σε τομείς όπως η προσωρινή μνήμη για αναπαραγωγή πολυμέσων, η αυθεντικοποίηση πληροφορίας ενδιάμεσων ελέγχων από προγράμματα προστασίας κτλ.
- Διαδικτυακά Πρωτόκολλα: Μια δοκιμή ασάφειας πάνω σε ένα διαδικτυακό πρωτόκολλο μπορεί να εξετάσει τόσο τους διακομιστές, όσο και τους παραλήπτες

πληροφοριών για ευπάθειες που έχουν χρησιμοποιούν πιθανές αδυναμίες του ίδιου του πρωτόκολλου [3].

2.1.3 ΔΟΚΙΜΈΣ ΑΚΡΑΙΩΝ ΤΙΜΏΝ ΕΙΣΌΔΟΥ

Η χρήση ακραίων τιμών ως δεδομένα εισόδου είναι μία τεχνική ελέγχου λογισμικού κατά την οποία οι δοκιμές τροφοδοτούν ακραίες τιμές στα διανύσματα εισόδου της εφαρμογής υπό έλεγχο. Αυτές οι τιμές αποτελούν οριακές τιμές μέσα σε ένα εύρος τιμών που ορίζεται ανάλογα με το είδος των δεδομένων εισόδου και εξόδου που παρουσιάζει η εφαρμογή. Οι οριακές τιμές εισόδου και εξόδου μιας εφαρμογής αποτελούν συχνά πεδίο που παρουσιάζει αυξημένα προβλήματα λειτουργικότητας και λαθών. Ένα παράδειγμα θα ήταν η χρήση τιμών –2147483648 ή μικρότερες ως δεδομένα εισόδου σε ένα διάνυσμα εισόδου που δέχεται ακέραιους αριθμούς (Υπενθυμίζεται πως το εύρος των ακέραιων τιμών στις γλώσσες προγραμματισμού είναι –2147483648 έως 2147483647).

Η μορφή και το είδος των αναμενόμενων τιμών για τα δεδομένα εισόδου και εξόδου πρέπει να εξαχθούν με κάποιο τρόπο από την εφαρμογή υπό έλεγχο. Οι τιμές αυτές μετά ομαδοποιούνται σε σύνολα με συγκεκριμένα όρια τιμών. Κάθε σύνολο περιέχει τιμές που είναι «αναμενόμενες» από την εφαρμογή και τίθενται υπό επεξεργασία με συγκεκριμένο τρόπο. Η χρήση τιμών οριακά ή έξω από τα σύνολα μπορεί να οδηγήσει σε αποκάλυψη ευπαθειών της εφαρμογής.

Το επόμενο κεφάλαιο εστιάζει στην δεύτερη κατηγορία τεχνικών για εύρεση αδυναμιών και τρόπων διείσδυσης, την κατηγορία λευκού-κουτιού (white-box, χρήση αγγλικού όρου εφεξής). Σε αντίθεση με τις δοκιμές black-box, στις white-box υπάρχει πλήρης πρόσβαση στον πηγαίο κώδικα, στις πληροφορίες, στη δομή και στη λειτουργία της εφαρμογής υπό έλεγχο.

3. WHITE-BOX TESTING

Οι τεχνικές Δοκιμών Διείσδυσης που συγκαταλέγονται στη κατηγορία των δοκιμών *Λευκού τύπου* (*White-box* εφεξής), είναι οι μέθοδοι ελέγχου λογισμικού που μελετούν τις εσωτερικές δομές και λειτουργίες μιας εφαρμογής. Σε αντίθεση με τις δοκιμές τύπου black-box, στις δοκιμές white-box παρέχονται όλες οι πληροφορίες, ο κώδικας, τα διαγράμματα σχεδίασης, η δομή κατασκευής και τα τεχνικά χαρακτηριστικά της εφαρμογής στον αναλυτή πριν την ανάλυση, χωρίς να γίνεται καμία παρακράτηση πληροφοριών. Τέτοιου είδους τεχνικές έχουν μεγαλύτερο ποσοστό επιτυχίας από τις αντίστοιχες του black-boxing, καθαρά λόγω υλικού προς εξέταση, αφού στο black-boxing ο αναλυτής λειτουργεί ως εξωτερικός επιτιθέμενος έχοντας μηδενικές πληροφορίες για την εφαρμογή υπό μελέτη.

Τόσο οι αναλυτές, όσο και οι τεχνικές που χρησιμοποιούν χωρίζονται σε κατηγορίες. Ο *αναλυτής ασφάλειας (security tester)* έχει ως στόχο την εύρεση παραλείψεων και ελλιπών διαδικασιών οι οποίες μπορούν να αλλοιώσουν την προδιαγεγραμμένη λειτουργία μιας εφαρμογής η να παραβιάσουν την *εμπιστευτικότητα*, *ακεραιότητα* και *διαθεσιμότητα* (*Confidentiality*, *Integrity*, *Availiability* – C.I.A.) των πληροφοριών που αυτή διαχειρίζεται. Συμπληρωματικά, ο *αναλυτής λογισμικού (software tester)* ελέγχει το λειτουργικό κομμάτι της εφαρμογής, ελέγχοντας τη δομή του κώδικα προκειμένου να αξιολογήσει την προγραμματιστική ευρωστία του λογισμικού. Οι δύο στόχοι είναι συμπληρωματικοί και συχνά επικαλύπτονται σε κάποια σημεία, ειδικά στους ελέγχους white-box που μελετάται εκτενώς ο κώδικας της εφαρμογής. Οι πλέον διαδεδομένες τεχνικές white-boxing που χρησιμοποιούνται γενικότερα από τους αναλυτές του χώρου είναι οι ακόλουθες:

- Έλεγχοι με Εισαγωγή Δεδομένων (Input Application Testing)
- Ανάλυση Πηγαίου Κώδικα (Code Audit)
    - Λειτουργικοί Έλεγχοι (Function / Unit Testing)
    - Έλεγχοι Σχεδιασμού (Design Testing)
    - Έλεγχοι της ροής εκτέλεσης (Control Flow Testing)
    - Έλεγχοι ροής πληροφοριών (Data Flow Testing)
- Έλεγχοι εκτελέσιμων αρχείων (Binary Executable File Testing)
    - Έλεγχοι PE επικεφαλίδας (PE Header Testing)
    - Έλεγχοι εντολών συστήματος (Machine-level Testing)
- Έλεγχοι Λογικής (Application Logic Tests)
    - Πίνακες Απόφασης
    - Πίνακες Μετάβασης Καταστάσεων

Ακολουθεί η περιληπτική παρουσίαση των προαναφερθέντων τεχνικών που χρησιμοποιούνται στους ελέγχους white-box, τόσο από τους αναλυτές ασφάλειας, όσο και από τους αναλυτές συστημάτων.

3.1 ΕΛΕΓΧΟΙ ΜΕ ΕΙΣΑΓΩΓΗ ΔΕΔΟΜΕΝΩΝ (INPUT APPLICATION TESTING)

Κατά τις δοκιμές εισόδου δεδομένων, γίνεται προσπάθεια εύρεσης αδυναμιών της εφαρμογής προσπαθώντας να «διαρρήξουν» την εκάστοτε πλατφόρμα, υπηρεσία, τεχνολογία που αυτή χρησιμοποιεί. Τέτοια παραδείγματα αποτελούν η εύρεση αδυναμιών και επικίνδυνων προεγκατεστημένων μεθόδων σε συγκεκριμένες εκδόσεις των υπηρεσιών που τρέχουν σε εφαρμογές διαδικτύου (PHP, Joomla, HTML κτλ), στις εξωτερικές δυναμικές βιβλιοθήκες (DLL), σε συναρτήσεις κώδικα που χρησιμοποιούνται «ως-έχει» (όπως στις βιβλιοθήκες της Java) ή ακόμη και στις εκδόσεις των μεταγλωττιστών της γλώσσας προγραμματισμού που χρησιμοποιήθηκε μέσω χρήσης τεχνικών τύπου Δυνατών Συνδυασμών (2.1.1), για εύρεση συναρτήσεων επιρρεπών σε επιθέσεις υπερχείλισης, όπως η strcpy() κτλ.

3.2 ΑΝΑΛΥΣΗ ΠΗΓΑΙΟΥ ΚΩΔΙΚΑ (CODE AUDIT)

3.2.1 ΛΕΙΤΟΥΡΓΙΚΟΙ ΕΛΕΓΧΟΙ (FUNCTION / UNIT TESTING)

Συνηθισμένη πρακτική είναι η μελέτη του πηγαίου κώδικα της εφαρμογής προκειμένου να εντοπισθούν πιθανές αδυναμίες και ελαττωματικές συναρτήσεις. Σε αυτές τις τεχνικές γίνεται χρήση αυτόματων μηχανισμών που εντοπίζουν γνωστές, επικίνδυνες συναρτήσεις γλωσσών προγραμματισμού, βρόχους επαναλήψεων χωρίς έλεγχο των καταχωρητών μνήμης και άλλες δομές, οι οποίες μπορεί να οδηγήσουν σε κατάρρευση της εφαρμογής υπό συγκεκριμένες συνθήκες.

3.2.2 ΕΛΕΓΧΟΙ ΣΧΕΔΙΑΣΜΟΥ (DESIGN TESTING)

Κατά τους ελέγχους σχεδιασμού γίνεται ανάλυση των διεπαφών και των συστατικών που χρησιμοποιεί η εφαρμογή για τυχόν αδυναμίες ή ευπάθειες. Μελετάται η δομή του κώδικα και το κατά πόσον εύρωστα προγραμματισμένη είναι η εφαρμογή γενικότερα. Λάθη στις διεπαφές επικοινωνίας του κώδικα ή στη δομή

λειτουργίας του μπορεί να οδηγήσουν συχνά σε λογικές αδυναμίες, αδυναμίες που γίνονται δύσκολα ή και ποτέ αντιληπτές από αυτόματους *σαρωτές αδυναμιών* (*vulnerability scanners*).

3.2.3 ΕΛΕΓΧΟΙ ΤΗΣ ΡΟΗΣ ΕΚΤΕΛΕΣΗΣ (CONTROL FLOW TESTING)

Ο *έλεγχος ροής* και *εκτέλεσης* του πηγαίου κώδικα (*execution path testing*) ασχολείται με την σειρά κατά την οποία θα εκτελεστούν οι διακριτές συναρτήσεις και εντολές κώδικα κατά τη ορθή λειτουργία του προγράμματος καθώς και μελέτη του *μονοπατιού εκτέλεσης* του προγράμματος (*execution path*), βήμα-βήμα στο κώδικα που παρέχεται. Υπάρχουν διάφορα είδη ελέγχου ροής κώδικα τα οποία διαφοροποιούνται και ανάμεσα στις γλώσσες προγραμματισμού. Μερικά εκ των οποίων είναι τα ακόλουθα, κατηγοριοποιημένα ανάλογα με τον αντίκτυπο που έχουν στη ροή εκτέλεσης των εντολών:

- Ασυνεχής μετάβαση εκτέλεσης σε άλλο κομμάτι κώδικα (unconditional branch / jump)
- Εκτέλεση κώδικα υπό προϋποθέσεις μόνο (conditional branch),
- Επανάληψη εκτέλεσης μπλοκ εντολών μέχρι την ικανοποίηση κάποιας προϋπόθεσης (loop),
- Εκτέλεση υπορουτίνας ή άλλης συνάρτησης και επιστροφή στην προηγούμενη ροή εκτελέσεων (υπορουτίνες, functions),
- Παύση της εκτέλεσης της εφαρμογής χωρίς δυνατότητα επανάληψης λειτουργίας από το σημείο που σταμάτησε (unconditional halt) [23].

3.2.4 ΕΛΕΓΧΟΙ ΡΟΗΣ ΔΕΔΟΜΕΝΩΝ (DATA FLOW TESTING)

Οι τεχνικές *ελέγχου ροής δεδομένων* (*data flow testing*) του πηγαίου κώδικα είναι τεχνικές που χρησιμοποιούν τις σχέσεις και τα βήματα της ροής δεδομένων μέσα σε ένα πρόγραμμα προκειμένου να ελέγχονται κατά βούληση οι τρόποι επεξεργασίας δεδομένων από τις εφαρμογές υπό έλεγχο. Οι υπάρχουσες τεχνικές dataflow ελέγχου ροής δεδομένων μπορούν να εφαρμοστούν τόσο σε ξεχωριστές μεθόδους, όσο και σε συναρτήσεις κώδικα μέσα σε μία κλάση οι οποίες επικοινωνούν μεταξύ τους μέσω μηνυμάτων.

3.3 BINARY EXECUTABLE FILE TESTING

Τα εκτελέσιμα αρχεία μπορούν να ελεγχθούν για αδυναμίες ακόμα και χωρίς την πρόσβαση στον πηγαίο τους κώδικα. Σε αυτές τις περιπτώσεις, τα εκτελέσιμα αρχεία ελέγχονται σε επίπεδο δυαδικού κώδικα για αδυναμίες ασφάλειας όπως υπερχείλιση, αδύναμες εντολές, ευπάθειες εκμετάλλευσης μνήμης ή αρχείου κτλ.

3.3.1 PE HEADER TESTING

Αυτή η μέθοδος χρησιμοποιεί την πληροφορία που είναι αποθηκευμένη στο PE header (τμήματα πληροφορίας και πίνακες) ενός Windows NT/XP *εκτελέσιμου αρχείου (executable file)*, μαζί με την πληροφορία που πηγάζει από τα συνολικά περιεχόμενα του, ως τρόπο για να εντοπιστούν αδυναμίες, ευπάθειες και ανωμαλίες στην ασφάλεια του λογισμικού που μπορεί να φανούν χρήσιμες σε κακόβουλους χρήστες [8].

Η μέθοδος αυτή εφαρμόζεται στα PE Window Files και δεν πραγματοποιεί *αποσυναρμολόγηση σε κώδικα μηχανής (disassembling)* του εκτελέσιμου. Τα εκτελέσιμα αρχεία τύπου PE, είναι από τις πιο διαδεδομένες μορφές εκτελέσιμων αρχείων στο λειτουργικό σύστημα Windows. Η μέθοδος αποτελεί κλασσικό παράδειγμα Δοκιμών Διείσδυσης σε εφαρμογή μέσω εύρεσης ευπαθειών σε εκτελέσιμο αρχείο, αφού μπορεί να εντοπίσει μια πληθώρα τρόπων διείσδυσης ή διακοπής λειτουργίας μιας εφαρμογής, καθώς και επιπλέον κινδύνους όπως, παραδείγματος χάριν, την ύπαρξη κακόβουλου λογισμικού το οποίο ενσωματώνει τον εαυτό του μέσα στο δυαδικό χώρο μιας εφαρμογής προκειμένου να μεταφερθεί από σύστημα σε σύστημα. Περιληπτικά, μέσω της επικεφαλίδας PE εντοπίζονται :

- Λανθασμένα μεγέθη πινάκων επικεφαλίδας
- Μεγάλες μηδενικές περιοχές από byte που μπορούν να χρησιμοποιηθούν για ανεπιθύμητη εκτέλεση κώδικα. Τέτοιες περιοχές χρησιμεύουν ως αποθήκες στις οποίες οι κακόβουλοι χρήστες μπορούν να ενσωματώσουν δικές τους εντολές και μετά να τις καλέσουν μέσα από την εκτέλεση του προγράμματος.
- Συμπιεσμένα αρχεία μέσα στη πληροφορία της εφαρμογής.
- Τομείς του εκτελέσιμου που είναι ταυτόχρονα ανοιχτοί σε εγγραφή και εκτέλεση εντολών, γεγονός που οδηγεί σε κακόβουλη εκτέλεση εντολών μέσα από την εφαρμογή χωρίς εγκεκριμένη χρήση.
- Μεθόδους και διαδικασίες που είναι επιρρεπείς σε επιθέσεις υπερχείλισης (buffer overflow application attacks).

3.3.2 MACHINE-LEVEL TESTING

Οι κυριότερες μέθοδοι ανάλυσης *δυαδικών, εκτελέσιμων αρχείων* (*binary executable files*) είναι αυτές που πραγματοποιούν disassembling του εκτελέσιμου αρχείου υπό έλεγχο και ύστερα εφαρμόζουν κάποιο είδος στατικής μελέτης στο παραγόμενο αποτέλεσμα κώδικα assembly. Μία τέτοια χαρακτηριστική μέθοδος είναι και η ανάλυση εκτελέσιμων αρχείων για επεξεργαστές τύπου Intel x86 [12].

Στην ανωτέρω έρευνα, η ανάλυση επικεντρώνεται στη χρήση στατικού ελέγχου του δυαδικού κώδικα της εφαρμογής αλλά και στην συμβολική εκτέλεση του προγράμματος προκειμένου να βρεθούν τυχόν αδυναμίες και ευπάθειες. Ο βασικός λόγος χρήσης συμβολικού ελέγχου είναι η προσπάθεια να προσπελαστούν όλα τα πιθανά μονοπάτια εκτέλεσης κώδικα. Αυτή η τεχνική είναι μία από τις τεχνικές που θα χρησιμοποιηθούν και από την υλοποίησή μας για προσδιορισμό λογικών αδυναμιών.

3.4 ΕΛΕΓΧΟΙ ΛΟΓΙΚΗΣ (APPLICATION LOGIC TESTS)

Κατά τους έλεγχους λογικής (application logic tests), γίνεται προσπάθεια να εντοπιστούν λάθη στη λογική διαδικασία των βημάτων εκτέλεσης εντολών μέσα στην εφαρμογή (steps in execution path). Τέτοια λάθη συνήθως δεν γίνονται αντιληπτά με τον έλεγχο του συστήματος, καθώς αποτελούν λάθη μέσα στα λογικά βήματα λειτουργίας της εφαρμογής και είναι άμεσα συσχετισμένα με το τί κάνει η εφαρμογή σαν εφαρμογή και όχι με την τεχνολογία που αυτή χρησιμοποιεί για να λειτουργεί.

Συγκεκριμένα, η διαδικασία που ακολουθεί μια εφαρμογή προκειμένου να εκτελέσει το σκοπό της, αποτελείται από ένα σύνολο διαδοχικών βημάτων που, μόλις ολοκληρωθεί το ένα, δίνει τα σκήπτρα στο επόμενο. Παραδείγματος χάρη, η αγορά ενός προϊόντος από ένα διαδικτυακό μαγαζί προϋποθέτει ότι ο χρήστης θα εκτελέσει μια σειρά βημάτων προκειμένου να κάνει την αγορά: Αυθεντικοποίηση, επιλογή προϊόντος, συμπλήρωση στοιχείων, πληρωμή και έξοδο. Αυτή η λογική συνέχεια αποτελεί τη λογική διαδικασία της εφαρμογής. Οι ευπάθειες που προκύπτουν από λανθασμένο προγραμματισμό των λογικών αυτών βημάτων, δηλαδή τα λάθη που μπορεί να δημιουργήσουν παρακαμπτήριες οδούς στην εκτέλεση της λογικής διαδικασίας, αποτελούν τον στόχο των Ελέγχων Λογικής.

Ο έλεγχος λογικών ευπαθειών πραγματοποιείται κυρίως ως τεχνική white box, αφού η εξαγωγή πληροφοριών για τη λογική χαρτογράφηση αδυναμιών είναι εξαιρετικά δύσκολη υπόθεση για τους ελέγχους διείσδυσης, λόγω της έλλειψης

βασικών και αναγκαίων πληροφοριών για την εφαρμογή υπό έλεγχο, πληροφοριών γύρω από την σημασιολογία της.

3.4.1 ΠΙΝΑΚΕΣ ΑΠΟΦΑΣΗΣ

Μια αναπαράσταση της λογικής που χρησιμοποιείται για ανάλυση της εφαρμογής (και που ίσως βοηθήσει στον εντοπισμό λογικών αδυναμιών) παρουσιάζεται με τους πίνακες απόφασης. Ένας πίνακας απόφασης χωρίζεται σε τέσσερα μέρη όπως φαίνεται παρακάτω:

(Application Logic Tests)	
(Application Logic Tests)	Condition alternatives
(Application Logic Tests)	Action entries

ΠΙΝΑΚΑΣ 2: ΤΑ ΤΕΣΣΕΡΑ ΤΕΤΑΡΤΗΜΟΡΙΑ ΛΟΓΙΚΩΝ ΑΠΟΦΑΣΕΩΝ [24]

Κάθε επιλογή μονοπατιού στη λειτουργία του συστήματος αντιστοιχεί σε μία μεταβλητή, σχέση ή ενέργεια, της οποίας οι πιθανές τιμές καταγράφονται κατά μήκος των εναλλακτικών επιλογών δράσης της εφαρμογής. Κάθε εκτέλεση του λογισμικού είναι μια διαδικασία ή λειτουργία και οι εγγραφές του πίνακα δείχνουν αν και σε ποιά σειρά η εκτέλεση της διαδικασίας πρέπει να γίνει για το σύνολο παραμέτρων που της αντιστοιχεί.

3.4.2 ΠΙΝΑΚΕΣ ΜΕΤΑΒΑΣΗΣ ΚΑΤΑΣΤΑΣΕΩΝ

Ένας πίνακας μετάβασης καταστάσεων δείχνει σε ποιά κατάσταση (state) ένα πεπερασμένο σύνολο μεταβάσεων στο μονοπάτι εκτέλεσης θα κινηθεί, ανάλογα από τη θέση που βρίσκεται την εκάστοτε στιγμή. Πρακτικά, ένας Πίνακας Μετάβασης περιέχει τα πιθανά δεδομένα εισόδου της τωρινής κατάστασης και τα αναμενόμενα δεδομένα εξόδου της δείχνουν την επόμενη κατάσταση λειτουργίας στα λογικά βήματα.

ΔΥΣΔΙΑΣΤΑΤΟΙ ΠΙΝΑΚΕΣ ΜΕΤΑΒΑΣΗΣ

Οι πίνακες μετάβασης χρησιμοποιούνται συνήθως στην δυσδιάστατη μορφή τους. Η κάθετη διάσταση περιέχει τις παρούσες καταστάσεις της εφαρμογής (Current State), η οριζόντια διάσταση περιέχει τα Γεγονότα (Events) και τα κελιά του πίνακα

περιέχουν την επόμενη κατάσταση στην οποία θα μεταπηδήσει το execution path αν συμβεί το αντίστοιχο Γεγονός μαζί με την εκτέλεση λειτουργίας που συνδέεται με την εκάστοτε μετάβαση κατάστασης.

State Transition Table				
Events State	E_1	E_2	...	E_n
S_1	-	A_y/S_j	...	-
S_2	-	-	...	A_x/S_i
...	...	...	...	...
S_m	A_z/S_k	-	...	-

ΠΙΝΑΚΑΣ 3: ΠΑΡΑΔΕΙΓΜΑ ΣΧΕΣΕΩΝ ΓΕΓΟΝΟΤΩΝ ΚΑΤΑΣΤΑΣΕΩΝ

(S: state, E: event, A: action, -: illegal transition)

Το επόμενο κεφάλαιο παρουσιάζει το τρίτο και τελευταίο είδος δοκιμών διείσδυσης και εύρεσης αδυναμιών, το οποίο αποτελεί την γκρίζα ζώνη ανάμεσα στα δύο είδη δοκιμών black-box και white-box.

4. GRAY-BOX TESTING

Παρόλο που οι κυρίαρχες κατηγορίες του penetration testing είναι οι δύο προαναφερθείσες, η white box και η black box, στη πραγματικότητα όλες οι τεχνικές ελέγχου λογισμικού είναι ουσιαστικά ενδιάμεσες ή αλλιώς «γκρι» (Gray-Box Testing). Η εκάστοτε «απόχρωση» του γκρι, το κατά πόσο δηλαδή ένας έλεγχος εφαρμογής έχει περισσότερα στοιχεία white-box ή black-box ελέγχου, επηρεάζεται πολύ από τον εκάστοτε έλεγχο, την εκάστοτε επιλεγμένη διαδικασία και, φυσικά, από την εφαρμογή που μελετάται και τις πληροφορίες που υπάρχουν διαθέσιμες για αυτή κατά την διάρκεια ελέγχου. Συνήθως χρησιμοποιούνται οι απόλυτοι και ακραίοι χαρακτηρισμοί «λευκό-μαύρο» προκειμένου να απλοποιηθεί η ομαδοποίηση.

Πραγματικές καταστάσεις διακριτού και απόλυτου white-box ή black-box testing σπάνια συναντώνται αφού, συνήθως, υπάρχουν διαθέσιμες κάποιες πληροφορίες για το σύστημα υπό έλεγχο πριν το penetration test, έστω και σε περιορισμένη έκταση (οπότε δεν νοείται σαν πραγματικό black-box test) και ταυτόχρονα σχεδόν ποτέ δεν είναι διαθέσιμες όλες οι πληροφορίες, απλά και μόνο γιατί σπάνια δίνεται τόσο πλήρης και εκτεταμένη βοήθεια από τους κατασκευαστές του λογισμικού (οπότε δεν είναι ποτέ πραγματικά κατ’ εξοχήν white-box test).

Το επόμενο κεφάλαιο παρουσιάζει τα βασικά είδη των εργαλείων που χρησιμοποιούνται συχνά σε διαδικασίες εύρεσης αδυναμιών και σε δοκιμές διείσδυσης. Κάθε κατηγορία εργαλείων αναλύεται και εξηγείται ο τρόπος λειτουργίας τους και οι μέθοδοι και διαδικασίες πάνω στις οποίες βασίζονται προκειμένου τα εργαλεία αυτά να εξάγουν τα συμπεράσματα τους.

5. ΜΕΘΟΔΟΙ ΚΑΙ ΕΡΓΑΛΕΙΑ ΣΤΙΣ ΔΟΚΙΜΕΣ ΔΙΕΙΣΔΥΣΗΣ

Η διαδικασία εφαρμογής ενός Ελέγχου Διείσδυσης ξεκίνησε ως μία κατ' εξοχήν χειροκίνητη διαδικασία ελέγχου. Οι αναλυτές των λογισμικών ανέπτυσσαν ικανότητες εντοπισμού λαθών μέσα από πολυετή και συνεχή επαφή με εφαρμογές του χώρου. Αυτή η καθαρά εμπειρική εκτίμηση των αδυναμιών συχνά μπορεί να συνοδεύονταν από μια ευρεία γκάμα εργαλείων τα οποία βοηθούσαν τον αναλυτή να εξάγει τα επιθυμητά συμπεράσματα.

Παρόλα αυτά, η βάση της δουλειάς γινόταν από τον εμπειρογνώμονα, γεγονός που καθιστούσε την αξιοπιστία ενός ελέγχου διείσδυσης υπό αμφισβήτηση. Ο ανθρώπινος παράγοντας παρείχε κάποια πλεονεκτήματα αλλά και κάποια μειονεκτήματα στη διαδικασία ελέγχου. Τα δύο συνηθέστερα προβλήματα ήταν να εντοπίζονται αδυναμίες που, εν τέλει, δεν υπήρχαν (λανθασμένα θετικά αποτελέσματα, *false positives*), αλλά και να μην εντοπίζονται αδυναμίες που υπήρχαν στο πρόγραμμα υπό έλεγχο, αλλά δεν τις αντιλήφθηκε ο αναλυτής (*true negatives).* Από την άλλη μεριά, η συνδυαστική σκέψη ενός αναλυτή και η λογική κρίση του μπορούν να εντοπίσουν πολύπλοκα λάθη –κυρίως στη λογική διαδικασία λειτουργίας των εφαρμογών- που τα εργαλεία του χώρου δύσκολα θα μπορούσαν να εντοπίσουν λόγω έλλειψης κριτικής ικανότητας.

Με την ανάπτυξη του τομέα της Ασφάλειας και την έλευση εργαλείων αυτόματου ελέγχου εφαρμογών, οι διαδικασίες ελέγχου εξελίχθηκαν. Η χρήση τέτοιων εργαλείων αυτόματης δοκιμής καθιερώθηκε ως μια συνηθισμένη –και συχνά αναγκαία- πρακτική κατά την Δοκιμή Διείσδυσης (Penetration Testing) σε ένα πρόγραμμα ή σύστημα υπό μελέτη. Παρά το γεγονός ότι τα αυτόματα εργαλεία μπορούν να εντοπίσουν μια πληθώρα αδυναμιών σε σύντομο χρόνο –εύρος αδυναμιών που στον αναλυτή θα έπαιρνε πολύ χρόνο για να εξετάσει σχολαστικά-, είναι σχεδόν αδύνατο να αντικαταστήσουν πλήρως τον αναλυτή, καθώς η κριτική ικανότητα που προέρχεται από την εμπειρία, είναι δύσκολο να προσομοιωθεί πλήρως.

Τα εργαλεία που χρησιμοποιούνται στο ευρύτερο φάσμα του Penetration Testing για ευπάθειες, ανήκουν γενικά σε δύο ευρύτερες κατηγορίες εργαλείων, ανάλογα με την μέθοδο ανάλυσης που αυτά χρησιμοποιούν: Στα εργαλεία Στατικής ανάλυσης και στα εργαλεία Δυναμικής ανάλυσης.

5.1 ΣΤΑΤΙΚΗ ΑΝΑΛΥΣΗ

Τα *εργαλεία Στατικής Ανάλυσης (Static analysis tools)* ελέγχουν τον κώδικα των προγραμμάτων και συστημάτων (είτε αυτός παρέχεται με τη μορφή πηγαίου κώδικα, είτε σε δυαδική μορφή) προσπαθώντας να αναγνωρίσουν λάθη και αδυναμίες που εισχώρησαν κατά την ανάπτυξη του συστήματος ή προγράμματος, όπως αδυναμίες τύπου *υπερχείλισης της μνήμης* (*buffer overflow*) και άλλες.

Οι τεχνικές στατικής ανάλυσης αποτελούν ένα γνωστό όπλο υπέρ των διαδικασιών Δοκιμής Διείσδυσης και εύρεσης αδυναμιών σε προγράμματα και συστήματα γιατί παρέχουν απτά και αξιοποιήσιμα αποτελέσματα αυτόματα, χωρίς να χρειάζεται κανονική εκτέλεση της εφαρμογής από κάποιο αναλυτή. Αντίθετα, όλη η λειτουργία και η διαδικασία εκτελείται δίχως την ανάγκη για μεταγλώττιση και εκτέλεση των προγραμμάτων υπό έλεγχο. Επιπλέον, οι υπολογισμοί που κάνουν τα εργαλεία στατικής ανάλυσης ως προς την ανάλυση της λειτουργίας του προγράμματος, συχνά βοηθούν στην επίτευξη χαμηλών ποσοστών από false negatives αποτελέσματα. Ένα σημαντικό κομμάτι των σημερινών ερευνών έχει εστιάσει στην ανάλυση προγραμμάτων τα οποία είναι γραμμένα σε γλώσσες υψηλού επιπέδου, όπως Java, C και PHP με χρήση μεθόδων στατικής ανάλυσης. Οι αναλυτές λογισμικού έχουν αναπτύξει εργαλεία στατικής ανάλυσης για έλεγχο πηγαίου κώδικα προκειμένου να φιλτράρουν αρχεία κώδικα, ψάχνοντας για αδυναμίες ασφάλειας. Έρευνες του χώρου όπως αυτή των Halfond, Choudhary και Orso κάνουν χρήση των *διανυσμάτων εισόδου δεδομένων (input vectors)* στο πηγαίο κώδικα διαδικτυακών εφαρμογών προκειμένου να εκτελέσουν στατικό έλεγχο εύρεσης αδυναμιών με ευριστικό τρόπο [15]. Άλλες έρευνες όπως αυτή των Tevis και Hamilton Jr ανέπτυξαν εργαλεία στατικού ελέγχου προκειμένου να ελέγξουν εκτελέσιμα αρχεία εφαρμογών (*executables*) δίχως να έχουν πρόσβαση στο πηγαίο κώδικά τους [8].

5.2 ΜΕΘΟΔΟΙ ΣΤΑΤΙΚΗΣ ΑΝΑΛΥΣΗΣ

Οι πρακτικές μέθοδοι χρήσης στατικής ανάλυσης για εντοπισμό προβλημάτων ασφαλείας υλοποιούνται στα ευρύτερα πλαίσια του *Ελέγχου Λογισμικού* (*Software Testing*) και αξιοποιούνται ταυτόχρονα και στο τομέα των Δοκιμών Διείσδυσης και εύρεσης αδυναμιών. Δύο από τις βασικότερες μεθόδους που θα μας απασχολήσουν και κάνουν χρήση στατικής ανάλυσης, είναι η *Διασπορά Μολυσμένων αντικειμένων* (*Tainted Object Propagation*) και ο *Έλεγχος Μοντέλων Κατάστασης* (*State Model Checking*). Τμήματα των δύο αυτών θεωριών χρησιμοποιούνται και στην ανάπτυξη της μεθόδου εύρεσης λογικών αδυναμιών του LogGIC που υλοποιείται στη παρούσα διπλωματική.

5.2.1 ΔΙΑΣΠΟΡΑ ΜΟΛΥΣΜΕΝΩΝ ΑΝΤΙΚΕΙΜΕΝΩΝ (TAINTED OBJECT PROPAGATION)

Η Διασπορά Μολυσμένων Αντικειμένων χρησιμοποιείται για τον στατικό προσδιορισμό της χρήσης ανεπιθύμητων δεδομένων σε σημαντικές λειτουργίες της εφαρμογής. Μία κατάσταση μολυσμένου αντικειμένου συμπεριλαμβάνει μια ομάδα από περιγραφείς πηγής (source descriptors), περιγραφείς δεξαμενής (sink descriptors) και περιγραφείς απόρροιας (derivation descriptors). Προκειμένου να βρεθούν αδυναμίες και πιθανοί τρόποι παραβίασης του συστήματος, είναι αναγκαίο να είναι γνωστά τα αντικείμενα στα οποία αναφέρονται οι περιγραφείς μέσα στο κώδικα. Αυτό το πρόβλημα συναντάται συχνά με την ονομασία Pointer ή Point-to πρόβλημα. Στην έρευνά τους, οι Livshits και Lam του πανεπιστημίου του Stanford χρησιμοποιούν στατική ανάλυση για να βρουν όλα τα πιθανά *αποτυπώματα παραβιάσεων* (*violation fingerprints*), δεδομένων των πηγών και των προορισμών της μολυσμένης πληροφορίας εισόδου [10].

- Οι *περιγραφείς πηγής* (*source descriptors*) προσδιορίζουν τρόπους με τους οποίους ένας χρήστης μπορεί να παρέχει πληροφορία σε ένα σύστημα. Οι πηγές ανεπιθύμητων δεδομένων αναγνωρίζονται με ποικίλους τρόπους, όπως με παρακολούθηση των παραμέτρων που εισάγονται σε μια εφαρμογή είτε από γραφικό περιβάλλον, είτε από κονσόλα εντολών (command prompt).
- Οι *περιγραφείς δεξαμενής* (*sink descriptors*) προσδιορίζουν τρόπους με τους οποίους τα δεδομένα μπορεί να χρησιμοποιηθούν από το πρόγραμμα ή αλλιώς, που αποθηκεύεται προσωρινά η πληροφορία μέσα στο κώδικα μέχρι ότου προχωρήσει μαζί με την ροή της πληροφορίας κατά την εκτέλεση της εφαρμογής.
- Οι *περιγραφείς απόρροιας* (*derivation descriptors*) προσδιορίζουν πώς τα δεδομένα διαδίδονται ανάμεσα σε αντικείμενα του συστήματος. Ένας περιγραφέας απόρροιας προσδιορίζει ότι μία τυχόν κλήση σε μία μέθοδο μπορεί να μεταδώσει μία μολυσμένη πληροφορία από το ένα αντικείμενο κώδικα σε αυτό που καλείται με χρήση της μολυσμένης πληροφορίας.

5.2.2 ΕΛΕΓΧΟΣ ΜΟΝΤΕΛΩΝ ΚΑΤΑΣΤΑΣΗΣ (STATE MODEL CHECKING)

Στην Πληροφορική και στην επιστήμη των υπολογιστών γενικότερα, ο όρος Έλεγχος Μοντέλων (model checking) αναφέρεται στη διαδικασία κατά την οποία, δοσμένου ενός μοντέλου καταστάσεων λειτουργίας μιας εφαρμογής, ελέγχεται αν το παρεχόμενο μοντέλο ανταποκρίνεται στους περιορισμούς και τις προδιαγραφές εκτέλεσης που είχαν οι δημιουργοί για την εφαρμογή τους.

Έλεγχοι Μοντέλων Κατάστασης χρησιμοποιούνται τόσο σε επίπεδο hardware υλοποιήσεων όσο και σε software. Οι περιορισμοί και οι προδιαγραφές κινούνται συνήθως σε ένα ευρύ πλαίσιο που ενσωματώνει από την ανάγκη έλλειψης αδιεξόδων κατά την εκτέλεση της εφαρμογής, μέχρι την εύρεση κρίσιμων καταστάσεων και επικίνδυνων μονοπατιών εκτέλεσης του προγράμματος που μπορεί να οδηγήσουν σε κατάρρευση του λογισμικού υπό μελέτη. Με άλλα λόγια, ο Έλεγχος Μοντέλων Καταστάσεων αποτελεί μια τεχνική για αυτόματη επαλήθευση της ορθής λειτουργίας εφαρμογών σε συστήματα με πεπερασμένο αριθμό πιθανών καταστάσεων εκτέλεσης.

Αξίζει να αναφερθεί το συνηθέστερο πρόβλημα των υλοποιήσεων που χρησιμοποιούν έλεγχο Μοντέλων Κατάστασης, που ονομάζεται συνδυαστική έκρηξη του χώρου καταστάσεων (combinational blow up of the state-space ή αλλιώς state explosion problem). Αυτό το πρόβλημα δημιουργείται από την εκθετική αύξηση των καταστάσεων που πρέπει να αποθηκευτούν και να ελεγχθούν κατά την διαδικασία ελέγχου, η οποία χειροτερεύει όσο αυξάνεται το μέγεθος και οι λειτουργίες της εφαρμογής. Ο Ιωσήφ Σηφάκης ήταν ο Έλληνας ερευνητής στον οποίο απονεμήθηκε το βραβείο Turing για το 2007, μαζί με τον Έντμουντ Κλάρκ και τον Έρνεστ Άλλεν Έμερσον, για τις εργασίες τους στον έλεγχο μοντέλων (model checking) και στο ανωτέρω πρόβλημα της συνδυαστικής έκρηξης του χώρου καταστάσεων που αναφέρθηκε.

5.3 ΔΥΝΑΜΙΚΗ ΑΝΑΛΥΣΗ

Οι μέθοδοι *Δυναμικής Ανάλυσης* (*Dynamic analysis tools*) παρατηρούν ένα σύστημα κατά τη διάρκεια που αυτό εκτελείται.Σε μερικές υλοποιήσεις του, παρέχουν επιπρόσθετα στο σύστημα λανθασμένες, κακόβουλες ή τυχαίες πληροφορίες υπό τη μορφή εισόδου προσπαθώντας έτσι να φέρουν στο φως λάθη και αδυναμίες. Όπως παρουσιάστηκε και νωρίτερα, αυτή η τεχνική αυτή είναι ευρέως γνωστή και ως Fuzzing [3].

5.4 ΜΕΘΟΔΟΙ ΔΥΝΑΜΙΚΗΣ ΑΝΑΛΥΣΗΣ

Οι μέθοδοι Δυναμικής Ανάλυσης μπορούν να κατακερματιστούν σε ακόμα δύο διακριτές υποκατηγορίες, ανάλογα με το τρόπο που διεξάγουν την ανάλυσή τους. Η προσέγγιση με βάση *την απτή δοκιμή του συστήματος* προσπαθεί να φέρει στο φως τυχόν αδυναμίες και ευπάθειες συστημάτων με το να τροφοδοτεί το πρόγραμμα με διάφορες τυχαίες και κακόβουλες πληροφορίες εισόδου ενώ, η προσέγγιση *με βάση*

την παρακολούθηση του συστήματος εξετάζει την εκτέλεση της εφαρμογής κατά την διάρκεια της κανονικής της λειτουργίας, προσπαθώντας να εντοπίσει τυχόν ανώμαλη λειτουργία.

Ένας μεγάλος αριθμός αδυναμιών σε εφαρμογές δημιουργούνται λόγω της έλλειψης δυναμικών ελέγχων στα δεδομένα εισόδου, γεγονός που αποτελεί παράγοντα εμφάνισης επιθέσεων τύπου υπερχείλισης μνήμης (buffer overflow) και *διαχείρισης συμβολοσειράς* (*string manipulation attacks*). Διάφορες έρευνες όπως αυτή των Vigna και Zhou εφαρμόζουν μια τεχνική δυναμικής επανεγγραφής εντολών του προγράμματος υπό έλεγχο στο επίπεδο *εντολών συστήματος* (*system call interface*) και στις βιβλιοθήκες του προγράμματος προκειμένου να αναλύσουν κλήσεις συστήματος και εκτελέσεις εντολών της εφαρμογής. Η έρευνα αποσκοπεί στην εύρεση λογικών αδυναμιών στις εφαρμογές μέσω των προαναφερθέντων τεχνικών [11].

Στο κεφάλαιο που ακολουθεί γίνεται μια συστηματική παρουσίαση του σύγχρονου ερευνητικού χώρου της Ασφάλειας που ασχολείται με την εύρεση αδυναμιών σε εφαρμογές. Παρουσιάζεται μια πληθώρα δημοσιεύσεων, άλλες εκτενώς και άλλες περιληπτικά, ανάλογα με το βαθμό σημαντικότητάς τους αλλά και επιρροής τους στην διαδικασία έρευνας της παρούσας διπλωματικής.

6. ΤΕΛΕΥΤΑΙΕΣ ΕΞΕΛΙΞΕΙΣ ΣΤΟ ΧΩΡΟ ΤΩΝ ΔΟΚΙΜΩΝ ΔΙΕΙΣΔΥΣΗΣ ΚΑΙ ΕΥΡΕΣΗΣ ΑΔΥΝΑΜΙΩΝ (STATE OF THE ART REVIEW)

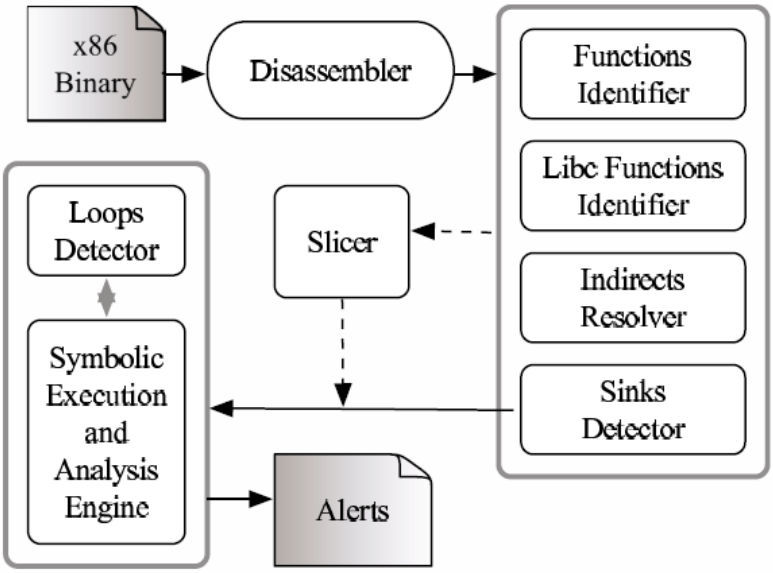
Ακολουθεί η σύντομη παρουσίαση κάποιων από τις πιο ενδιαφέρουσες εξελίξεις στο χώρο των δοκιμών διείσδυσης και εύρεσης αδυναμιών σε εφαρμογές, οι οποίες χαρτογραφούν ταυτόχρονα και την πορεία προς την τελική επιλογή των τεχνικών για την ανάπτυξη του εργαλείου και της μεθόδου που παρουσιάζονται στο πλαίσιο αυτής της διπλωματικής.

6.1 ΣΤΑΤΙΚΟΣ ΕΝΤΟΠΙΣΜΟΣ ΑΔΥΝΑΜΙΩΝ ΣΕ X86 ΕΚΤΕΛΕΣΙΜΑ ΑΡΧΕΙΑ

Η πρώτη προσπάθεια προσέγγισης μιας μεθόδου εξεύρεσης αδυναμιών έγινε μελετώντας κάποιες έρευνες που μελετούσαν τις μεθόδους ελέγχου εκτελέσιμων αρχείων χωρίς πρόσβαση στο πηγαίο τους κώδικα ή σε πληροφορίες ανάπτυξης. Οι διαδικασίες αυτές πλησιάζουν την λογική των δοκιμών black box αν και προϋποθέτουν κάποιες βασικές πληροφορίες προκειμένου να μπορούν να λειτουργήσουν (σε αντίθεση με την βασικό, θεωρητικό ορισμό του black-box). Μία έρευνα που χαρακτηρίζει το χώρο ως προς την προσπάθεια εύρεσης αδυναμιών σε binary εκτελέσιμα αρχεία με χρήση τεχνικών στατικής ανάλυσης, είναι η έρευνα των Cova, Felmetsger, Banks και Vigna, "*Static Detection of Vulnerabilities in x86 Executables*" [12].

Η έρευνα επικεντρώνεται σε εκτελέσιμα αρχεία που εκτελούνται σε επεξεργαστές της οικογένειας Intel x86 γιατί αποτελούν την πιο διαδεδομένη αρχιτεκτονική εκτελέσιμων εφαρμογών που χρησιμοποιείται αυτή τη στιγμή παγκοσμίως. Ο βασικός της στόχος είναι η εξεύρεση ευπαθειών στο λογισμικό που ενεργοποιούνται από την είσοδο κακόβουλων πληροφοριών στην εφαρμογή από μη-έμπιστες πηγές (tainted data).

Η ανάλυση της συγκεκριμένης έρευνας χωρίζεται σε δύο φάσεις: Αρχικά γίνεται χρήση διάφορων τεχνικών προκειμένου να προσεγγιστεί η κατάσταση εκτέλεσης (state) της εφαρμογής κατά την εκτέλεσή της. Στη συνέχεια, γίνεται ανάλυση των πληροφοριών προκειμένου να εντοπιστούν ευπάθειες.



ΕΙΚΟΝΑ 2: Η ΑΡΧΙΤΕΚΤΟΝΙΚΗ ΤΟΥ ΕΡΓΑΛΕΙΟΥ ΕΥΡΕΣΗΣ ΑΔΥΝΑΜΙΩΝ ΤΩΝ COVA, FELMETSGER, BANKS ΚΑΙ VIGNA.

Αρχικά, με χρήση αλγορίθμου *γραμμικής σάρωσης* (*linear-sweep algorithm*) γίνεται *αποσυναρμολόγηση σε γλώσσα μηχανής* (*disassembly*) του αρχείου και παράγεται ο κώδικας μηχανής του. Στη συνέχεια γίνεται καταγραφή και ανάλυση των κλήσεων συστήματος (system calls) και χαρτογράφηση των σημείων μετάβασης στον κώδικα (jumps), καθώς και η αναγνώριση των διάφορων μεθόδων λειτουργίας. Αυτές οι πληροφορίες οδηγούν στη δημιουργία ενός γράφου ελέγχου εντολών και ενός *γράφου ροής εκτελέσιμων εντολών* (*Control Flow Graph*). Στη συνέχεια αναγνωρίζονται οι επαναληπτικές διαδικασίες-βρόχοι εκτέλεσης στον κώδικα (loops). Η διαδικασία ανάλυσης ολοκληρώνεται με την εύρεση των μεθόδων στις βιβλιοθήκες κώδικα (library functions used).

Η δεύτερη φάση της ανάλυσης ακολουθεί την τυπική και ευρέως διαδεδομένη μέθοδο ανάλυσης μολυσμένων δεδομένων εισόδου (5.2.1 ΔΙΑΣΠΟΡΑ ΜΟΛΥΣΜΕΝΩΝ ΑΝΤΙΚΕΙΜΕΝΩΝ (TAINTED OBJECT PROPAGATION)). Πρακτικά, η έρευνα δείχνει με ποιο τρόπο, η ανάλυση μολυσμένων δεδομένων εισόδου μπορεί να χρησιμεύσει στον εντοπισμό αδυναμιών στο λογισμικό. Η προσέγγιση γίνεται διαχωρίζοντας τη προαναφερθείσα διαδικασία σε τέσσερα διακριτά μέρη:

- Αναγνώριση των πηγών κακόβουλων δεδομένων εισόδου όπως παράμετροι γραμμής εντολών, μεταβλητές συστήματος, δεδομένα εισόδου από αρχεία κτλ.
- Αναγνώριση «δεξαμενών» αποθήκευσης μολυσμένων δεδομένων όπως κλήσεις σε μεθόδους system( ), popen( ).
- Αναγνώριση μεταφοράς και διάδοσης των μολυσμένων δεδομένων εισόδου και

- Δημιουργία αναφοράς όταν τα μολυσμένα δεδομένα φτάσουν σε κάποια κρίσιμη δεξαμενή που σηματοδοτεί την ύπαρξη ευπάθειας στο λογισμικό.

Η ανάλυση βασίζεται στη *συμβολική εκτέλεση* του προγράμματος υπό έλεγχο. Η συμβολική εκτέλεση είναι η διαδικασία κατά την οποία ένα πρόγραμμα εκτελείται σε ελεγχόμενο περιβάλλον και στο οποίο παρέχονται ελεγχόμενα δεδομένα εισόδου αναδρομικά, αντί για τα τυπικά και αναμενόμενα δεδομένα εισόδου.

Η ανωτέρω μέθοδος που αναπτύχθηκε αποτελεί έναν εύρωστο -αν και σχετικά εξειδικευμένο- τρόπο εντοπισμού αδυναμιών σε μια εφαρμογή, κάνοντας χρήση μιας θεωρητικής μεθόδου που πλησιάζει την λογική των δοκιμών black-box, αφού δεν παρέχονται πληροφορίες για το λογισμικό, πέραν από αυτές που παράγονται με την αποσυναρμολόγηση ενός αγνώστου εκτελέσιμου αρχείου. Τα πειράματα που έγιναν στα πλαίσια της έρευνας, προκειμένου να αξιολογηθεί η ικανότητα της μεθόδου στον εντοπισμό αδυναμιών, έδειξαν πως η εφαρμογή βρήκε σχεδόν όλες τις αδυναμίες που υπήρχαν, αφού παρουσίασε πολύ χαμηλό ποσοστό, σχεδόν 0%, στην αποτυχημένη εξεύρεση υπαρχουσών αδυναμιών (*false negatives*).

Όσων αφορά την έρευνα της παρούσας διπλωματικής, η μέθοδος αυτή δεν αποτελεί μια εύχρηστη οδό προσέγγισης ενός τρόπου για εντοπισμό λογικών αδυναμιών, καθώς στηρίζεται ιδιαίτερα στη τεχνική ανάλυση του παραγόμενου κώδικα μηχανής και οδηγεί αποκλειστικά στον εντοπισμό αδυναμιών που σχετίζονται με την έγχυση κακόβουλων πληροφοριών για εκμετάλλευση αδυναμιών. Ο εντοπισμός λογικών αδυναμιών χρειάζεται μια πιο θεωρητική προσέγγιση των διαδικασιών της εφαρμογής, καθώς και προσδιορισμό του σκοπού της εφαρμογής και του πώς αυτός υλοποιείται σε επίπεδο λογικών βημάτων. Πρέπει, με κάποιο τρόπο, να εκμαιεύεται η λογική της λειτουργίας της εφαρμογής και να περιγράφεται έπειτα υπό μορφή περιορισμών. Παραδείγματος χάριν, αν η λογική μιας λειτουργίας είναι να είναι υπεύθυνη για καταχώρηση άρθρων σε μια εφαρμογή, αυτό δρα ως περιοριστικός παράγοντας απέναντι στην ικανότητά της να καταχωρεί και άλλο είδος αρχείων. Η λογική της λειτουργίας δρα ως περιορισμός απέναντι σε άλλες πιθανές λειτουργίες που μπορεί να διεκπεραιώνει. Ο προσδιορισμός της λογικής μιας εφαρμογής μέσα από κώδικα μηχανής και την θεωρία Μολυσμένων Αντικειμένων είναι ελλιπής, λόγω του ότι επικεντρώνεται στα διανύσματα εισόδου και στη μεταφορά δεδομένων εισόδου. Επιπλέον, ο κώδικας μηχανής είναι δύσχρηστος για μελέτη των λογικών διαδικασιών και στόχων των λειτουργιών μιας εφαρμογής.

Συνεπώς, η συγκεκριμένη μέθοδος black-box ανάλυσης και εύρεσης αδυναμιών δεν αξιοποιήθηκε κατά την ανάπτυξη της μεθόδου που θα παρουσιαστεί στα πλαίσια της παρούσας διπλωματικής.

6.2 ΔΟΚΙΜΕΣ ΑΣΑΦΕΙΑΣ ΓΙΑ ΕΛΕΓΧΟ ΑΣΦΑΛΕΙΑΣ ΚΑΙ ΠΟΙΟΤΗΤΑΣ ΛΟΓΙΣΜΙΚΟΥ (TAKANEN, DEMOTT, & MILLER, 2008)

Διάφορες προσεγγίσεις του χώρου της ασφάλειας ασχολούνται με τις δοκιμές ασάφειας. Η ακόλουθη ανάλυση των Takanen, DeMott και Miller συνοψίζει την βασική δομή και τον τρόπο λειτουργίας αυτών των δοκιμών ασάφειας, και προσδιορίζει τους τρεις διαφορετικούς στόχους αυτών των δοκιμών:

- Την εξασφάλιση της ποιότητας του λογισμικού ως προϊόν.
- Την εξασφάλιση όσο το δυνατόν καλύτερης ασφάλειας του λογισμικού που υλοποιείται σε κρίσιμες θέσεις-κλειδιά ενός πληροφοριακού συστήματος.
- Την αξιολόγηση του κατά πόσον είναι εύκολο να παραβιάσει ένας κακόβουλος χρήστης την ασφάλεια ενός λογισμικού ή ενός πληροφοριακού συστήματος.

Το συγκεκριμένο βιβλίο ασχολείται με την ανάλυση των εξ' ολοκλήρου αυτόματων μεθόδων δοκιμών, μέθοδοι οι οποίες χρησιμοποιούν διάφορες πιθανές, ακραίες τιμές ως λανθασμένα δεδομένα εισόδου στις εφαρμογές υπό έλεγχο (βλ. 2.1.3), προκειμένου να εντοπίσουν αδυναμίες που μπορούν να εκμεταλλευθούν κακόβουλοι χρήστες. Σε αυτή την ανάλυση, ο κύριος στόχος των μελετητών είναι η παρουσίαση τεχνικών τύπου black-box ή grey-box δοκιμών δηλαδή, η εισαγωγή συγκεκριμένων δεδομένων εισόδου μέσω διαφορετικών διεπαφών και η μελέτη των εξόδων του συστήματος, δίχως γνώση για την εσωτερική του λειτουργία. Η θεωρία που παρουσιάζεται στο συγκεκριμένο βιβλίο συνοψίζει την θεωρητική βάση της επόμενης έρευνας που παρουσιάζεται και ταυτόχρονα αποτελεί και την βάση ενός τμήματος της μεθόδου που υλοποιήθηκε σε αυτή την διπλωματική.

6.3 ΔΟΚΙΜΕΣ ΔΙΕΙΣΔΥΣΗΣ ΜΕ ΒΕΛΤΙΩΜΕΝΗ ΑΝΑΓΝΩΡΙΣΗ ΔΙΑΝΥΣΜΑΤΩΝ ΕΙΣΟΔΟΥ (WILLIAM G.J. HALFOND, SHAUVIK ROY CHOUDHARY, AND ALESSANDRO ORSO)

Η δεύτερη ερευνητική προσέγγιση που παρουσιάζεται αφορά την ανάπτυξη μιας καινούργιας προσέγγισης των δοκιμών διείσδυσης [15], η οποία στοχεύει στην βελτίωση των δοκιμών διείσδυσης στις διαδικτυακές εφαρμογές. Προκειμένου να πετύχει τον στόχο της, η έρευνα επικεντρώνεται σε δύο περιοχές έρευνας στις οποίες οι τεχνικές που υπάρχουν είναι περιορισμένης χρησιμότητας: Την αναγνώριση των διανυσμάτων εισόδου δεδομένων σε μια διαδικτυακή εφαρμογή και στον εντοπισμό

του αποτελέσματος από την χρήση μολυσμένων δεδομένων ως είσοδο στην εφαρμογή.

Μία πιθανή μορφή επίθεσης με μολυσμένα δεδομένα εισόδου θα ήταν αν ο κακόβουλος χρήστης ήθελε να αυθεντικοποιηθεί σε μια εφαρμογή και αντί για το όνομα χρήστη του να έβαζε την συμβολοσειρά

<ΟΝΟΜΑ ΧΡΗΣΤΗ> ; drop table UserTable --

σαν δεδομένο εισόδου. Αυτή η ενέργεια είναι πιθανό να οδηγούσε στην εκτέλεση της εντολής SQL που ακολουθεί μετά το όνομα χρήστη και η οποία θα οδηγούσε στην διαγραφή ενός πίνακα πληροφοριών από την βάση δεδομένων που χρησιμοποιεί η λειτουργία αυθεντικοποίησης για να εκτελέσει το στόχο της.

Η μέθοδος που αναπτύχθηκε στο πλαίσιο της έρευνάς τους, χρησιμοποιεί δύο σύγχρονες τεχνικές ανάλυσης. Η πρώτη είναι μια μέθοδος στατικής ανάλυσης κώδικα για τον εντοπισμό και την αναγνώριση πιθανών διανυσμάτων εισόδου δεδομένων. Η δεύτερη είναι μια δυναμική ανάλυση που αυτοματοποιεί την ανάλυση δεδομένων εξόδου της εφαρμογής υπό έλεγχο.

6.3.1 ΣΤΑΔΙΟ ΣΥΛΛΟΓΗΣ ΠΛΗΡΟΦΟΡΙΩΝ

Η μέθοδος στατικής ανάλυσης χρησιμοποιείται προκειμένου να αναλυθεί ο κώδικας της εφαρμογής υπό έλεγχο και να εντοπιστούν οι θέσεις των διανυσμάτων εισόδου, ο τρόπος που αυτά ομαδοποιούνται (δηλαδή, ποιές ομάδες διανυσμάτων προσπελαύνονται ταυτόχρονα από μία λειτουργία της εφαρμογής), και ποιές είναι οι πληροφορίες τομέα των διανυσμάτων (δηλαδή, οι τιμές και οι περιορισμοί που μπορούν να προσδιοριστούν από οποιονδήποτε χρήστη της εφαρμογής). Η συλλογή ορθών και ποιοτικών πληροφοριών κατά το στάδιο συλλογής πληροφοριών της μεθόδου, αποτελεί μία λειτουργία ζωτικής σημασίας για την ποιότητα των αποτελεσμάτων της τεχνικής. Όσο καλύτερες πληροφορίες συλλεχθούν για μια εφαρμογή, τόσο πιο αποτελεσματικές θα είναι οι δοκιμές που θα υλοποιηθούν έπειτα και τόσο πιο εύστοχα θα είναι τα συνολικά αποτελέσματα.

Στο στάδιο της στατικής ανάλυσης, η μέθοδος που αναπτύχθηκε είναι βασισμένη στη τεχνική WAM (πλήρες όνομα τεχνικής, WAM), η οποία αναπτύχθηκε από δύο άτομα της ομάδας που υλοποίησε και την παρούσα έρευνα που παρουσιάζεται σε αυτό το κεφάλαιο. Η WAM αναλύει τον κώδικα μιας διαδικτυακής εφαρμογής σε δύο φάσεις και παράγει μια λίστα από όλα τα διανύσματα εισόδου στην εφαρμογή, μαζί με τις αντίστοιχες πληροφορίες ομαδοποίησης και περιορισμών για

το καθένα. Προκειμένου να αναγνωρίσει τα ονόματα των διανυσμάτων διείσδυσης, η WAM προσπαθεί να αναλύσει τις τιμές των παραμέτρων που χρησιμοποιούν την πληροφορία που προέρχεται από τα διανύσματα αυτά. Από την χρήση των μεταβλητών που γίνονται αποδέκτες των δεδομένων εισόδου, η WAM καταλήγει σε ομαδοποίηση των διανυσμάτων. Παραδείγματος χάρη, σε ένα σύστημα αυθεντικοποίησης χρήστη, η WAM ομαδοποίησε τις ακόλουθες μεταβλητές με τον παρακάτω τρόπο: {“userAction”}, {“userAction,” “login,” “password”} και {“userAction,” “login,” “address”}.

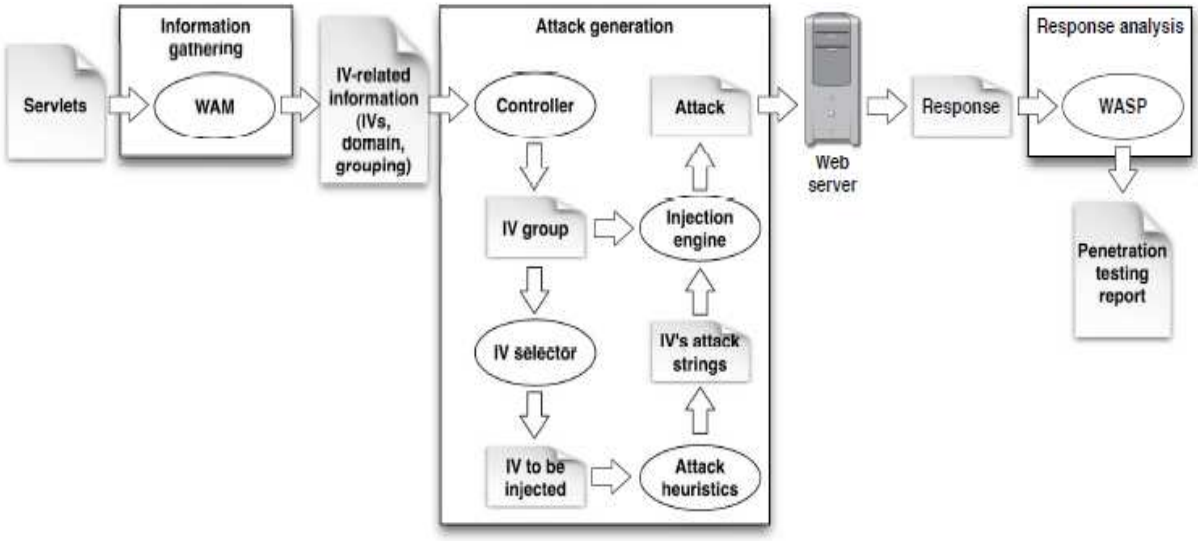
6.3.2 ΦΑΣΗ ΔΗΜΙΟΥΡΓΙΑΣ ΔΕΔΟΜΕΝΩΝ

Στην φάση δημιουργίας πληροφοριών εισόδου από την μέθοδο προκειμένου να αξιολογηθεί η εφαρμογή υπό έλεγχο, η τεχνική αυτής της έρευνας επικεντρώνεται στα αναγνωρισμένα διανύσματα εισόδου και χρησιμοποιεί τις πληροφορίες που ανέκτησε στο προηγούμενο στάδιο προκειμένου να παράγει ρεαλιστικά και χρήσιμα δεδομένα εισόδου για τις *περιπτώσεις δοκιμής* που θα ελεγχθούν (*Test cases*).

Όπως και το στάδιο συλλογής πληροφοριών, το στάδιο ανάλυσης των αποτελεσμάτων προκειμένου να δειχθεί αν μια περίπτωση δοκιμής έφερε στην επιφάνεια κάποια αδυναμία είναι εξίσου σημαντικό. Η έρευνα των William G.J. Halfond, Shauvik Roy Choudhary, and Alessandro Orso προτείνει λύσεις στο προαναφερθέν ζήτημα. Η μέθοδος της έρευνας υπό παρουσίαση αναγνωρίζει αυτόματα τις πιθανές επιθέσεις *έγχυσης πληροφορίας* (*injection attacks*) στα διανύσματα εισόδου κάνοντας χρήση της δυναμικής παρακολούθησης μολυσμένων αντικειμένων (5.2.1 ΔΙΑΣΠΟΡΑ ΜΟΛΥΣΜΕΝΩΝ ΑΝΤΙΚΕΙΜΕΝΩΝ (TAINTED OBJECT PROPAGATION)) και επιτρέπει στο εργαλείο που αναπτύχθηκε να εκτελέσει ένα πλήρως αυτοματοποιημένο εντοπισμό των επιτυχών επιθέσεων που δοκιμάστηκαν και των αδυναμιών που βρέθηκαν.

Ακολουθώντας το προηγούμενο παράδειγμα ομαδοποίησης διανυσμάτων του κεφαλαίου (6.3.1 ΣΤΑΔΙΟ ΣΥΛΛΟΓΗΣ ΠΛΗΡΟΦΟΡΙΩΝ), ακολουθεί ένα παράδειγμα της φάσης δημιουργίας δεδομένων με χρήση της WAM, προκειμένου να γίνει καλύτερα κατανοητή η διαδικασία:

Έχοντας την προηγούμενη ομαδοποίηση διανυσμάτων {“login”, “password”,“userAction”}, κατά την διάρκεια δημιουργίας περιπτώσεων χρήσης για επίθεση στο σύστημα, τα εργαλεία θα στοχεύσουν καθένα από τα διανύσματα αυτά με πιθανά δεδομένα τα οποία δημιουργήσανε με βάση κάποιους ευριστικούς αλγορίθμους. Όταν το πρώτο διάνυσμα “login” εξετασθεί, η μέθοδος θα δημιουργήσει διάφορες συμβολοσειρές και θα τις χρησιμοποιήσει σαν τιμή για το διάνυσμα αυτό. Η προσέγγιση που παρουσιάζεται σε αυτή την έρευνα χρησιμοποιεί



τις πληροφορίες ομαδοποίησης και περιορισμών που εξήγαγε στο προηγούμενο στάδιο. Αυτό το γεγονός θα οδηγήσει στη δημιουργία ορθών, αλφαριθμητικών τιμών για τα υπόλοιπα τρία διανύσματα. Η χρήση των πληροφοριών που συλλέχθηκαν επιτρέπει στη δοκιμή διείσδυσης να ξεπεράσει τυχόν προβλήματα ασυμβατότητας των πληροφοριών που παράγει με τις μεταβλητές που διαχειρίζονται τα δεδομένα εισόδου. Αυτό έχει άμεσο αποτέλεσμα στον καλύτερο εντοπισμό αδυναμιών.

6.3.3 SDAPT

Το εργαλείο που αναπτύχθηκε στα πλαίσια της έρευνας που παρουσιάζεται σε αυτό το κεφάλαιο ονομάζεται SDAPT [15], το οποίο χρησιμοποιήθηκε προκειμένου να αξιολογηθούν εννέα διαδικτυακές εφαρμογές με πολύ θετικά αποτελέσματα.

Η έρευνα του ινστιτούτου τεχνολογίας της Γεωργίας προσέφερε μια καινούργια προσέγγιση στις δοκιμές διείσδυσης και έκανε μια πρωτότυπη χρήση των τεχνικών ανάλυσης μολυσμένων αντικειμένων από τα διανύσματα εισόδου. Συγκεκριμένα χαρακτηριστικά της, όπως η διαδικασία εντοπισμού διανυσμάτων διείσδυσης, χρησιμοποιήθηκαν και στην υλοποίηση της παρούσας διπλωματικής για τον εντοπισμό πιθανών αδυναμιών διαχείρισης των δεδομένων εισόδου στις εφαρμογές με γραφική διεπαφή.

6.4 ΕΝΤΟΠΙΣΜΟΣ ΕΠΙΘΕΣΕΩΝ ΠΟΥ ΑΞΙΟΠΟΙΟΥΝ ΤΑ ΛΟΓΙΚΑ ΛΑΘΗ ΤΩΝ ΕΦΑΡΜΟΓΩΝ (JINGYU ZHOU AND GIOVANNI VIGNA)

Τα λογικά λάθη συμβαίνουν όταν μια εφαρμογή συμπεριφέρεται με τρόπο ο οποίος δεν ήταν αρχικά στους σκοπούς του δημιουργού της. Η έρευνα που

παρουσιάζεται σε αυτό το κεφάλαιο [16] είναι αποτέλεσμα της δουλειάς –μεταξύ άλλων- της ομάδας ασφάλειας με μέλη τους Zhou και Vigna [2][11][12]. Η τεχνική της έρευνας χρησιμοποιεί *δυαδική επανεγγραφή εντολών* (*binary rewriting*) προκειμένου να ελέγχει την ροή λειτουργίας των εφαρμογών υπό έλεγχο και να εξάγει ορθές και ολοκληρωμένες πληροφορίες. Αυτές τις πληροφορίες τις συνδυάζει έπειτα με συγκεκριμένα *αποτυπώματα* αδυναμιών (*signatures*). Τα αποτυπώματα αδυναμιών είναι ουσιαστικά δομές που αντικατοπτρίζουν με πληρότητα γνωστές λογικές αδυναμίες.

Προκειμένου να μπορεί κανείς να εντοπίσει λογικές αδυναμίες σε μια εφαρμογή, ένας υποσχόμενος τρόπος είναι ο έλεγχος συγκεκριμένων σημείων στην διαδικασία εκτέλεσης και στη ροή ελέγχου της εφαρμογής υπό έλεγχο. Η μέθοδος της παρούσας έρευνας είναι διαλειτουργική και υποστηρίζει μια πληθώρα αποτυπωμάτων λογικών αδυναμιών. Μπορεί επίσης να συλλέγει δυναμικά πληροφορίες για την ροή εκτέλεσης και ελέγχου της εφαρμογής από οποιοδήποτε σημείο της ενώ αυτή εκτελείται. Στις επόμενες παραγράφους αναλύεται διεξοδικά η εν λόγω προσέγγιση.

6.4.1 ΔΥΑΔΙΚΗ ΕΠΑΝΕΓΓΡΑΦΗ (BINARY REWRITING)

Η δυαδική επανεγγραφή είναι μια τεχνική που εφαρμόζεται σε εκτελέσιμα αρχεία (και όχι σε πηγαίο κώδικα), η οποία είναι ικανή να αλλάξει άμεσα τον δυαδικό κώδικα των εκτελέσιμων αρχείων, είτε μέσα στο σύστημα αρχείων, είτε όταν αυτά φορτώνονται στη μνήμη του συστήματος για να εκτελεστούν. Η επανεγγραφή δουλεύει τόσο για στατικά προγράμματα όσο και για προγράμματα που διαχειρίζονται δυναμικά βιβλιοθήκες και πληροφορίες. Παρέχει πρόσβαση στα εσωτερικά της εφαρμογής υπό έλεγχο καθώς και στις βιβλιοθήκες της εφαρμογής.

Υπάρχουν δύο είδη δυαδικής επανεγγραφής: Η *στατική επανεγγραφή* (*static rewriting*) και η *δυναμική επανεγγραφή* (*dynamic rewriting*). Η στατική επανεγγραφή τροποποιεί την εικόνα του προγράμματος μέσα στο σύστημα αρχείων του σκληρού δίσκου, ενώ η δυναμική επανεγγραφή τροποποιεί την περιοχή της μνήμης στην οποία αποθηκεύεται προσωρινά μια εφαρμογή όταν εκτελείται. Η μέθοδος που ανέπτυξε η επιστημονική ομάδα χρησιμοποιεί την ελεύθερη διεπαφή προγραμματισμού με το όνομα Dynist [25]. Το Dynist αποτελεί ένα εργαλείο δυναμικής επανεγγραφής το οποίο διατίθεται ελεύθερα στο διαδίκτυο υπό την μορφή βιβλιοθηκών.

Η δυναμική επανεγγραφή διαχωρίζει τις ρουτίνες και τις διαδικασίες ελέγχου από την ίδια την εφαρμογή υπό έλεγχο, σε αντίθεση με άλλες τεχνικές που ενσωματώνουν τους διάφορους ελέγχους μέσα στον πηγαίο κώδικα της εφαρμογής και μετά τον εκτελούν συνολικά. Με την δυναμική επανεγγραφή, προστίθεται μια *υπομονάδα λειτουργιών ελέγχου* (*audit module*) χωρίς την ανάγκη μεταγλώττισης του

πηγαίου κώδικα ξανά από την αρχή. Η υπομονάδα αυτή επιλέγεται όταν η εφαρμογή υπό έλεγχο βρίσκεται σε λειτουργία, επιτρέποντας έτσι σχετική ευελιξία των παραμέτρων που χρειάζονται κάθε φορά για τον έλεγχο μιας εφαρμογής. Επιπλέον, οι ρουτίνες ελέγχου τρέχουν στον ίδιο χώρο διευθύνσεων της μνήμης που λειτουργεί και η εφαρμογή υπό έλεγχο, χωρίς ιδιαίτερη επιβάρυνση σε υπολογιστικούς πόρους.

Το εργαλείο που ανέπτυξαν οι Zhou και Vigna κάνει χρήση μιας υπομονάδας που προσάπτεται στον πυρήνα λειτουργίας του συστήματος μέσα στο οποίο λειτουργεί η εφαρμογή (*kernel module*) καθώς και σε ένα *δαίμονα ελέγχου* (*user-space daemon*) για διαχείριση της εφαρμογής υπό έλεγχο. Ο σκοπός αυτών των εργαλείων είναι η παρακολούθηση της λειτουργίας του λειτουργικού συστήματος για γεγονότα που αναπαριστούν τις εκτελέσεις τις εφαρμογής υπό έλεγχο. Έπειτα, τα ίδια εργαλεία πραγματοποιούν έγχυση (injection) από ρουτίνες ελέγχου υπό την μορφή εντολών μέσα στην ροή εκτέλεσης της εφαρμογής κατά την έναρξη της λειτουργίας της. Το εργαλείο που υλοποιήθηκε στα πλαίσια της συγκεκριμένης έρευνας του Vigna και των συνεργατών του, λειτουργεί ομαλά με τον πυρήνα 2.4 του λειτουργικού Linux. Ο μηχανισμός δυναμικής επανεγγραφής υλοποιήθηκε με χρήση του περιβάλλοντος Dynist.

Η υπομονάδα λειτουργιών ελέγχου που προσάπτεται στον πυρήνα του λειτουργικού συστήματος επιτρέπει να γίνεται κλήση της λειτουργίας έναρξης μιας εφαρμογής κατά βούληση. Η υπομονάδα υλοποιεί μια τεχνική κατά την οποία διακόπτει την εκτέλεση της εντολής συστήματος execve( ). Έτσι, διακόπτεται η λειτουργία της εφαρμογής υπό έλεγχο ακριβώς πριν την εκτέλεση της πρώτης της εντολής, προκειμένου να εισαχθούν οι λειτουργίες ελέγχου λογικών αδυναμιών μέσα στο χώρο διευθύνσεων της μνήμης.

Η πραγματική διαχείριση της διεργασίας εύρεσης λογικών αδυναμιών γίνεται από το δαίμονα ελέγχου, ο οποίος είναι μια διεργασία με πλήρη πρόσβαση στους πόρους και τις λειτουργίες του συστήματος (*administrator process*). Για την ορθή λειτουργία του, ο δαίμονας χειρίζεται δύο αποθήκες δεδομένων ελέγχου: Την *αποθήκη ελέγχων* (*audit repository*) και την *αποθήκη διορθώσεων* (*patch repository*). Η αποθήκη διορθώσεων περιέχει τον κώδικα για διαχείριση των εφαρμογών υπό έλεγχο. Η αποθήκη διορθώσεων περιέχει τον κώδικα ελέγχων που θα προσαρτηθεί στις διαδικασίες της εφαρμογής υπό έλεγχο. Και στις δύο αποθήκες, ο κώδικας βρίσκεται υπό την μορφή δυναμικών βιβλιοθηκών.

6.4.2 ΓΛΩΣΣΑ ΠΡΟΣΔΙΟΡΙΣΜΟΥ ΕΛΕΓΧΩΝ

Το εργαλείο υποστηρίζει την δημιουργία ελέγχων, διορθώσεων και διαδικασιών εύρεσης λογικών αδυναμιών. Παρόλα αυτά, η διαδικασία ανάπτυξης

βιβλιοθηκών και κώδικα για την συγκεκριμένη δουλειά είναι πολύπλοκη και επιρρεπής σε λάθη. Γι αυτό το λόγο, οι ερευνητές της συγκεκριμένης ομάδας επιλέξανε να αναπτύξουν μια απλή γλώσσα προγραμματισμού προκειμένου να διευκολυνθεί η παραγωγή των ελέγχων, χωρίς να χρειάζεται βαθειά γνώση της δομής και της λειτουργίας του εργαλείου εύρεσης λογικών αδυναμιών. Μέσω της γλώσσας αυτής, παράγεται ένα κομμάτι κώδικα που εκτελεί μια συγκεκριμένη λειτουργία, το οποίο μεταφράζεται αυτόματα σε μια υπομονάδα ελέγχου και προστίθεται μέσα σε μια δυναμική βιβλιοθήκη που χρησιμοποιεί το εργαλείο.

Αυτή η έρευνα που παρουσιάστηκε σε συνδυασμό με τα εμπειρικά στοιχεία που παρουσιάζει και την αξιολόγηση του εργαλείου της, αποτελεί ένα αξιόλογο τρόπο εύρεσης λογικών αδυναμιών, εμφανίζοντας υψηλά ποσοστά επιτυχίας με μικρή εμφάνιση λανθασμένων αποτελεσμάτων. Παρόλα αυτά, η περιορισμένη της λειτουργία μόνο σε λειτουργικά συστήματα που φέρουν τον πυρήνα 2.4 του λειτουργικού Linux, σε συνδυασμό με την πολύπλοκη και εξαιρετικά δύσκολη υλοποίηση της θεωρητικής μεθόδου σε άλλα συστήματα (υλοποίηση η οποία προϋποθέτει την βαθιά γνώση του λειτουργικού συστήματος για το οποίο αναπτύσσεται) οδήγησαν στην επιλογή ενός εναλλακτικού τρόπου εύρεσης λογικών αδυναμιών. Στην μέθοδο που αναπτύχθηκε στα πλαίσια της παρούσας διπλωματικής, η ανωτέρω έρευνα δεν αξιοποιήθηκε.

6.5 ΕΝΑ ΠΡΩΤΟ ΒΗΜΑ ΠΡΟΣ ΤΟΝ ΑΥΤΟΜΑΤΟ ΕΝΤΟΠΙΣΜΟ ΑΔΥΝΑΜΙΩΝ ΥΠΕΡΧΕΙΛΙΣΗΣ ΜΝΗΜΗΣ (DAVID WAGNER JEFFREY S. FOSTER ERIC A. BREWER ALEXANDER AIKEN, UNIVERSITY OF CALIFORNIA, BERKELEY)

Η δημοσίευση αυτή περιγράφει μια τεχνική εύρεσης πιθανών αδυναμιών που οδηγούν σε υπερχείλιση της μνήμης μέσα στον πηγαίο κώδικα προγραμμάτων που αναπτύχθηκαν στη γλώσσα προγραμματισμού C. Θεμέλιο λίθο της έρευνας αποτελεί η χρήση στατικής ανάλυσης του κώδικα της εφαρμογής υπό έλεγχο.

Η έρευνα αυτή μοντελοποιεί την διαδικασία εύρεσης αδυναμιών υπερχείλισης κατασκευάζοντας μια μέθοδο ανάλυσης του εύρους των αριθμητικών τιμών των μεταβλητών ενός προγράμματος. Ένα βασικό πλεονέκτημα της στατικής ανάλυσης είναι ότι, τα διάφορα προβλήματα υλοποίησης σε επίπεδο κώδικα μπορούν να επιλυθούν πριν την εκτέλεση της εφαρμογής.

Η ομάδα υλοποίησε την μέθοδό της σε ένα πρωτότυπο εργαλείο (δεν παρέχεται συγκεκριμένο όνομα εργαλείου στις δημοσιεύσεις) και το χρησιμοποίησε

προκειμένου να εντοπίσει νέες αδυναμίες που μπορούν να αξιοποιηθούν από απομακρυσμένους χρήστες, μέσα σε μεγάλα πληροφοριακά συστήματα.

6.6 ΔΥΝΑΜΙΚΗ ΑΝΑΛΥΣΗ ΜΟΛΥΣΜΕΝΩΝ ΔΕΔΟΜΕΝΩΝ ΓΙΑ ΤΟΝ ΑΥΤΟΜΑΤΟ ΕΝΤΟΠΙΣΜΟ, ΑΝΑΛΥΣΗ ΚΑΙ ΚΑΤΑΓΡΑΦΗ ΑΔΥΝΑΜΙΩΝ ΣΕ ΕΜΠΟΡΙΚΟ ΛΟΓΙΣΜΙΚΟ (JAMES NEWSOME, DAWN SONG, JULY 2005, SCHOOL OF COMPUTER SCIENCE, CARNEGIE MELLON UNIVERSITY. PITTSBURGH)

Οι αδυναμίες λογισμικού αποτελούν σοβαρή απειλή για τους χρήστες του διαδικτύου αλλά και των εμπορικών δικτύων. Η ύπαρξη κακόβουλου λογισμικού με την μορφή ιών, σκουληκιών (worms) κ.ά. αποτελούν υπαρκτές απειλές, τόσο για την ορθή λειτουργία όσο και για την διατήρηση της εμπιστευτικότητας των πληροφοριών που είναι προσβάσιμες μέσω διαδικτύου.

Η έρευνα που παρουσιάζεται σε αυτή τη δημοσίευση, ασχολείται με τρόπους καταπολέμησης αυτού του είδους των κινδύνων που ελλοχεύουν στο διαδίκτυο, με την βοήθεια γρήγορων και αυτόματων μηχανισμών ανίχνευσης τέτοιων επιθέσεων. Η δημοσίευση προτείνει μια μέθοδο αντιμετώπισης τέτοιων απειλών η οποία δεν έχει ανάγκη τον πηγαίο κώδικα των κακόβουλων λογισμικών προκειμένου να μπορέσει να εντοπίσει τις επιθέσεις τους, γεγονός που επιτρέπει στην μέθοδο να αναλύσει σενάρια επίθεσης σε εμπορικές εφαρμογές διαδικτύου.

Προκειμένου να παρουσιαστούν τα αποτελέσματα της μεθόδου, η ερευνητική ομάδα υλοποίησε το TaintCheck, ένα μηχανισμό ο οποίος μπορεί να πραγματοποιήσει δυναμική ανάλυση μολυσμένων δεδομένων μέσω της τεχνικής δυαδικής επανεγγραφής, την ώρα που η εφαρμογή υπό έλεγχο λειτουργεί.

Τέλος, προτείνεται μια καινούργια μέθοδος αυτόματης παραγωγής αποτυπωμάτων για τον εντοπισμό των επιθέσεων των κακόβουλων λογισμικών, με το όνομα «Παραγωγή αποτυπωμάτων με βάση την εννοιολογική ανάλυση». Μέσω της αντίστροφης προσπέλασης της αλυσίδας μετάδοσης μολυσμένων δεδομένων μέσα σε μια εφαρμογή, μπορεί κανείς να εντοπίσει αυτόματα ποιό σημείο στη ροή εκτέλεσης είναι υπεύθυνο για την επιτυχία της επίθεσης στην εφαρμογή.

6.7 ITS4: ΕΝΑΣ ΣΤΑΤΙΚΟΣ ΣΑΡΩΤΗΣ ΑΔΥΝΑΜΙΩΝ ΓΙΑ ΠΗΓΑΙΟ ΚΩΔΙΚΑ ΤΩΝ ΓΛΩΣΣΩΝ C ΚΑΙ C++ CODE (JOHN VIEGA, J.T. BLOCH, YOSHI KOHNO, GARY MCGRAW. RELIABLE SOFTWARE TECHNOLOGIES, DULLES, VIRGINIA)

Η έρευνα που παρουσιάζεται σε αυτή τη δημοσίευση περιγράφει το εργαλείο ITS4, ένα εργαλείο για την στατική σάρωση πηγαίου κώδικα γραμμένου στη γλώσσα προγραμματισμού C και τον εντοπισμό αδυναμιών μέσα στον κώδικα. Σε σύγκριση με παρόμοιες υλοποιήσεις, η συγκεκριμένη τεχνική σάρωσης προσπαθεί να διατηρήσει μια ισορροπία ανάμεσα στην ακρίβεια και στην αποδοτικότητα της μεθόδου ανίχνευσης αδυναμιών.

Η μέθοδος που παρουσιάζεται προσφέρει πληροφορίες στους αναλυτές ασφαλείας σε πραγματικό χρόνο χωρίς να παράγει μεγάλο αριθμό ψεύτικων θετικών αποτελεσμάτων (false positive results). Σε αντίθεση με άλλες τεχνικές, η συγκεκριμένη τεχνική είναι απλή, αφού είναι ικανή να σαρώνει μεγάλες ποσότητες πηγαίου κώδικα σε μικρό χρόνο, χωρίς να επιβαρύνεται από την πολυπλοκότητα της γλώσσας προγραμματισμού C.

Το εργαλείο ITS4 κατακερματίζει ένα στατικό αρχείο πηγαίου κώδικα σε μια σειρά από λεξικά σύμβολα κι έπειτα τα συγκρίνει με υπάρχοντα πρότυπα γνωστών αδυναμιών που διαθέτει, προκειμένου να εντοπίσει γνωστές αδυναμίες στον κώδικα. Τα πρότυπα που αναφέρθηκαν, εισάγονται από τον χρήστη του εργαλείου μέσω αρχείων κειμένου.

6.8 ΜΙΑ ΕΥΡΙΣΤΙΚΗ ΠΡΟΣΕΓΓΙΣΗ ΓΙΑ ΕΝΤΟΠΙΣΜΟ ΑΔΥΝΑΜΙΩΝ ΕΓΧΥΣΗΣ ΕΝΤΟΛΩΝ SQL ΣΕ ΔΙΑΔΙΚΤΥΑΚΕΣ ΕΦΑΡΜΟΓΕΣ (ANGELO CIAMPA, CORRADO AARON VISAGGIO, MASSIMILIANO DI PENTA, DEPT. OF ENGINEERING-RCOST, UNIV. OF SANNIO, ITALY)

Οι επιθέσεις έγχυσης εντολών SQL είναι ένας από τους πιο διαδεδομένους και επικίνδυνους τύπους επιθέσεων που χρησιμοποιούν οι κακόβουλοι χρήστες ενάντια στις διαδικτυακές εφαρμογές. Η συγκεκριμένη ερευνητική δημοσίευση προτείνει μια προσέγγιση εύρεσης αδυναμιών έγχυσης εντολών SQL και υλοποιεί ένα εργαλείο με το όνομα V1p3R για εκτέλεση δοκιμών διείσδυσης σε διαδικτυακές εφαρμογές.

Η μέθοδος που παρουσιάζεται στη παρούσα δημοσίευση είναι βασισμένη στο συνδυασμό υποδειγμάτων διαφόρων μηνυμάτων λάθους με τις εξόδους που παράγονται από τις εφαρμογές υπό έλεγχο. Βασίζεται σε μια εκτενή βάση γνώσης η οποία συντίθεται από ένα μεγάλο αριθμό προτύπων επιθέσεων. Σε αντίθεση με την συνηθισμένη πρακτική κατά την οποία δημιουργούνται τυχαίες επερωτήσεις SQL ως είσοδοι στο πρόγραμμα, το εργαλείο της συγκεκριμένης έρευνας βασίζεται στη βάση γνώσης που διαθέτει προκειμένου να χρησιμοποιήσει ευριστικές μεθόδους για την παραγωγή των επερωτήσεων που θα χρησιμεύσουν ως δεδομένα εισόδου στην εφαρμογή υπό έλεγχο.

Συγκεκριμένα, η μέθοδος αυτή αναλύει την διαδικτυακή εφαρμογή υπό έλεγχο με το σκοπό να καθορίσει τη δομή των υπερσυνδέσεων (hyperlinks) και να εντοπίσει τις φόρμες εισόδου δεδομένων. Στη συνέχεια, ξεκινά να παρέχει μια σειρά συνηθισμένων και διαδεδομένων επιθέσεων SQL, με σκοπό να επιτρέψει στην διαδικτυακή εφαρμογή να επιστρέψει κάποιο μήνυμα λάθους. Έπειτα, φιλτράρει τα δεδομένα εξόδου του μηνύματος μέσα από μια εκτενή βάση από κανονικές εκφράσεις που περιγράφουν γνωστές αδυναμίες τέτοιου τύπου. Οι περιγραφικές αυτές εκφράσεις έχουν παραχθεί από γνωστές βάσεις του εμπορίου, όπως βάσεις με διαδικτυακές διεπαφές τραπεζών και ιδιωτικών οργανισμών. Για λόγους εμπιστευτικότητας, η έρευνα δεν παρέχει τα ονόματα και τις διευθύνσεις των βάσεων που χρησιμοποιήθηκαν. Μέσα από αυτή την διαδικασία, το εργαλείο V1p3R παράγει ευριστικά καινούργιες επιθέσεις με επερωτήσεις που συνθέτει το ίδιο, βασισμένο στο κείμενο που απέκτησε από τα μηνύματα λάθους των βάσεων που ελέγχει. Η διαδικασία αυτή συνεχίζεται μέχρις ότου να μπορέσει το εργαλείο να βρει μια αδυναμία και να ανακτήσει έστω και ένα τμήμα της βάσης δεδομένων της εφαρμογής υπό έλεγχο.

Τα αποτελέσματα της έρευνας που πραγματοποιήθηκε με το εργαλείο V1p3R σε ένα σύνολο 12 αληθινών διαδικτυακών εφαρμογών, έδειξε πως η συγκεκριμένη μέθοδος έχει υψηλές επιδόσεις στην εύρεση αδυναμιών έγχυσης. Παρόλα αυτά, η έγχυση εντολών SQL ήταν έξω από τους στόχους της παρούσας διπλωματικής, οπότε η γνώση της συγκεκριμένης έρευνας δεν αξιοποιήθηκε σε κάποιο σημείο.

6.9 ΜΕΘΟΔΟΙ ΓΙΑ ΤΗΝ ΑΠΟΤΡΕΨΗ, ΤΟΝ ΕΝΤΟΠΙΣΜΟ ΚΑΙ ΤΗΝ ΑΠΟΜΑΚΡΥΝΣΗ ΑΔΥΝΑΜΙΩΝ ΛΟΓΙΣΜΙΚΟΥ (JAY-EVAN J. TEVIS, JOHN A. HAMILTON, JR. DEPARTMENT OF COMPUTER SCIENCE AND SOFTWARE ENGINEERING, AUBURN UNIVERSITY, AUBURN, ALABAMA)

Η έρευνα της συγκεκριμένης δημοσίευσης εστιάζει σε μια καινούργια μέθοδο ανάπτυξης σαρωτών για αδυναμίες στον πηγαίο κώδικα. Τα εργαλεία δοκιμών διείσδυσης και εύρεσης αδυναμιών ψάχνουν γενικά για «λάθη» στον πηγαίο κώδικα των εφαρμογών τα οποία μπορούν να αξιοποιηθούν από κακόβουλους χρήστες και ειδοποιούν τον προγραμματιστή για το μέρος που αυτά εμφανίζονται στον κώδικα. Παρόλο που τέτοιου είδους εργαλεία καλύπτουν μια βασική ανάγκη, αυτή της εύρεσης αδυναμιών στον κώδικα των εφαρμογών, υπάρχει μια κοινή αδυναμία των εργαλείων αυτών, η οποία είναι η προληπτική εύρεση αδυναμιών.

Ένας θεωρητικά τέλειος σαρωτής αδυναμιών θα μπορούσε να ανακαλύπτει τα κρισιμότερα μεταξύ του συνόλου των κριτηρίων για την εύρεση αδυναμιών σε πηγαίο κώδικα, αλλά παρόλα αυτά να μην μπορεί να προτείνει προληπτικά μέτρα ασφάλειας σε επισφαλή σημεία του κώδικα. Η πρόληψη χρειάζεται μια ολοκληρωμένη μελέτη της διαδικασίας εκτέλεσης του κώδικα, συγκεκριμενοποιημένη στον τρόπο και στόχο λειτουργίας της εκάστοτε εφαρμογής. Γι αυτό το λόγο, τεχνικά ένας σαρωτής μπορεί να εντοπίσει την ύπαρξη αδυναμιών, αλλά να μην ξέρει πια πιθανή σωστή λύση του προβλήματος ταιριάζει στο εκάστοτε κομμάτι κώδικα. Μια ορθή λύση σε ένα πρόγραμμα θα μπορούσε κάλλιστα να δημιουργεί άλλη ευπάθεια αν χρησιμοποιηθεί σε κάποιο άλλο. Ένας σαρωτής με ικανότητα προληπτικής αντιμετώπισης αδυναμιών, θα μπορούσε να εντοπίσει πολλαπλές πρακτικές προγραμματισμού οι οποίες θεωρούνται επισφαλείς και επικίνδυνες όταν χρησιμοποιούνται στο πηγαίο κώδικα και να προτείνει εναλλακτικούς τρόπους υλοποίησης του ίδιου κώδικα. Οι προτεινόμενες εναλλακτικές θα μπορούνε να περιέχουν συγκεκριμένα παραδείγματα υλοποιημένων αλγορίθμων και πηγαίου κώδικα που αντιμετωπίζει το λάθος που μπορεί να δημιουργηθεί, πριν καν αυτό υλοποιηθεί ως αδυναμία στην λειτουργία της εφαρμογής.

Η δημοσίευση αυτή ξεκινά με το να δώσει μια επισκόπηση των υπαρχόντων εφαρμογών σάρωσης κώδικα για αδυναμίες. Στη συνέχεια δίνει μια σύντομη περιγραφή της διαδικασίας αυτοματοποίησης της σάρωσης κώδικα για εντοπισμό αδυναμιών, όπως αυτή πραγματοποιείται από τα υπάρχοντα προγράμματα που διατίθενται στον ακαδημαϊκό και εμπορικό τομέα. Τελειώνει την παρουσίαση με μια συζήτηση πάνω στο λειτουργικό προγραμματισμό και στις τεχνικές που μπορούν να

αποτελέσουν μια καλύτερη λύση για την ανάπτυξη αντίστοιχων σαρωτών, με τα χαρακτηριστικά που αναφέρθηκαν νωρίτερα.

6.10 ΣΤΑΤΙΚΗ ΑΝΑΛΥΣΗ ΑΝΩΜΑΛΙΩΝ ΚΑΙ ΑΔΥΝΑΜΙΩΝ ΑΣΦΑΛΕΙΑΣ ΣΕ ΕΚΤΕΛΕΣΙΜΑ ΑΡΧΕΙΑ (JAY-EVAN J. TEVIS, JOHN A. HAMILTON, JR. DEPARTMENT OF COMPUTER SCIENCE, WESTERN ILLINOIS UNIVERSITY, MACOMB, ILLINOIS)

Η ακαδημαϊκή ομάδα που δημοσίευσε τη συγκεκριμένη έρευνα ανέπτυξε μια μεθοδολογία η οποία χρησιμοποιεί τις πληροφορίες που υπάρχουν στις επικεφαλίδες, στους δυαδικούς τομείς (sectors) και στους πίνακες ενός εκτελέσιμου αρχείου του λειτουργικού συστήματος Windows NT/XP. Προσπαθεί να εντοπίσει ανωμαλίες στο δυαδικό κώδικα των εκτελέσιμων αρχείων, καθώς και να φέρει στο φως τυχόν αδυναμίες ασφάλειας που μπορεί αυτό να κρύβει, χωρίς να χρειαστεί να πραγματοποιήσει αποσυναρμολόγηση (disassembly) του αρχείου σε κώδικα μηχανής.

Την μέθοδο που αναπτύχθηκε, την υλοποιήσανε σε ένα εργαλείο λογισμικού με το όνομα “findssv”, το οποίο πραγματοποιεί στατική εκτέλεση σε εκτελέσιμα αρχεία αυτόματα.

Συγκεκριμένα, η μέθοδος που αναπτύχθηκε περιλαμβάνει την σάρωση αρχείων που συμμορφώνονται με το μορφότυπο PE εκτελέσιμων αρχείων των λειτουργικών συστημάτων Windows NT και XP, αλλά και μεταγενέστερα. Η επικεφαλίδα PE περιέχει την επικεφαλίδα DOS, μια εναπομείνασα περιοχή υπογραφής από την επικεφαλίδα MS-DOS, το αποτύπωμα PE, την επικεφαλίδα του αρχείου, την επικεφαλίδα παραμέτρων, τον πίνακα τομέων (section header), τον πίνακα συμβόλων (symbol table), τον πίνακα συμβολοσειρών και το πίνακα εξαγωγών (export table).

Η μέθοδος κάνει χρήση των πληροφοριών κυρίως από τον πίνακα τομέων προκειμένου να εντοπίσει αδυναμίες που αναγνωρίζονται από τα χαρακτηριστικά κάθε τομέα που περιγράφεται στον πίνακα. Τα περιεχόμενα του πίνακα λένε στο λειτουργικό σύστημα αν κάποιος τομέας περιέχει εκτελέσιμο κώδικα ή δεδομένα αρχικοποίησης της εκτέλεσης. Επίσης περιγράφουν αν κάποια σημεία σε κάποιο τομέα είναι ανοιχτά προς ανάγνωση, εγγραφή ή εκτέλεση.

Η μέθοδος κάνει χρήση πληροφοριών από τον πίνακα συμβόλων προκειμένου να εντοπίσει τα ονόματα μεθόδων που μπορούν να χρησιμοποιηθούν για επιθέσεις υπερχείλισης της μνήμης. Κάθε όνομα μεθόδου στον πίνακα συγκρίνεται με μια λίστα συνηθισμένων μεθόδων των βιβλιοθηκών της γλώσσας προγραμματισμού C,

προκειμένου να εντοπιστούν μέθοδοι που είναι επιρρεπείς σε επιθέσεις υπερχείλισης της μνήμης.

Περιληπτικά, η μέθοδος δημιουργεί ένα αρχείο αναφοράς για το εκτελέσιμο αρχείο υπό έλεγχο, με βάση τις πληροφορίες που εξάγει και φιλτράρει από την PE επικεφαλίδα. Στη συνέχεια ελέγχει για διαφορές ανάμεσα στα αναφερόμενα μεγέθη δυαδικών περιοχών που αναφέρονται στην επικεφαλίδα και στα πραγματικά μεγέθη που έχουνε μέσα στο αρχείο. Μια τέτοια διαφορά θα μπορούσε να είναι ένδειξη ύπαρξης επιπλέον κώδικα από κάποιο κακόβουλο χρήστη, ο οποίος μπορεί να εκτελεστεί μαζί με την εκτέλεση του κανονικού προγράμματος.

Η μέθοδος σαρώνει τις πληροφορίες επικεφαλίδας, δημιουργώντας ένα χάρτη του πώς είναι κατανεμημένη η δυαδική πληροφορία μέσα στο εκτελέσιμο αρχείο. Αναλύει άγνωστες περιοχές πληροφορίας στο εκτελέσιμο αρχείο και ενημερώνει τον χρήστη για τυχόν κενές περιοχές που μπορούν να χρησιμοποιηθούν για εκτέλεση κώδικα μέσω δυαδικής έγχυσης εντολών.

Τα αποτελέσματα, αν και περιορισμένης έκτασης λόγω της φύσης των εκτελέσιμων PE αρχείων, αποτελούν μια από τις πρώτες ολοκληρωμένες προσπάθειες εξεύρεσης αδυναμιών δίχως την χρήση αποσυναρμολόγησης του εκτελέσιμου αρχείου υπό έλεγχο σε κώδικα μηχανής.

Παρόλο που η συγκεκριμένη δημοσίευση δεν παρείχε κάποια επιπλέον ιδέα για την ανάπτυξη της μεθόδου που παρουσιάζεται στη παρούσα διπλωματική, εντούτοις αποτελεί μια ολοκληρωμένη και αξιόλογη δουλειά στον τομέα της εύρεσης αδυναμιών σε εφαρμογές, οπότε κρίθηκε ότι έπρεπε να συμπεριληφθεί στις παρουσιάσεις.

6.11 PIXY: ΕΝΑ ΕΡΓΑΛΕΙΟ ΣΤΑΤΙΚΗΣ ΑΝΑΛΥΣΗΣ ΓΙΑ ΤΟΝ ΕΝΤΟΠΙΣΜΟ ΑΔΥΝΑΜΙΩΝ ΣΕ ΔΙΑΔΙΚΤΥΑΚΕΣ ΕΦΑΡΜΟΓΕΣ (NENAD JOVANOVIC, CHRISTOPHER KRUEGEL, AND ENGIN KIRDA, TECHNICAL UNIVERSITY OF VIENNA, SECURE SYSTEMS LAB)

Στα πλαίσια αυτής της δημοσίευσης [26], αναπτύχθηκε μια μέθοδος η οποία κάνει χρήση των δεδομένων της ροής εντολών κάθε λειτουργίας μιας διαδικτυακής εφαρμογής, προκειμένου να ανακαλύψει αδύναμα σημεία μέσα στο πρόγραμμα που μπορεί να εκμεταλλευτεί ένας κακόβουλος χρήστης. Επιπλέον, γίνεται χρήση μεθόδων αποσαφήνισης των αποτελεσμάτων, προκειμένου να υπάρξει όσο το δυνατόν μεγαλύτερη ακρίβεια στα αποτελέσματα.

Οι τεχνικές που παρουσιάζονται και χρησιμοποιούνται στη συγκεκριμένη δημοσίευση, χρησιμοποιούν τις γενικότερες μεθόδους εύρεσης αδυναμιών που βασίζονται στη μόλυνση δεδομένων. Η έρευνα υλοποιεί τέτοιες μεθόδους εύρεσης αδυναμιών για τον εντοπισμό διαφόρων ειδών πιθανών επιθέσεων, όπως επιθέσεις έγχυσης επερωτήσεων SQL και επιθέσεις έγχυσης εντολών ελέγχου.

Στα πλαίσια της έρευνας, αναπτύχθηκε και ένα εργαλείο ελεύθερου λογισμικού με το όνομα Pixy, σκοπός του οποίου είναι ο εντοπισμός αδυναμιών προδιαγεγραμμένης εκτέλεσης κώδικα PHP μέσω εξωτερικών ιστοσελίδων (cross-site scripting vulnerabilities in PHP).

6.12 ΕΞΕΡΕΥΝΩΝΤΑΣ ΤΟ BLACK-BOX: ΚΑΝΟΝΤΑΣ ΧΡΗΣΗ ΔΥΝΑΜΙΚΗΣ ΔΙΑΧΕΙΡΙΣΗΣ ΓΙΑ ΤΗΝ ΔΗΜΙΟΥΡΓΙΑ ΑΔΥΝΑΜΙΩΝ ΑΣΦΑΛΕΙΑΣ (BARTON P. MILLER, MIHAI CHRISTODORESCU, ROBERT IVERSON, TEVFIK KOSAR, ALEXANDER MIRGORODSKII, FLORENTINA POPOVICI. COMPUTER SCIENCES DEPARTMENT, UNIVERSITY OF WISCONSIN, MADISON)

Τα προγράμματα που τρέχουν σε επισφαλείς ή κακόβουλους διαδικτυακούς εξυπηρετητές έχουν θεωρούνται ως οι πιο ενδεδειγμένοι στόχοι για τις επιθέσεις ασφάλειας. Η τεχνολογία που κατέστησε εφικτές τέτοιου είδους επιθέσεις, είναι η δυνατότητα να μπορεί ν' αναλυθεί και να ελεγχθεί η εκτέλεση των διαδικτυακών προγραμμάτων. Η *δυναμική διαχείριση* (*dynamic instrumentantion*) παρέχει την απαραίτητη τεχνολογία για αυτού του είδους την ανάλυση κι έλεγχο. Όπως ενσωματώνεται και στη βιβλιοθήκη DynInst API που χρησιμοποιείται από την έρευνα, η δυναμική διαχείριση επιτρέπει την εύκολη κατασκευή εργαλείων, ικανών να:

(1) επιθεωρούν μια τρέχουσα διεργασία (process) εφαρμογής υπό εκτέλεση, και να λάβουν πληροφορίες για την δομή του προγράμματος,

(2) ελέγχουν την εκτέλεση του προγράμματος,

(3) προξενήσουν την δυναμική φόρτωση καινούργιων βιβλιοθηκών στο χώρο διευθύνσεων της εφαρμογής,

(4) συνδέουν νέες ακολουθίες εντολών στο κορμό ενός τρέχοντος προγράμματος αλλά και να τις αφαιρούν, καθώς και να

(5) αντικαθιστούν μεμονωμένες εντολές συστήματος ή ακόμα και ολόκληρες μεθόδους και λειτουργίες της εφαρμογής.

Με βάση αυτήν την τεχνολογία, η παρούσα έρευνα υλοποίησε δύο επιδείξεις χρήσης: Τον εντοπισμό ευπαθειών σε ένα κατανεμημένο σύστημα προγραμματισμού δράσεων με το όνομα Κόνδορας, και την παράκαμψη της διαδικασίας αυθεντικοποίησης σε έναν απομακρυσμένο υπολογιστή μέσω ενός επεξεργαστή κειμένου.

6.13 ΒΕΛΤΙΩΣΗ ΔΗΜΙΟΥΡΓΙΑΣ ΔΟΚΙΜΩΝ ΒΑΣΕΙ ΣΕΝΑΡΙΩΝ ΚΑΝΟΝΤΑΣ ΧΡΗΣΗ ΑΥΤΟΜΑΤΟΥ ΠΡΟΣΔΙΟΡΙΣΜΟΥ ΔΙΕΠΑΦΩΝ (WILLIAM G.J. HALFOND AND ALESSANDRO ORSO. COLLEGE OF COMPUTING. GEORGIA INSTITUTE OF TECHNOLOGY)

Η ταχύτατη ανάπτυξη των διαδικτυακών εφαρμογών έχει καταστήσει την αναγνώριση των διεπαφών επικοινωνίας των εφαρμογών με τον χρήστη μια δύσκολη και επίπονη διαδικασία. Πολλές τεχνικές σάρωσης και ανάλυσης αδυναμιών που λειτουργούν αποτελεσματικά σε απλές διαδικτυακές εφαρμογές, κρίνονται ανεπαρκείς όταν έρθουν αντιμέτωπες με σύγχρονες, δυναμικές διαδικτυακές εφαρμογές. Τα αποτελέσματά τους είναι ανεπαρκή, κυρίως λόγω της ανικανότητας τους να αναλύσουν επαρκώς τις εφαρμογές υπό έλεγχο.

Η εξαγωγή ημιτελών συμπερασμάτων και πληροφοριών από την ανάλυση μιας διαδικτυακής εφαρμογής περιορίζει την αποδοτικότητα των μεθόδων ανάλυσης, αφού αποτρέπει τα εργαλεία από το να μπορέσουν να προσδιορίσουν λειτουργίες τις διαδικτυακής εφαρμογής. Ο λόγος είναι η ανικανότητα προσδιορισμού της διεπαφής που υλοποιεί τις συγκεκριμένες λειτουργίες.

Προκειμένου να βρεθεί μια λύση στο ανωτέρω πρόβλημα, η συγκεκριμένη μελέτη παρουσιάζει μια μέθοδο για την αυτόματη αναγνώριση διεπαφών χρήστη-εφαρμογής σε διαδικτυακό περιβάλλον, η οποία κάνει χρήση πρωτοποριακών αλγορίθμων που διεξάγουν στατική ανάλυση στις εφαρμογές υπό έλεγχο. Ο πυρήνας αυτής της προσέγγισης είναι βασισμένος σε μια τεχνική στατικής ανάλυσης πηγαίου κώδικα, η οποία είναι ικανή να αναλύσει τις διαδικτυακές εφαρμογές που παρουσιάζουν πολλαπλά δυναμικά στοιχεία και έχουν πολλαπλές –ίσως και κρυμμένες- διεπαφές επικοινωνίας.

Ο αλγόριθμος δουλεύει σε δύο στάδια: Στο πρώτο στάδιο, αναγνωρίζει πληροφορίες γύρω από τις παραμέτρους κάθε τομέα λειτουργίας της εφαρμογής (είδη μεταβλητών και πιθανές τιμές τους), παράμετροι οι οποίες μπορεί να είναι τμήμα από τις διεπαφές τις διαδικτυακής εφαρμογής. Στο δεύτερο στάδιο, ο αλγόριθμος

αναγνωρίζει το πώς αυτές οι παράμετροι ομαδοποιούνται και χρησιμοποιούνται από τις διαφορετικές διεπαφές που έχει η εφαρμογή. Το τελικό αποτέλεσμα είναι ένα σύνολο διεπαφών όπου η κάθε διεπαφή συντίθεται από ένα σύνολο παραμέτρων και πληροφοριών τομέα λειτουργίας για αυτές τις παραμέτρους. Αυτό το σύνολο διεπαφών χρησιμοποιείται μετά για την υποστήριξη και βελτίωση της λειτουργίας αυτόματων σαρωτών για εύρεση αδυναμιών και πιστοποίηση κώδικα.

6.14 SWADDLER: ΜΙΑ ΠΡΟΣΕΓΓΙΣΗ ΓΙΑ ΤΟΝ ΕΝΤΟΠΙΣΜΟ ΠΑΡΑΒΙΑΣΕΩΝ ΤΩΝ ΠΕΡΙΟΡΙΣΜΩΝ ΚΑΤΑΣΤΑΣΗΣ ΣΕ ΔΙΑΔΙΚΤΥΑΚΕΣ ΕΦΑΡΜΟΓΕΣ ΚΑΝΟΝΤΑΣ ΧΡΗΣΗ ΑΝΩΜΑΛΙΩΝ (MARCO COVA, DAVIDE BALZAROTTI, VIKTORIA FELMETSGER, AND GIOVANNI VIGNA. DEPARTMENT OF COMPUTER SCIENCE, UNIVERSITY OF CALIFORNIA SANTA BARBARA)

Σε αυτή τη έρευνα, η ομάδα ασφαλείας που την διεκπεραίωσε παρουσιάζει το εργαλείο Swaddler, μια καινούργια προσέγγιση για τον εντοπισμό αδυναμιών που μπορούν να χρησιμοποιηθούν για επίθεση σε μια διαδικτυακή εφαρμογή, η οποία κάνει χρήση αναγνώρισης ανωμαλιών ενδεδειγμένης λειτουργίας στη δομή της εφαρμογής υπό έλεγχο.

Το εργαλείο αναλύει την εσωτερική κατάσταση μιας διαδικτυακής εφαρμογής και εξάγει σχέσεις συσχέτισης ανάμεσα στις κρίσιμες εντολές εκτέλεσης του πηγαίου κώδικα και στην εκάστοτε εσωτερική κατάσταση της εφαρμογής υπό έλεγχο. Με αυτό το τρόπο, το Swaddler μπορεί να αναγνωρίσει επιθέσεις που προσπαθούν να οδηγήσουν την λειτουργία μιας εφαρμογής σε ασταθή κατάσταση. Μια τέτοια κατάσταση θα μπορούσε να είναι η παραβίαση της σχεδιασμένης ροής εκτέλεσης των λειτουργιών και η μετάβαση της λειτουργίας της εφαρμογής σε μη-αναμενόμενες καταστάσεις.

Η έρευνα αποδεικνύει πως η μέθοδος εντοπισμού ανωμαλιών ενδεδειγμένης λειτουργίας που βασίζεται τόσο στις τιμές μεμονωμένων μεταβλητών, όσο και στις σχέσεις πολλαπλών μεταβλητών μεταξύ τους, αποτελεί έναν αποτελεσματικό τρόπο εντοπισμού περίπλοκων αδυναμιών που μπορούν να χρησιμοποιηθούν σε επιθέσεις ενάντια στην εφαρμογή υπό έλεγχο. Παραλλαγή της συγκεκριμένης θεωρητικής προσέγγισης χρησιμοποιήθηκε και στην ανάπτυξη του εργαλείου LogGIC, κατά την διαδικασία ελέγχου των αποτελεσμάτων της δυναμικής ανάλυσης με χρήση του εργαλείου Daikon όπως θα δούμε και παρακάτω.

Σε αυτό το σημείο τελειώνει η παρουσίαση των διάφορων ερευνών του χώρου της Ασφάλειας που ασχολούνται με μεθόδους αυτόματου εντοπισμού αδυναμιών. Εξαιρώντας τα διάφορα θεωρητικά κομμάτια που χρησιμοποιήθηκαν από τις προηγούμενες έρευνες, μια εκ των μεθόδων που υλοποιούνται στη παρούσα διπλωματική βασίστηκε στην έρευνα: «Toward Automated Detection of Logic Vulnerabilities in Web Applications - Christopher Kruegel Giovanni Vigna»[2]. Η συγκεκριμένη δημοσίευση δεν αναλύεται συνοπτικά στο παρόν κεφάλαιο, καθώς η θεωρητική μεθοδολογία που χρησιμοποιεί, χρησιμοποιήθηκε με κάποιες αλλαγές στην παρούσα διπλωματική και στην ανάπτυξη ενός τμήματος του εργαλείου LogGIC Μία εκ των θεωρητικών μεθόδων εύρεσης λογικών αδυναμιών που χρησιμοποιεί το LogGIC είναι ουσιαστικά μια παραλλαγή της θεωρητικής μεθόδου που παρουσιάζεται στη ανωτέρω δημοσίευση [2]. Οι υπόλοιπες μέθοδοι και διαδικασίες εύρεσης λογικών αδυναμιών που υλοποιούνται στο LogGIC, στηρίζονται στις θεωρίες Διανυσμάτων Εισόδου[15], καθώς και σε ποικίλες άλλες έρευνες [18][17][20][10]. Γι αυτό το λόγο, η μεθοδολογία παρουσιάζεται αναλυτικά στο επόμενο κεφάλαιο 7.5 της διπλωματικής εργασίας μου, ως τμήμα της έρευνας που πραγματοποιήθηκε στα πλαίσια ανάπτυξης της μεθόδου και του εργαλείου LogGIC.

6.15 ΚΟΙΝΑ ΚΡΙΤΗΡΙΑ ΑΞΙΟΛΟΓΗΣΗΣ ΚΑΙ Η ΚΛΙΜΑΚΑ CCWAPPS

Ο λόγος ύπαρξης της κλίμακας CCWAPSS είναι προκειμένου να υπάρχει μία κοινή μέθοδος αξιολόγησης για την ασφάλεια των διαδικτυακών εφαρμογών ανάμεσα στους αναλυτές ασφαλείας και στους τελικούς πελάτες των εφαρμογών. Η κλίμακα δεν αποσκοπεί στο να αντικαταστήσει άλλα πρότυπα αξιολόγησης. Προτείνει απλώς έναν απλό τρόπο αξιολόγησης του επιπέδου ασφαλείας μιας διαδικτυακής εφαρμογής. Η κλίμακα CCWAPSS εστιάζει στην αξιολόγηση του επιπέδου ασφαλείας μιας διακριτής και αυτόνομης εφαρμογής ή διαδικτυακής υπηρεσίας [27].

Common criteria

Authentication	
Best Practices	Authentication mechanisms should prevent users without **credentials** from accessing application functionality and prevents against brute-force attacks by restricting smartly block malicious passwords guessing attempts.
Common issues	Login processs bypass, auto-login script, identity switching, ...
OWASP Topics:	Section 12 "Guide to Authentication"
Authorization	
Best Practices	Mechanisms should **prevent authenticated users from accessing others users data** or functionalities without the appropriate privileges.
Common issues	Hidden or guessable privileged functionalities, a user A can read/modify data of an user B, GET/POST parameters fuzzing...
OWASP Topics:	Section 13 "Guide to Authorization"
User's Input Sanitization	
Best Practices	**All user-controlled data should be checked** for validity. Bounds checking should be used to prevent buffer overflows and/or variable assignment violations. Syntax checking should be enforced to prevent data encoding, data injection, and/or format string attacks, and to reject forbidden characters. **Output data** generated using users controlled inputs must be sanitized and properly formatted to avoid any client-side script injection through the application.
Common issues	Parameters overflow, JavaScript injection, SQL Injection, Cross Site Scripting, server-side file inclusion, ...
OWASP Topics:	Section 21 "Buffer Overflows", Section 15 "Data Validation", Section 16 "Interpreter Injection"

ΕΙΚΟΝΑ 3: ΜΕΡΟΣ ΤΟΥ ΟΡΙΣΜΟΥ ΚΡΙΤΗΡΙΩΝ ΤΗΣ CCWAPSS [27]

Εκ των έντεκα κριτηρίων που περιλαμβάνει η κλίμακα CCWAPSS -όπως παρουσιάστηκε σε προηγούμενο κεφάλαιο- το LogGIC βρίσκει υποσύνολα αδυναμιών που εμπίπτουν στις πρώτες τρεις κατηγορίες που φαίνονται στην παραπάνω εικόνα. Οι τρεις αυτές κατηγορίες καλύπτουν ένα ευρύτερο φάσμα εφαρμογών, το οποίο επεκτείνεται σε αδυναμίες που εμφανίζονται σε ευρύτερο σύνολο εφαρμογών, πέραν των διαδικτυακών που αξιολογεί η κλίμακα CCWAPSS. Τα υπόλοιπα κριτήρια μηχανισμών αφορούν διάφορα αντικείμενα των διαδικτυακών εφαρμογών, μερικά εκ των οποίων είναι τα ακόλουθα:

- Πολυπλοκότητα των συνθηματικών,
- Η διατήρηση εμπιστευτικότητας των δεδομένων των χρηστών,
- Μηχανισμοί για έλεγχο των αναγνωριστικών συνεδριών (session ids)
- Διαχείριση ανανεώσεων λογισμικού (updates),
- Αποξένωση κρίσιμων διεπαφών (όπως αυτές των διαχειριστών) και,
- Ασφάλεια δικτυακών επικοινωνιών και επικινδυνότητα αλληλεπίδρασης εφαρμογών με υπηρεσίες τρίτων.

7. Η ΜΕΘΟΔΟΣ LOGGIC

7.1 ΕΙΣΑΓΩΓΗ

Ένα μεγάλο ποσοστό των ευπαθειών και των αδυναμιών που μπορούν να αξιοποιηθούν από κακόβουλους χρήστες αντιστοιχεί στις λογικές αδυναμίες που παρουσιάζονται σε μεγάλα και μικρά πληροφοριακά συστήματα και προγράμματα, ανεξάρτητα από το αν αυτά είναι προσβάσιμα διαδικτυακά ή όχι. Τα τελευταία χρόνια, η ανάγκη εντοπισμού και αντιμετώπισης των λογικών αδυναμιών αρχίζει να κερδίζει τα φώτα της δημοσιότητας, κερδίζοντας την προσοχή μεγάλων ερευνητών του χώρου της Πληροφορικής και του ευρύτερου ακαδημαϊκού τομέα. Παρόλα αυτά, η ανάλυση λογικών αδυναμιών είναι μία δύσβατη περιοχή που βρίσκεται υπό ανάπτυξη, δίχως να παρουσιάζει τα σημάδια ωριμότητας άλλων περιοχών της Ασφάλειας.

Με τον όρο *λογικό λάθος* ή *λογική αδυναμία* εννοούμε έναν συγκεκριμένο, διαθέσιμο και επιτρεπτό συνδυασμό δράσεων ενός χρήστη πάνω σε μία εφαρμογή, ο οποίος οδηγεί τις λειτουργίες της σε ανεπιθύμητα αποτελέσματα. Η μέθοδος που αναπτύχθηκε δεν αποτελεί μια ακόμα υλοποίηση συνηθισμένων δοκιμών διείσδυσης που ελέγχουν τεχνικές αδυναμίες (όπως έγχυση εντολών SQL, υπερχείλιση πινάκων κ.τ.λ.) αλλά στοχεύει τα λάθη λογικής στον προγραμματισμό των λειτουργιών των εφαρμογών. Μια λειτουργία προσαρμοσμένη στις ανάγκες και τους στόχους της εφαρμογής της, δεν μπορεί να ερμηνευτεί με μη-αμφισβητήσιμο τρόπο, πόρισμα που εξάγεται από τις αρχές των Cohen και Turing. Τα λογικά λάθη χρησιμοποιούνται συχνά ως κερκόπορτες των κακόβουλων χρηστών για να παρακάμψουν όλων των ειδών τους περιορισμούς στις λειτουργίες του λογισμικού. Κάποια παραδείγματα λογικών αδυναμιών είναι τα ακόλουθα:

”*Δοσμένης μιας εφαρμογής που υλοποιεί ένα διαδικτυακό μαγαζί πωλήσεων, υλοποιείται ένα σύστημα κουπονιών που επιτρέπει στους πελάτες να αξιοποιήσουν μια έκπτωση σε ορισμένα αγαθά κατά τη διάρκεια των συναλλαγών τους. Η γενική λογική ιδέα είναι ότι κάθε κουπόνι μπορεί να χρησιμοποιηθεί μόνο μία φορά και για ένα αγαθό αλλά, αν από λογικό λάθος της εφαρμογής επιτρέπεται σε ένα κακόβουλο χρήστη να χρησιμοποιήσει ένα κουπόνι πολλαπλές φορές, θα μπορούσε να επαναλάβει τη διαδικασία μέχρι να μειώσει την τιμή του αγαθού που θέλει να αγοράσει σε μηδέν*” [2].

Ένα άλλο παράδειγμα λογικής αδυναμίας είναι μια παράπλευρη διαδικασία ενεργειών την οποία μπορεί να αξιοποιήσει ένας κακόβουλος χρήστης προκειμένου

να αποφύγει την διαδικασία αυθεντικοποίησης στη διεπαφή εισαγωγής συνθηματικών μιας εφαρμογής:

"*Κάθε εφαρμογή με διαδικασία αυθεντικοποίησης παρέχει και μια παράλληλη διαδικασία με την οποία κάποιος μπορεί να ανακτήσει το συνθηματικό του σε περίπτωση που το έχει ξεχάσει. Η λογική της διαδικασίας είναι ο χρήστης να παρέχει το email του και η εφαρμογή να στέλνει επιβεβαιωτικό email με το αντίστοιχο συνθηματικό στο mail inbox του χρήστη που έκανε την αίτηση. Αν από λογικό λάθος η εφαρμογή, μετά την αποστολή mail, δεν επιστρέφει στην διαδικασία αυθεντικοποίησης αλλά αυθεντικοποιεί τον χρήστη με βάση τη σωστή παροχή email, τότε ένας κακόβουλος χρήστης θα μπορούσε να μαντέψει ή να βρει το email ενός εγγεγραμμένου χρήστη και να χρησιμοποιήσει την διαδικασία ανάκτησης συνθηματικού προκειμένου να εκτελέσει μια επίθεση αντιγραφής ταυτότητας στην εφαρμογή*."

Προκειμένου η δουλειά της παρούσας διπλωματικής να μπορέσει να ελεγχθεί και πρακτικά, υλοποιήθηκε επιπλέον μια εφαρμογή γραφικής διεπαφής με το όνομα LogicBomb. Η εφαρμογή LogicBomb υλοποιήθηκε επίσης στην γλώσσα προγραμματισμού Java και χρησιμοποιείται ως *έδαφος δοκιμών* ("*test-bed*") για την απόδειξη της ορθής λειτουργίας της μεθόδου και του προγράμματος LogGIC που αναπτύσσονται στα πλαίσια της παρούσας διπλωματικής. Η εφαρμογή αυτή περιλαμβάνει διάφορα είδη λογικών αδυναμιών τα οποία η μέθοδος που υλοποιήθηκε καλείται να εντοπίσει και να αναγνωρίσει, χρησιμοποιώντας ένα καθολικό και διαλειτουργικό (δηλαδή μη-εξειδικευμένο στον εκάστοτε έλεγχο) τρόπο εξεύρεσης αδυναμιών.

Η διαδικασία εξεύρεσης λογικών αδυναμιών σε εφαρμογές αποτελεί ένα εξαιρετικά δύσκολο στόχο, καθαρά λόγω της φύσης των αδυναμιών που αναζητούνται. Οι λογικές αδυναμίες διαφέρουν μεταξύ τους ως προς τη δομή, το τρόπο εμφάνισης και τα ιδιαίτερα χαρακτηριστικά που έχει η κάθε μία από υλοποίηση σε υλοποίηση. Η κατάσταση δυσχεραίνεται ακόμη περισσότερο όταν ο στόχος της έρευνας είναι η χρήση ελέγχων τύπου Black-box για εξεύρεση λογικών αδυναμιών. Σε αυτή τη περίπτωση οι πληροφορίες που παρέχονται είναι ελάχιστες, αν όχι μηδαμινές, κάνοντας έτσι την ανάπτυξη μιας τέτοιας μεθόδου εύρεσης λογικών αδυναμιών ένα υπερβολικά δύσκολο στόχο.

Έχοντας τις ανωτέρω δυσκολίες στο μυαλό, στόχος της διπλωματικής είναι η διαμόρφωση και υλοποίηση μιας αυτόματης μεθόδου που να μπορεί να προσεγγίσει, να εντοπίσει και, εν μέρει, να παρουσιάσει τυχόν λογικά λάθη σε προγράμματα Java. Η μέθοδος κατευθύνει με τα αποτελέσματά της τον αναλυτή σε συγκεκριμένα σημεία του κώδικα μιας εφαρμογής στα οποία πιθανολογείται να υπάρχει κάποιο λογικό λάθος.

Παρόλο που δεν φαντάζει πίθανο για τα τωρινά δεδομένα να βρεθεί μέθοδος που να αυτοματοποιεί πλήρως την διαδικασία εύρεσης λογικών αδυναμιών, καθαρά λόγω της φύσης του ίδιου του λάθους, είναι εντούτοις δυνατό να μειωθεί χαρακτηριστικά ο όγκος της δουλειάς ενός αναλυτή λογικών αδυναμιών, μέσω της χρήσης αλγοριθμικής ανάλυσης των προγραμμάτων υπό έλεγχο. Ένας προσεγγιστικά σωστός εντοπισμός των σημείων που μπορεί να υπάρχουν λογικά λάθη θα μπορούσε να περιορίσει τη δουλειά του αναλυτή σε μεγάλο σημείο. Ο αναλυτής θα δίνει έμφαση σε λίγα, συγκεκριμένα και διακριτά σημεία στο κώδικα, αντί να διαβάζει ολόκληρα τα αρχεία πηγαίου κώδικα και να ελέγχει διεξοδικά ολόκληρο το λογισμικό από την αρχή μέχρι το τέλος.

Ένας ορθός τρόπος που θα μπορούσε να χρησιμοποιηθεί προκειμένου να προσεγγιστούν τα λογικά λάθη, θα ήταν να καταγραφούν ολόκληρα τα μονοπάτια εκτέλεσης (execution paths) για κάθε πιθανή λειτουργία και πιθανή εκτέλεση εντολών της εφαρμογής υπό έλεγχο. Παρόλα αυτά, μία τέτοια υλοποίηση θα ήταν απαγορευτική, τόσο λόγω του όγκου δεδομένων που θα έπρεπε να παραχθούν και να αποθηκευτούν, όσο και λόγω της δυσκολίας στο να παραχθούν σωστά και με ορθό τρόπο αυτά τα δεδομένα[15]. Ως εκ τούτου, έπρεπε να βρεθεί ένας διαφορετικός, ευριστικός και πιο γρήγορος τρόπος εντοπισμού λογικών αδυναμιών που θα είναι πρακτικά εφαρμόσιμος τόσο σε μικρές όσο και σε μεγάλες εφαρμογές.

7.2 ΛΟΓΙΚΑ ΛΑΘΗ ΚΑΙ Η ΜΕΘΟΔΟΣ ΤΗΣ ΔΙΠΛΩΜΑΤΙΚΗΣ

Ένα λογικό λάθος ορίζεται ως μια παρέκκλιση από την προγραμματισμένη ροή εκτέλεσης εντολών μιας λειτουργίας, κατά την οποία μια εφαρμογή παρουσιάζει ασυνεπή συμπεριφορά όσων αφορά τον προδιαγεγραμμένο στόχο της λειτουργίας της [2]. Το λογικό λάθος αποτελεί μια λανθασμένη επιλογή μέσα σε μια ροή αποφάσεων. Αν θεωρηθεί ότι υπάρχει ένας συγκεκριμένος στόχος για κάθε μια λειτουργία ενός προγράμματος, τότε αυτός ο στόχος επιτυγχάνεται με την λήψη συγκεκριμένων αποφάσεων κατά την εκτέλεση των εντολών. Η εκτέλεση της εφαρμογής κατευθύνεται μέσω τέτοιων αποφάσεων που παίρνουν την μορφή εντολών διακλάδωσης υπό συνθήκη, όπως εντολές τύπου ‘if’ και ‘switch’. Οι αποφάσεις αυτές υπάρχουν στο κώδικα υπό την μορφή ελέγχων και *διακλαδώσεων* (*branches*) της εκτέλεσης και λειτουργούν βάσει συγκεκριμένων, διακριτών περιορισμών που αυτές εφαρμόζουν στην ροή εκτέλεσης. Μελετώντας αυτές τις εντολές διακλάδωσης και θέτοντάς τες ως την βάση για την εύρεση λογικών αδυναμιών, μπορούμε να συνοψίσουμε την λογική εκτέλεσης μιας λειτουργίας με το να αξιολογήσουμε την

ορθότητα των μεταβλητών που αποτελούν τους περιορισμούς για την διακλάδωση της εκτέλεσης. Αυτές οι μεταβλητές ορίζουν τα βασικά στοιχεία που μπορούν να προδώσουν την ύπαρξη ή όχι ενός λογικού λάθους μέσα σε μια εφαρμογή.

Ως συμπληρωματική διαδικασία της ανωτέρω μεθόδου που αναπτύχθηκε στα πλαίσια της παρούσας διπλωματικής, υλοποιείται και μια διαδικασία ανάλυσης Μολυσμένων αντικειμένων (5.2.1 ΔΙΑΣΠΟΡΑ ΜΟΛΥΣΜΕΝΩΝ ΑΝΤΙΚΕΙΜΕΝΩΝ (TAINTED OBJECT PROPAGATION)) η οποία μελετά τα διανύσματα εισόδου δεδομένων από τον χρήστη και τις μεταβλητές που αυτά επηρεάζουν. Τα

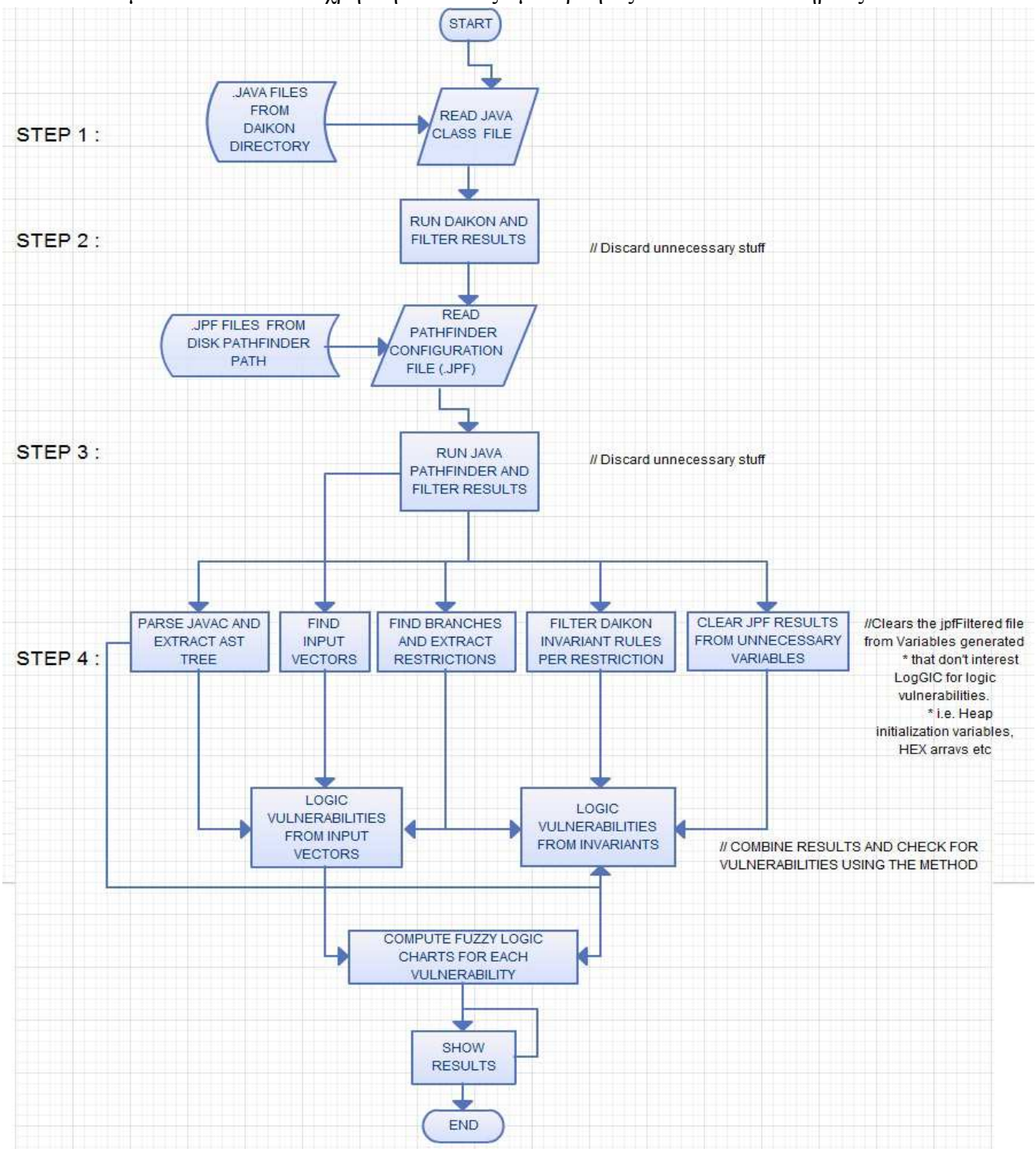


FIGURE 1: ΔΙΑΓΡΑΜΜΑ ΡΟΗΣ ΜΕΘΟΔΟΥ LOGGIC

αποτελέσματα αυτής της μεθόδου βοηθούν ιδιαίτερα την εύρεση λογικών αδυναμιών που στηρίζονται στα δεδομένα εισόδου των χρηστών.

Παρόλο που αυτός ο τρόπος ελέγχου απορρίπτει ένα μεγάλο όγκο πληροφοριών και πηγαίου κώδικα της εφαρμογής υπό έλεγχο, εντούτοις ο τρόπος αυτός αποδεικνύεται επαρκής και αποτελεσματικός ως προς τον εντοπισμό των πιθανών λογικών αδυναμιών που μελετά η παρούσα διπλωματική.

Η τρίτη μέθοδος εύρεσης λογικών αδυναμιών (η οποία κάνει ταυτόχρονα χρήση των συμπερασμάτων της ανωτέρω λογικής των διακλαδώσεων) είναι η αξιολόγηση των κανόνων που παράγει η Δυναμική Ανάλυση μιας εφαρμογής. Η αξιολόγηση των κανόνων που παράγει η δυναμική ανάλυση γίνεται με χρήση της ανωτέρω ανάλυσης κώδικα, αλλά και με βάση τα μονοπάτια εκτέλεσης που παράγει η στατική ανάλυση της εφαρμογής υπό έλεγχο. Όλα τα αποτελέσματα συνδυάζονται, συγκρίνονται και επιβεβαιώνουν ή διαψεύδουν ένα λογικό κανόνα της Δυναμικής Ανάλυσης, προσδιορίζοντας έτσι την ύπαρξη ή μη μιας λογικής αδυναμίας. Η διαδικασία αυτή αποτελεί την ακρογωνιαία λίθο των μελετών που ασχολούνται με διαδικασίες εύρεσης λογικών ευπαθειών και πάνω σε αυτή τη σκέψη στηρίζεται και η μέθοδος της ερευνητικής ομάδας του πανεπιστημίου της Καλιφόρνια [2], μέρος της οποίας χρησιμοποιείται στη παρούσα διπλωματική και θα αναπτυχτεί αναλυτικότερα σε ακόλουθα κεφάλαια.

7.3 Η ΜΕΘΟΔΟΣ ΠΡΟΣΕΓΓΙΣΗΣ ΛΟΓΙΚΩΝ ΕΠΙΛΟΓΩΝ

Προκειμένου να γίνει περισσότερο κατανοητή η ευριστική μέθοδος που υλοποιείται στην παρούσα διπλωματική, ακολουθεί μια αναλυτικότερη παρουσίαση της λογικής ελέγχου μιας διακλάδωσης κατά την εκτέλεση ενός προγράμματος. Ας υποθέσουμε ότι υπάρχει μια διακλάδωση τύπου if στον κώδικα με την μορφή:

if (A && B)

η οποία είναι υπεύθυνη για την αυθεντικοποίηση ή όχι του χρήστη, ανάλογα με τα διαπιστευτήρια που αυτός έδωσε ως είσοδο. Προκειμένου ο έλεγχος να είναι αληθής πρέπει και οι δύο μεταβλητές της παρένθεσης να έχουν τιμή αληθή, λόγω του λογικού ΚΑΙ (&&) της εξίσωσης. Η σχέση Α θα μπορούσε να είναι μια μεταβλητή της μορφής ‘ *Boolean notNull* ‘, δηλαδή μια μεταβλητή σωστού/λάθους που είναι αληθής μόνο όταν ο χρήστης έχει εισάγει κάποια στοιχεία στην εφαρμογή (δεν είναι κενά τα πεδία τιμών όπως όνομα χρήστη (username) και συνθηματικό (password)). Έστω ότι η σχέση Β είναι μια τιμή τύπου σωστού λάθους όπως και η Α, η οποία όμως έχει τιμή ‘αληθής’ μόνο αν τα διαπιστευτήρια αυθεντικοποίησης που έδωσε ο χρήστης

ταιριάζουν με κάποια από την αντίστοιχη βάση δεδομένων (δηλαδή πρέπει ο χρήστης να υπάρχει στο σύστημα και να είναι νόμιμος χρήστης της εφαρμογής). Η Β θα μπορούσε να είναι της μορφής '*Boolean CorrectCredentials*'. Η αξιολόγηση της διακλάδωσης θα έπρεπε κανονικά να είναι διαφορετική ανάμεσα σε ορθές και λανθασμένες ροές εκτέλεσης του προγράμματος.

Σε ένα πρόγραμμα που περιέχει λογικό λάθος, ο έλεγχος της μεταβλητής Β θα μπορούσε να λείπει ή να μην εκτελείται σωστά σε όλες τις πιθανές ροές εκτέλεσης του προγράμματος. Αν αυτό συνέβαινε, τότε κάθε έλεγχος ροής εκτέλεσης που ικανοποιούσε τη μεταβλητή Α, ταυτόχρονα θα σήμαινε πως ολόκληρος ο έλεγχος είναι αληθής και η εκτέλεση πρέπει να συνεχίσει μέσα στο if, γεγονός που όμως δεν είναι σωστό. Όπως γίνεται εύκολα αντιληπτό, κάθε εκτέλεση είναι λογικά σωστή μέχρι τη στιγμή που το Α θα είναι αληθές και το Β θα έπρεπε να μην είναι σωστό, δηλαδή να ισχύει (Α ^ ¬Β). Σε αυτή την περίπτωση, αν ολόκληρος ο έλεγχος διακλάδωσης επιστρέψει σωστό λόγω έλλειψης του Β, τότε έχουμε ένα λογικό λάθος στην εκτέλεση. Στο ανωτέρω παράδειγμα που δώσαμε με την αυθεντικοποίηση χρήστη, η έλλειψη της μεταβλητής Β ως στοιχείο ελέγχου θα σήμαινε ότι, οποιονδήποτε συνδυασμό ονόματος χρήστη και συνθηματικού (username και password) να έδινε ένας χρήστης, η εφαρμογή θα τον κατηγοριοποιούσε ως υπαρκτό και νόμιμο χρήστη του προγράμματος.

Παρατηρώντας τον τρόπο λειτουργίας γίνεται κατανοητό πως, αν κανείς παρακολουθεί της μεταβλητές που αποτελούν τους περιορισμούς για την διακλάδωση της εκτέλεσης εντολών μιας εφαρμογής, μπορεί να εξάγει συμπεράσματα για την ύπαρξη ή όχι λογικών αδυναμιών. Η αξιολόγηση των περιορισμών μπορεί να οδηγήσει στην εύρεση πιθανών λογικών αδυναμιών. (Προκειμένου να υπάρξει καλύτερη αξιοποίηση των αποτελεσμάτων από τον αναλυτή, το LogGIC κάνει χρήση ενός μοντέλου *Ασαφούς Λογικής* (*Fuzzy Logic*, στα αγγλικά εφεξής) για να ορίσει διακριτούς κανόνες που δρουν ως διαχειριστές του μεγέθους κινδύνου των σημείων κώδικα στα οποία ενδεχομένως να έχουν εντοπιστεί αδυναμίες, όπως θα δούμε αναλυτικά σε ακόλουθο κεφάλαιο).

Αυτός ο ευριστικός τρόπος προσέγγισης των λογικών αδυναμιών, αν και αποτελεσματικός, έχει κάποιους περιορισμούς όπως θα εξηγήσουμε και στα ακόλουθα κεφάλαια και, παρόλο που παράγει αποτελέσματα, δεν θα πρέπει να θεωρηθεί ικανός να εντοπίσει όλα τα πιθανά λογικά λάθη ενός προγράμματος.

Η κριτική ικανότητα ενός έμπειρου αναλυτή κρίνεται απαραίτητη προκειμένου να εντοπιστούν όλα τα λογικά λάθη και πάλι, αυτό δεν είναι πάντα εφικτό. Παρόλα αυτά, η περιορισμένη έρευνα πάνω στο συγκεκριμένο τομέα και οι λίγες πρακτικά χρήσιμες μέθοδοι που έχουν προταθεί για εύρεση λογικών αδυναμιών,

θέτει την δουλειά που παρουσιάζεται σε αυτή την διπλωματική κοντά στην αιχμή των εξελίξεων.

7.4 ΜΕΘΟΔΟΣ ΑΝΑΛΥΣΗΣ ΔΙΑΝΥΣΜΑΤΩΝ ΕΙΣΟΔΟΥ

Προκειμένου να επεκταθεί η μέθοδος εύρεσης των λογικών αδυναμιών ενός προγράμματος, καθώς και για να δοθεί στην μέθοδό μας η ικανότητα εντοπισμού λογικών αδυναμιών που είναι άμεσα συνδεδεμένα με τα δεδομένα εισόδου που παρέχει ο χρήστης και δεν εντοπίζονται από την ανάλυση των αποτελεσμάτων των υπόλοιπων τεχνικών ανάλυσης, στη παρούσα διπλωματική αναπτύχθηκε και προστέθηκε στο εργαλείο μια μέθοδος ανάλυσης και εντοπισμού των δεδομένων που παρέχονται από το χρήστη της εφαρμογής υπό έλεγχο μέσω των Διανυσμάτων Εισόδου της εφαρμογής. Η μέθοδος χρησιμοποιεί ως βάση την μέθοδο *Εύρεσης αδυναμιών μέσω Μολυσμένων Αντικειμένων* (βλ. κεφ.5.2.1, σελ. 30) και αποτελεί συμπληρωματική λειτουργία της μεθόδου εύρεσης λογικών αδυναμιών μέσω της ανάλυσης των διακλαδώσεων και των αποφάσεων εκτέλεσης εντολών.

Ακολουθώντας τις μεθόδους ανάλυσης Μολυσμένων Αντικειμένων, η μέθοδος που αναπτύχθηκε στο LogGIC εντοπίζει τα πιθανά διανύσματα εισόδου δεδομένων σε μια εφαρμογή γραφικής διεπαφής Java και τα προσθέτει στη λίστα πιθανών σημείων ύπαρξης λογικών αδυναμιών. Ύστερα ακολουθεί την μεταφορά των δεδομένων μέσα από τις μεταβλητές συστήματος, καταγράφοντας τα σημεία του πηγαίου κώδικα τα οποία παίρνουν τις τιμές αυτών των δεδομένων εισόδου. Τα συμπεράσματα από την παρακολούθηση των Μολυσμένων Αντικειμένων και των δεδομένων εισόδου αποθηκεύονται και δίνονται ως είσοδος στη μέθοδο εύρεσης λογικών αδυναμιών, η οποία τα συνδυάζει με τα υπόλοιπα αποτελέσματα από τις αναλύσεις των λογικών μεταβάσεων, διακλαδώσεων, λογικών κανόνων κτλ, αναζητώντας πιθανές λογικές αδυναμίες.

Προκειμένου να γίνει κατανοητή η λειτουργία της μεθόδου μολυσμένων αντικειμένων και της συνεργασίας αυτής με την διαδικασία λογικής ανάλυσης των διακλαδώσεων, ακολουθεί ένα απλό παράδειγμα. Έστω ότι τα δεδομένα ενός διανύσματος εισόδου αυθεντικοποίησης χρήστη (π.χ. το πεδίο ONOMA_ΧΡΗΣΤΗ ή αλλιώς USERNAME) αποθηκεύονται σε μια μεταβλητή συστήματος Χ. Το εργαλείο LogGIC θα ακολουθήσει τις χρήσεις της μεταβλητής Χ, καταγράφοντας τις συναρτήσεις και τις υπόλοιπες μεταβλητές που δέχονται τα δεδομένα που περιέχει η Χ. Ως εκ τούτου, αν μια μεταβλητή Α δεχτεί τα δεδομένα της Χ και μετά η Α χρησιμοποιηθεί ως παράμετρος στην εκτέλεση μιας μεθόδου, δημιουργώντας έτσι μια

αλληλουχία εκτελέσεων στο πρόγραμμα, τότε η X αλλά και η A -κατ' επέκταση- καταγράφονται ως πιθανές πηγές λογικής αδυναμίας. Αν μετά η λογική ανάλυση του LogGIC δείξει πως δεν υπάρχει κανένας έλεγχος πάνω σε αυτές τις μεταβλητές ο οποίος να ελέγχει την ορθότητα των δεδομένων που αυτές μεταφέρουν στις διάφορες λειτουργίες της εφαρμογής, σε κανένα στάδιο των λογικών βημάτων εκτέλεσης του προγράμματος που τις χρησιμοποιεί, τότε καταγράφεται μια ειδοποίηση προς τον αναλυτή πως το διάνυσμα εισόδου που αντιστοιχεί στην μεταβλητή X μπορεί να αποτελεί πηγή λογικού λάθους.

7.5 Η ΜΕΘΟΔΟΣ ΑΝΑΛΥΣΗΣ ΛΟΓΙΚΩΝ ΚΑΝΟΝΩΝ

Η τελευταία μέθοδος που υλοποιείται και παρουσιάζεται σε αυτή την διπλωματική είναι μια μέθοδος εύρεσης και αξιολόγησης λογικών αδυναμιών σε εφαρμογές γραφικής διεπαφής –και, υπό περιορισμούς, όχι μόνο-. Αποτελεί μια παραλλαγή σε μερικά σημεία της μεθόδου που υλοποιήθηκε από την ομάδα Ασφάλειας του Πανεπιστημίου της Καλιφόρνια και που παρουσιάζεται στην έρευνα «*Toward Automated Detection of Logic Vulnerabilities in Web Applications*» των Viktoria Felmetsger, Ludovico Cavedon, Christopher Kruegel και Giovanni Vigna. Στην παρούσα διπλωματική, υλοποιείται ένα υποσύνολο της έρευνας που διατέλεσε η ομάδα ασφαλείας του πανεπιστημίου της Καλιφόρνια, ως συμπλήρωμα της λειτουργίας που εκτελούν οι δύο άλλες μέθοδοι εύρεσης λογικών αδυναμιών. Τα αποτελέσματα αυτής της μεθόδου, όπως επίσης και τα αποτελέσματα των μεθόδων ανάλυσης μολυσμένων αντικειμένων και ευριστικής αξιολόγησης αποτελεσμάτων, αξιολογούνται στο τέλος της λειτουργίας της μεθόδου του εργαλείου LogGIC ως προς την ορθότητα και επικινδυνότητά τους με βάση ένα σύστημα Ασαφούς Λογικής.

Οι μέθοδος που παρουσίασε η ομάδα της Καλιφόρνια τροποποιείται προκειμένου να καλύψει την εύρεση λογικών αδυναμιών σε ένα διαφορετικό είδος εφαρμογών. Η έρευνα της ανωτέρω ομάδας στόχευε στην εύρεση λογικών αδυναμιών σε διαδικτυακές εφαρμογές που είχαν υλοποιηθεί πάνω σε Java Servlets και διαδικτυακά πρωτόκολλα. Η μέθοδος που υλοποιείται στα πλαίσια της διπλωματικής, αποτελεί μια προσπάθεια εύρεσης λογικών αδυναμιών σε *αλληλεπιδραστικά προγράμματα γραφικής διεπαφής χρήστη* (*Graphical User Interface applications - GUI*).

Στις εφαρμογές γραφικής διεπαφής δεν ορίζονται πάντα διακριτές ροές εκτέλεσης μεθόδων κώδικα προκειμένου να εκτελεστεί μια λειτουργία του προγράμματος από την αρχή μέχρι το τέλος. Δεν είναι, δηλαδή, πάντα δυνατή η εξαγωγή *μοντέλων ροής εντολών πεπερασμένης κατάστασης* (*finite-state models*). Η

χρήση *κεντρικών καταλόγων επιλογής λειτουργιών* (*Main menu*) και διαφόρων τύπων κουμπιών όπως 'Αποσύνδεση Χρήστη' ή 'Επαναφορά' οδηγούν στη δημιουργία άπειρων πιθανών μονοπατιών για το σύνολο των αλληλεπιδράσεων του χρήστη με την εφαρμογή, μονοπάτια τα οποία μπορούν να ανακυκλώνονται συνεχώς, δίχως να καταλήγουν σε μια τελική κατάσταση προγράμματος. Επίσης, οι έλεγχοι τέτοιων εφαρμογών γραφικής διεπαφής στηρίζονται εξ' ολοκλήρου στη χρήση μεθόδων τύπου «*ακροατών*» των ενεργειών του χρήστη (*input listeners*), γεγονός που δημιουργεί μια περίπλοκη δομή μεθόδων και κλάσεων, με συχνές αλληλεπιδράσεις του κώδικα με τις *διεπαφές προγραμματισμού εφαρμογής μηνυματοδοσίας* (*Application Programming Interface*) της γλώσσας Java. Η παραλλαγή της μεθόδου που αναπτύχθηκε και υλοποιήθηκε στη παρούσα διπλωματική αποτελεί μια προσπάθεια προσέγγισης και εντοπισμού των λογικών αδυναμιών και των σημείων στο κώδικα που αυτές εμφανίζονται.

Η μέθοδος που αναπτύχθηκε στα πλαίσια της διπλωματικής (όπως και με τις άλλες δύο μεθόδους) υλοποιήθηκε μέσα στο εργαλείο με όνομα **LogGIC**. Η βασική λειτουργία αυτής της μεθόδου είναι η εξαγωγή συμπερασμάτων τα οποία αποτελούν μια ένδειξη της λογικής με την οποία λειτουργεί η εφαρμογή υπό έλεγχο. Τα συμπεράσματα αυτά έχουν την μορφή πιθανών λογικών κανόνων ανάμεσα στις μεταβλητές της εφαρμογής υπό έλεγχο. Οι κανόνες αυτοί περιγράφουν τους τρόπους με τους οποίους δουλεύει το λογισμικό και εξάγονται με δύο τρόπους: Αφενός, μέσω της ανάλυσης των αποτελεσμάτων που παράγει μια μέθοδος Δυναμικής Ανάλυσης (Dynamic Analysis) κατά την διάρκεια της κανονικής λειτουργίας της εφαρμογής υπό έλεγχο. Αφετέρου, με την ανάλυση των διακλαδώσεων και των σημείων απόφασης που επηρεάζουν τα μονοπάτια εκτέλεσης της εφαρμογής μέσα από μια διαδικασία Στατικής ανάλυσης, η οποία παράγει τα πιθανά μονοπάτια εκτέλεσης κάθε λειτουργίας της εφαρμογής.

Ο λόγος για τον οποίο χρησιμοποιούνται οι ανωτέρω κανόνες από την Δυναμική Ανάλυση, είναι γιατί η (έστω και μερική) μοντελοποίηση της συμπεριφοράς μια εφαρμογής κατά την διάρκεια της κανονικής λειτουργίας της, επιτρέπει τον προσδιορισμό των χαρακτηριστικών της εφαρμογής, τα οποία μας ενδιαφέρουν για να χαρτογραφήσουμε τον τρόπο λειτουργίας της. Αυτό το βήμα είναι αναγκαίο προκειμένου να βρεθούν αυτόματα τα χαρακτηριστικά εκείνα τα οποία αντικατοπτρίζουν την λογική της λειτουργίας μιας συγκεκριμένης διαδικασίας. Αυτά τα χαρακτηριστικά, σε συνδυασμό με τα αποτελέσματα της Στατικής Ανάλυσης των αποφάσεων διακλάδωσης του κώδικα, αποτελούν τη πρώτη ύλη για την μετέπειτα εξερεύνηση του κώδικα της εφαρμογής υπό έλεγχο και την αναγνώριση των λογικών αδυναμιών της [2].

Η λειτουργία της συγκεκριμένης μεθόδου μέσα στο πρόγραμμα LogGIC στηρίζεται σε τρία διαφορετικά στάδια ανάλυσης των προγραμμάτων για την εύρεση λογικών αδυναμιών:

1. Στην Δυναμική εκτέλεση και ανάλυση (Dynamic execution and analysis) προκειμένου να παραχθούν οι δυναμικοί λογικοί κανόνες του κώδικα υπό έλεγχο,
2. Στη χρήση στατικής ανάλυσης του κώδικα, με έλεγχο μοντέλων *ροής εκτέλεσης* (*execution flow*) για την ανάλυση των δυναμικών κανόνων που παρήγαγε η δυναμική εκτέλεση (βήμα 1),
3. και στην επεξεργασία των μονοπατιών εκτέλεσης -που παρήγαγε η Στατική Ανάλυση- στα σημεία αποφάσεων ροής εκτέλεσης και εισαγωγής δεδομένων, προκειμένου να εντοπιστούν σημεία τα οποία έρχονται σε σύγκρουση με του λογικούς κανόνες, ένδειξη που αποτελεί χαρακτηριστικό των λογικών αδυναμιών.

Στο πρώτο βήμα, γίνεται χρήση του εργαλείου Δυναμικής Ανάλυσης *Daikon Invariant Detector* του Πανεπιστημίου του M.I.T.. Το συγκεκριμένο πρόγραμμα παράγει ένα σύνολο λογικών κανόνων που περιγράφουν την εφαρμογή υπό έλεγχο [17]. Το εργαλείο παρακολουθεί την εκτέλεση μιας εφαρμογής υπό έλεγχο, καθώς αυτή λειτουργεί με βάσει κάποια δεδομένα εισόδου με τα οποία την τροφοδοτεί ο χρήστης. Σημαντικό παράγοντα επιτυχίας της δυναμικής ανάλυσης του Daikon αποτελεί ο τρόπος χρήσης της εφαρμογής από τον αναλυτή κατά τη διαδικασία ελέγχου. Κατά την Δυναμική Ανάλυση, ο αναλυτής πρέπει να εκτελέσει τις λειτουργίες της εφαρμογής υπό έλεγχο με τρόπο τέτοιο που να μην αντιβαίνει στις προθέσεις του δημιουργού της. Οι πληροφορίες και οι κανόνες της ομαλής λειτουργίας ενός προγράμματος (είτε αυτό είναι δικτυακό όπως στην ανωτέρω έρευνα του πανεπιστημίου της Καλιφόρνια, είτε είναι πρόγραμμα με γραφική διεπαφή) δεν μπορούν να εξαχθούν αυτόματα με χρήση αυτοματοποιημένων εργαλείων δίχως την μεσολάβηση του χρήστη στις ενέργειες εισόδου, αφού τέτοιου είδους εργαλεία αυτοματοποίησης ενεργειών δεν αλληλεπιδρούν με τις εφαρμογές ακολουθώντας την αναμενόμενη ροή λειτουργίας που σκεφτόταν ο δημιουργός της εφαρμογής, ούτε χρησιμοποιούν δεδομένα εισόδου που αντιστοιχούν στα αναμενόμενα δεδομένα που περιμένει η εφαρμογή [2].

Το αποτέλεσμα της Δυναμικής Ανάλυσης του εργαλείου Daikon είναι η μερική χαρτογράφηση των χαρακτηριστικών της λογικής της εφαρμογής υπό έλεγχο και ο εντοπισμός πιθανών λογικών κανόνων που την διέπουν. Αυτοί οι κανόνες αναπαριστούν περιορισμούς διαφόρων μεταβλητών του κώδικα υπό έλεγχο σε διαφορετικά σημεία της εφαρμογής, καθώς και σχέσεις εξάρτησης μεταξύ των

μεταβλητών. Παρόλα αυτά, η περιορισμένη δυνατότητα εντοπισμού κανόνων από το Daikon καθώς και η δημιουργία κανόνων «μιας χρήσης» από το εργαλείο που δεν ισχύουν καθολικά για το πρόγραμμα υπό έλεγχο (αλλά είναι αποτέλεσμα της συγκεκριμένης εκτέλεσης που έγινε κατά την δυναμική ανάλυση), είναι δύο προβλήματα παραγωγής λανθασμένων κανόνων που χρίζουν αντιμετώπισης.

Ακολουθώντας την λογική της μεθόδου του πανεπιστημίου της Καλιφόρνια, υποθέτουμε ότι ένας λογικός κανόνας είναι ορθός αν υπάρχει τουλάχιστον ένα μονοπάτι εκτέλεσης που περιέχει έλεγχο μεταβλητής που επιβάλλει την ορθότητα του κανόνα. Αν ένας τέτοιος κανόνας επαληθεύεται σε ένα μονοπάτι εκτέλεσης αλλά δεν ισχύει σε κάποιο άλλο μονοπάτι με ίδια δεδομένα λειτουργίας που καταλήγει στο ίδιο αποτέλεσμα, τότε ο λογικός κανόνας επισημαίνεται ως πιθανό λογικό λάθος. Όταν ένας λογικός κανόνας παραβιάζεται πολλές φορές κατά την εκτέλεση, αλλά δεν υπάρχουν έλεγχοι στον κώδικα που να επαληθεύουν την ορθότητα του συγκεκριμένου κανόνα έστω και μία φορά, τότε αυτός ο κανόνας θεωρείται «μίας χρήσης» και τον αγνοούμε [2].

Στο δεύτερο βήμα της μεθόδου γίνεται χρήση ενός προγράμματος Στατικής Ανάλυσης της NASA που διατίθεται για ελεύθερη χρήση, του *Java PathFinder* (*JPF*). Πρόκειται για ένα εργαλείο στατικού ελέγχου εφαρμογών Java το οποίο, πέραν από τον έλεγχο του κώδικα, είναι ικανό να καταγράφει τα μονοπάτια εκτέλεσης της εφαρμογής, να παράγει μοντέλα λειτουργίας και να κοινοποιεί, εν τέλει, τις μεταβάσεις των καταστάσεων λειτουργίας που λαμβάνουν χώρα στην εκτέλεση του κώδικα κατά την διαδικασία στατικής ανάλυσης [18]. Η εφαρμογή LogGIC κάνει χρήση των αποτελεσμάτων του JPF προκειμένου να φιλτράρει τη λίστα κατηγοριοποιημένων δυναμικών κανόνων που παρήγαγε σε συνεργασία με το Daikon. Αυτή η διεργασία, σε συνδυασμό με την συσχέτιση των ανωτέρω αποτελεσμάτων με την λογική ανάλυση του εκτελέσιμου κώδικα υπό έλεγχο από το LogGIC, οδηγεί στην προσέγγιση των λογικών λαθών.

Για τις ανάγκες τις διπλωματικής, η βασική εφαρμογή του JavaPathfinder χρειάστηκε να παραμετροποιηθεί και να επεκταθεί με δύο επεκτάσεις, το *jpf-awt* και τους ωτακουστές του JPF, *ExecTracker* και *VarRecorder*. Το jpf-awt αποτελεί επέκταση του JPF που επιτρέπει την χρήση προδιαγεγραμμένων σεναρίων για την εκτέλεση ελέγχων σε εφαρμογές γραφικού περιβάλλοντος (GUI). Δυστυχώς, η υποστήριξη που παρέχει η επέκταση jpf-awt σε αντικείμενα γραφικών διεπαφών της γλώσσας Java είναι αρκετά περιορισμένη. Πολλά αντικείμενα γραφικού περιβάλλοντος της γλώσσας Java δεν υποστηρίζονται, γεγονός που δημιούργησε την ανάγκη παρέμβασης σε σημεία του κώδικα του jpf-awt. Συγκεκριμένα, στα πλαίσια της διπλωματικής ιχνηλατήθηκε και εντοπίστηκε ένα πρόβλημα που είχε η επέκταση jpf-awt με τα σύγχρονα αντικείμενα γραφικής απεικόνισης javax.swing της έκδοσης

1.6 της Java. Οι λεπτομέρειες του προβλήματος απεστάλησαν σε εκπονητή της εφαρμογής που ανήκει στην ομάδα υποστήριξης του JPF, ο οποίος εισήγαγε τις παρατηρήσεις μου στον κώδικα της εφαρμογής και διέθεσε την νέα έκδοση του κώδικα μαζί με την κανονική διανομή του προγράμματος.

Το τρίτο στάδιο της μεθόδου χρησιμοποιεί τα αποτελέσματα από την ευριστική μέθοδο ανάλυσης του κώδικα της εφαρμογής υπό έλεγχο από το LogGIC. Υπενθυμίζουμε πως η μέθοδος προσπελαύνει τον κώδικα και εντοπίζει τα σημεία εκείνα τα οποία λειτουργούν ως σημεία αποφάσεως της ροής εκτέλεσης, δηλαδή τα σημεία εκείνα στα οποία, ανάλογα με την εκάστοτε λογική της λειτουργίας, διακλαδώνουν την ροή εκτέλεσης μέσω κάποιων περιορισμών. Η λογική της συγκεκριμένης διαδικασίας παρουσιάστηκε πρωτύτερα σε κεφάλαιο της παρούσας διπλωματικής, στα πλαίσια προσέγγισης των λογικών επιλογών μέσα στο κώδικα.

Το στάδιο αυτό κάνει έντονη χρήση της πληροφορίας που υπάρχει στην σύνταξη των εντολών της γλώσσας προγραμματισμού JAVA. Ως εκ τούτου, ήταν αναγκαία η ανάλυση του κώδικα της εφαρμογής υπό έλεγχο σε επίπεδο μεταγλώττισης, στον πυρήνα του μεταγλωττιστή της JAVA. Λόγω της ανωτέρας ανάγκης απόκτησης των συγκεκριμένων πληροφοριών, υλοποιήθηκε και προστέθηκε στο LogGIC μια διαδικασία προσπέλασης και εξαγωγής πληροφοριών κατευθείαν από τον μεταγλωττιστή της Java. Το εργαλείο LogGIC αποκτά πρόσβαση στο δέντρο αφηρημένης σύνταξης το οποίο παράγεται από την μεταγλώττιση του πηγαίου κώδικα της εφαρμογής υπό έλεγχο. Έπειτα, εξάγει τις αναγκαίες πληροφορίες καλώντας τις συναρτήσεις μεταγλωττιστή και μετά κάνοντας προσωρινή επανεγγραφή (Override) της λειτουργίας τους. Με αυτό το τρόπο, εξάγονται όλες οι πληροφορίες για τις μεθόδους και τις μεταβλητές που υλοποιούνται στην εφαρμογή υπό έλεγχο από το LogGIC. Οι πληροφορίες αυτές αποθηκεύονται σε κωδικοποιημένη μορφή, αντίστοιχα στα δύο αρχεία *methods.loggic* και *variables.loggic* για λόγους ελέγχου.

Ακολουθώντας τα τρία προαναφερθέντα βήματα ανάλυσης, είναι δυνατόν να εντοπιστούν από την εφαρμογή LogGIC ένα υποσύνολο του συνόλου των πιθανών λογικών αδυναμιών που εμφανίζονται σε εφαρμογές. Πρέπει να επισημανθεί πως η προσέγγιση της συγκεκριμένης διπλωματικής δεν είναι πλήρης ως προς την κάλυψη του συνόλου των λογικών αδυναμιών.

7.6 ΕΡΓΑΛΕΙΑ ΥΛΟΠΟΙΗΣΗΣ ΤΗΣ ΜΕΘΟΔΟΥ

Η υλοποίηση του εργαλείου LogGIC που αναπτύχθηκε στα πλαίσια της παρούσας διπλωματικής έγινε στη γλώσσα προγραμματισμού Java. Ακολουθεί παρουσίαση των εργαλείων που ενσωματώθηκαν στο LogGIC και των οποίων τα χαρακτηριστικά παρουσιάστηκαν σε προηγούμενο κεφάλαιο της διπλωματικής :

7.6.1 ΕΞΑΓΩΓΗ ΚΑΝΟΝΩΝ ΜΕ ΤΟ DAIKON

Το Daikon παράγει πιθανούς λογικούς κανόνες κάνοντας χρήση των αποδεκτών αναγνωριστικών από τα ίχνη που παράγει η εκτέλεση της εφαρμογής υπό έλεγχο μέσω του Daikon. Μια επεξεργασμένη έκδοση των αποτελεσμάτων του Daikon μπορεί να βρει κανείς, αφού εκτελέσει την ανάλυση ενός προγράμματος με το LogGIC, στο φάκελο της εφαρμογής υπό έλεγχο. Το *αρχείο καταγραφής δραστηριοτήτων* (*application log*) έχει όνομα 'daikonFiltered.txt'. Το εργαλείο Daikon είναι ικανό να παράγει μια ευρεία ποικιλία λογικών κανόνων που καλύπτουν από μεταβλητές της εφαρμογής μέχρι λογικές σχέσεις μεταξύ των αναγνωρισμένων μεταβλητών. Οι παραγόμενοι κανόνες του Daikon ονομάζονται πιθανοί κανόνες, αφού πρέπει πρώτα να φιλτραριστούν με βάση το ενδιαφέρον τους και να διαγραφούν οι κανόνες «μιας χρήσης» που εμφανίζονται στα αποτελέσματα και δεν αποτελούν ορθή χαρτογράφηση της λογικής των μεταβλητών.

Το εργαλείο υποστηρίζει μια πληθώρα διεπαφών αλληλεπίδρασης. Κάθε διεπαφή χειρίζεται μια διαφορετική γλώσσα προγραμματισμού, όπως C και Java. Στην πρώτη φάση εκτέλεσης, οι διεπαφές παράγουν αρχεία ιχνηλάτησης της εφαρμογής υπό έλεγχο, καθώς ο χρήστης εκτελεί τις 'νόμιμες' λειτουργίες της εφαρμογής διαμέσου του Daikon. Στη συνέχεια, τα αρχεία αυτά αποστέλλονται στο πυρήνα του Daikon για την παραγωγή των λογικών κανόνων [17]. Στα πλαίσια της διπλωματικής, χρησιμοποιήθηκε η διεπαφή του Daikon για την γλώσσα προγραμματισμού Java με το όνομα Chicory.

Στα πλαίσια φιλτραρίσματος των λογικών κανόνων που δεν αφορούν το πρόγραμμα υπό έλεγχο, το LogGIC πραγματοποιεί και μια ομαδοποίηση και επισήμανση των λογικών κανόνων του Daikon που αφορούν μεταβλητές που παίρνουν δεδομένα από πηγές εισόδου πληροφοριών του χρήστη. Ακολουθώντας την λογική των μολυσμένων δεδομένων εισόδου, το LogGIC χρησιμοποιεί του κανόνες και τα μονοπάτια εκτέλεσης του JPF, προκειμένου να προσδιορίσει την επικινδυνότητα των συγκεκριμένων μεταβλητών στην εφαρμογή, σε σχέση πάντα με το τρόπο προσπέλασής τους από το πρόγραμμα υπό έλεγχο.

Ένα *σημείο προγράμματος* (*program point*) του κώδικα εκτέλεσης είναι ένα συγκεκριμένο σημείο στον πηγαίο κώδικα, όπως ακριβώς πριν από μια συγκεκριμένη γραμμή εντολής [17]. Η έξοδος αποτελεσμάτων του Daikon είναι οργανωμένη σε τέτοια σημεία προγράμματος για καλύτερη κατανόηση των αποτελεσμάτων. Παραδείγματος χάρη, μια καταχώρηση στους λογικούς κανόνες με την μορφή **foo( ) ::: ENTER** είναι το σημείο της εισόδου στην μέθοδο foo( ) κατά την εκτέλεση. Οι λογικοί κανόνες σε αυτό το σημείο είναι η προϋποθέσεις έναρξης της μεθόδου foo( ), χαρακτηριστικά τα οποία είναι πάντα αληθή κατά την εκτέλεση της μεθόδου. Αντίστοιχα, μια καταχώρηση τύπου foo( ) ::: EXIT είναι το σημείο προγράμματος στο οποίο υπάρχει έξοδος από την δεδομένη μέθοδο, και οι λογικοί κανόνες εκεί είναι οι μετέπειτα όροι λειτουργίας του προγράμματος. Όταν υπάρχουν πολλαπλά σημεία εξόδου από μια μέθοδο, οι διαφορετικές αυτές έξοδοι διαφοροποιούνται με το να τοποθετείται στο τέλος της καταχώρησης ο αριθμός γραμμής πηγαίου κώδικα στον οποίο αυτές εμφανίζονται (παράδειγμα, StackAr.top( ):::EXIT79) [17].

Στις αδυναμίες του Daikon συγκαταλέγεται η ανικανότητά του να εντοπίσει λογικούς κανόνες για τους βρόχους επεξεργασίας δεδομένων που εμφανίζονται στο κώδικα. Το εργαλείο μπορεί να εντοπίσει λογικούς κανόνες σε οποιαδήποτε περιοχή του προγράμματος, εκτός από τους βρόχους. Οι διεπαφές όπως το Chicory δεν τροφοδοτούν το Daikon με μεταβλητές που λειτουργούν ως περιορισμοί βρόχων. Ως εκ τούτου, η εύρεση λογικών λαθών σε βρόχους περιορίζεται σημαντικά καθώς δεν υπάρχει πρώτη ύλη για ανάλυση της λογικής τους.

7.6.2 ΠΑΡΑΓΩΓΗ ΜΟΝΟΠΑΤΙΩΝ ΕΚΤΕΛΕΣΗΣ ΤΗΣ ΣΤΑΤΙΚΗΣ ΑΝΑΛΥΣΗΣ ΚΑΙ ΤΟ ΕΡΓΑΛΕΙΟ NASA JAVA PATHFINDER (JPF)

Εφόσον παραχθούν οι λογικοί κανόνες από το Daikon, ακολουθεί η στατική ανάλυση της εφαρμογής υπό έλεγχο, προκειμένου να εντοπιστούν οι απαραίτητες καταστάσεις μεταβλητών και να παραχθούν τα μονοπάτια εκτέλεσης της εφαρμογής. Το LogGIC εκτελεί το JPF εσωτερικά, παράγοντας τα επιθυμητά αποτελέσματα και αποθηκεύοντάς τα στο φάκελο της εφαρμογής υπό έλεγχο. Τα στοιχεία των μονοπατιών εκτέλεσης καθώς και οι διάφορες καταχωρήσεις μεταβάσεων θα χρησιμοποιηθούν από το LogGIC προκειμένου να εντοπιστούν λογικοί κανόνες που παραβιάζεται η ορθότητά τους κατά την εκτέλεση της εφαρμογής. Κάθε τέτοια παραβίαση επισημαίνεται ως πιθανό λογικό λάθος και οι μεταβλητές που την διέπουν παίρνουν ένα συγκεκριμένο βάρος στη κλίμακα Likert, ανάλογα με το μέγεθος της επικινδυνότητάς τους μέσα στο πρόγραμμα.

Παρόμοια με την μέθοδο του πανεπιστημίου της Καλιφόρνια, το LogGIC παίρνει ως είσοδο τους λογικούς κανόνες του Daikon και αναλύει στατικά την εφαρμογή υπό έλεγχο. Μετά το πέρας της ανάλυσης, ελέγχεται η ορθότητα των λογικών κανόνων με το να συσχετιστεί η ορθότητά τους με τα παραγόμενα μονοπάτια εκτέλεσης του κώδικα της εφαρμογής.

Το JPF είναι ένα πρόγραμμα στατικής ανάλυσης και ελέγχου μοντέλων συστήματος που υλοποιεί μια ξεχωριστή *ιδεατή μηχανή εκτέλεσης Java* (*Java Virtual Machine*). Βασική του λειτουργία είναι η εξερεύνηση και περιήγηση των καταστάσεων λειτουργίας μιας εφαρμογής, μέσω της εκτέλεσης του *δυαδικού της κώδικα* (*bytecode*).

Το JPF μπορεί να παραμετροποιηθεί σε μεγάλο βαθμό και δέχεται μια πληθώρα επεκτάσεων που αναπτύσσονται από μεμονωμένες ή ακαδημαϊκές ομάδες που χρησιμοποιούν το εργαλείο. Επίσης, υποστηρίζει μια πληθώρα διεπαφών οι οποίες επιτρέπουν την επέκταση της λειτουργικότητάς του και σε είδη κώδικα που δεν είναι σχεδιασμένο να αναλύει. Με κάποιες εξαιρέσεις, το JPF μπορεί να εκτελέσει οποιοδήποτε αρχείο κώδικα Java το οποίο δεν στηρίζεται σε συγκεκριμένες, ιδιάζουσες βιβλιοθήκες κώδικα που έχουν αναπτυχθεί ανεπίσημα και εκτός της Oracle. Αντίθετα, πολλές βιβλιοθήκες της Java υποστηρίζονται απευθείας από το JPF χωρίς να χρειάζεται καμία παρέμβαση για την εκτέλεσή τους [2]. Παρόλα αυτά, αρκετές βασικές βιβλιοθήκες και κλάσεις κώδικα της Java έχουν αντικατασταθεί με διαφορετικές, απλούστερες εκδόσεις από τους δημιουργούς του JPF, προκειμένου να μειωθεί η πολυπλοκότητα χρήσης του εργαλείου, γεγονός που συχνά οδηγεί σε γενικές ασυμβατότητες του εργαλείου με πολύπλοκα ή πολύ σύγχρονα προγράμματα.

Η εκτέλεση του εργαλείου JPF στηρίζεται σε μια πληθώρα παραμετροποιημένων χαρακτηριστικών τα οποία ενδεχομένως να χρειαστούν κάποιο χρόνο προκειμένου το εργαλείο να δουλέψει σωστά. Η βιβλιογραφία και η γραπτή καθοδήγηση των χρηστών όσων άφορα το εργαλείο είναι σημαντικά περιορισμένη, γεγονός που καθυστερεί σημαντικά την τοποθέτηση και κατανόηση του τρόπου λειτουργίας του εργαλείου. Παρόλα αυτά, η ομάδα υποστήριξης που αναπτύσσει ακόμα το JPF απαντά σε ερωτήσεις τεχνικής υποστήριξης μέσω ενός καταλόγου αποδεκτών υποστήριξης της Google: *http://groups.google.com/group/java-pathfinder?pli=1*.

Τα μονοπάτια εκτέλεσης που παράγει το JPF χρησιμοποιούνται προκειμένου να εντοπιστούν οι έλεγχοι ροής εκτέλεσης (όπως οι διακλαδώσεις). Κάθε λογικός κανόνας που υποστηρίζεται από ένα σύνολο ελέγχων ενός μονοπατιού εκτέλεσης επισημαίνεται ως ορθός κανόνας για την συγκεκριμένη μεταβλητή. Ένα μονοπάτι εκτέλεσης κατά το οποίο παραβιάζεται ένας ορθός λογικός κανόνας επισημαίνεται επίσης. Όταν ένας λογικός κανόνας ισχύει για όλα τα μονοπάτια εκτέλεσης, τότε

ξέρουμε πως ο λογικός κανόνας ισχύει και είναι ορθός, δίχως να υπάρχει λογικό λάθος. Όταν όλα τα μονοπάτια εκτέλεσης παραβιάζουν ένα κανόνα, τότε υποθέτουμε πως ο κανόνας αυτός είναι «μιας χρήσης» και τον απορρίπτουμε. Ένας κανόνας που επιβεβαιώνεται σε ένα μονοπάτι εκτέλεσης αλλά κάπου αλλού παραβιάζεται, επισημαίνεται ως λογικό λάθος [2].

7.7 ΠΡΟΣΠΕΛΑΣΗ ΔΕΝΤΡΟΥ ΜΕΤΑΓΛΩΤΤΙΣΗΣ ΚΑΙ Ο JAVAC

Ο μεταγλωττιστής της γλώσσας προγραμματισμού Java με το όνομα Javac διαβάζει αρχεία πηγαίου κώδικα που είναι γραμμένα σ' αυτή τη γλώσσα και τα μεταγλωττίζει σε *αρχεία κλάσεων δυαδικού κώδικα* (*bytecode class files*). Πέραν αυτού, ο μεταγλωττιστής είναι υπεύθυνος και για την επεξεργασία των σχολίων του κώδικα, κάνοντας χρήση της πρόσθετης *Διεπαφής προγραμματισμού Επεξεργασίας Σχολίων* (*Pluggable Annotation Processing API*). Περιληπτικά, ο μεταγλωττιστής της Java δεν είναι τίποτα περισσότερο από ένα εργαλείο το οποίο μπορεί να λειτουργήσει τόσο μέσω γραμμής εντολών, όσο και μέσω κλήσεων από διάφορα προγράμματα Java που χρειάζονται τις ικανότητες του. Ο μεταγλωττιστής δέχεται ως είσοδο πηγαίο κώδικα, τα πρότυπα του οποίου έχουν καθοριστεί από τις προδιαγραφές της γλώσσας προγραμματισμού Java (Java Language Specification - JLS). Ως έξοδο, παράγει αρχεία κλάσεων επίσης στα πρότυπα που ορίζει η γλώσσα Java, μέσω των προδιαγραφών της *ιδεατής μηχανής της Java* (*Java Virtual Machine Specification - JVMS*).

Επιπρόσθετα με τη λειτουργία μεταγλώττισης πηγαίου κώδικα, ο μεταγλωττιστής Javac υλοποιεί και παρέχει διάφορες διεπαφές προγραμματισμού τα οποία επιτρέπουν την άμεση ή έμμεση χρήση των συστατικών και των λειτουργιών του από εξωτερικά προγράμματα Java καθώς και την εις βάθους ανάλυση του πηγαίου κώδικα που μεταγλωττίζεται από τον μεταγλωττιστή. Μία από τις διεπαφές που παρέχονται είναι η πρόσβαση και διαχείριση των πληροφοριών του *Συντακτικού Δέντρου Μεταγλώττισης* (*Java Compiler Tree*). Η διεπαφή του δέντρου μεταγλώττισης είναι ένα σύνολο διαφόρων διεπαφών οι οποίες αναπαριστούν τον πηγαίο κώδικα που μεταγλωττίζεται σε δυαδικό ως ένα δένδρο αφηρημένης σύνταξης (Abstract Syntax Tree – AST). Η ανάλυση και ο κατακερματισμός του πηγαίου κώδικα που γίνονται κατά την διαδικασία μεταγλώττισης από τον Javac, δημιουργούν και χρησιμοποιούν ένα δέντρο αφηρημένης σύνταξης, με το οποίο αναλύουν τον πηγαίο κώδικα ως ένα δέντρο με κόμβους. Κάθε κόμβος αναπαριστά ένα τμήμα του πηγαίου κώδικα (για

παράδειγμα, ο κόμβος “IfStatement” αναπαριστά την ύπαρξη μιας διακλάδωσης τύπου ‘IF’ στον κώδικα. Τα φύλλα του κόμβου αυτού θα είναι οι περιορισμοί της διακλάδωσης καθώς και οι λογικές σχέσεις μεταξύ τους). Το αφηρημένο δέντρο σύνταξης ορίζει περισσότερες από 60 AST υποκλάσεις με την μορφή κόμβων, καθένας εκ των οποίων αναπαριστά διαφορετικά στοιχεία της γλώσσας προγραμματισμού Java [21].

Για τις ανάγκες τις μεθόδου εύρεσης λογικών αδυναμιών που αναπτύχθηκε, υλοποιήθηκε και εφαρμόζεται στο εργαλείο LogGIC μια κλάση και τέσσερις μέθοδοι επεξεργασίας και ανάλυσης του δέντρου αφηρημένης σύνταξης. Οι μέθοδοι του LogGIC επεμβαίνουν απευθείας πάνω στον μεταγλωττιστή της Java, κάνοντας χρήση της διεπαφής παραγωγής και ανάλυσης του δέντρου αφηρημένης σύνταξης. Σκοπός του εργαλείου LogGIC είναι η γρήγορη συλλογή των αναγκαίων πληροφοριών του κώδικα της εφαρμογής υπό έλεγχο. Κατά την διαδικασία προσπέλασης του μεταγλωττιστή της Java, παράγονται δύο αρχεία του εργαλείου LogGIC τα οποία μπορούν να βρεθούν στον φάκελο με τα αρχεία της εφαρμογής υπό έλεγχο, που επιλέγεται στο πρώτο βήμα λειτουργίας του εργαλείου. Τα αρχεία έχουν ονόματα *variables.loggic* και *methods.loggic*, περιέχοντας όλες τις πληροφορίες του μεταγλωττιστή για τις μεταβλητές και τις μεθόδους που υλοποιούνται στην εφαρμογή υπό έλεγχο. Αυτές οι πληροφορίες συνδυάζονται με τις πληροφορίες που εξάγει η λογική ανάλυση του LogGIC καθώς και με τις πληροφορίες που εξάγουν τα εργαλεία δυναμικής και στατικής ανάλυσης προκειμένου να αποδειχθούν αληθή ή ψευδή τα συμπεράσματα αυτών των διαδικασιών ανάλυσης.

Οι πληροφορίες αυτές παρέχονται κατευθείαν από τον πυρήνα της γλώσσας προγραμματισμού Java, κάνοντας κλήση απευθείας στις διεπαφές παραγωγής του δέντρου σύνταξης κατά το χρόνο λειτουργίας του μεταγλωττιστή. Η ανάλυση της δομής της εφαρμογής υπό έλεγχο υλοποιήθηκε με την ανωτέρω διαδικασία επειδή, με αυτό το τρόπο, δεν επιβαρύνεται καθόλου η λειτουργία του LogGIC. Η επεξεργασία του κώδικα γίνεται εξωτερικά των λειτουργιών του μέσω της λειτουργίας του μεταγλωττιστή Javac, γεγονός που επιταχύνει σημαντικά την συνολική διαδικασία λειτουργίας του εργαλείου.

Η υλοποίηση της διαδικασίας εξαγωγής πληροφοριών από το δέντρο αφηρημένης σύνταξης μπορεί να βρεθεί στο πηγαίο κώδικα στη κλάση *MethodFinder.java*.

7.8 ΑΣΑΦΗΣ ΛΟΓΙΚΗ

Η *Ασαφής Λογική* (*Fuzzy Logic*) είναι μια μορφή λογικής αναπαράστασης που κάνει χρήση πολλαπλών πιθανών αποτελεσμάτων στις μεταβλητές του συστήματος. Προέρχεται από την Θεωρεία Ασαφών Συνόλων (Fuzzy Set Theory) και αναλύει θέματα αντίληψης της κατάστασης ενός συστήματος, προσεγγίζοντας τα αποτελέσματα περισσότερο προσεγγιστικά παρά με διακριτή, αριθμητική ακρίβεια. Στην Ασαφή Λογική γίνεται χρήση *γλωσσικών μεταβλητών* (*linguistic variables*) αντί για μαθηματικές μεταβλητές, όπως για παράδειγμα: Θερμοκρασία, Αντίσταση, Επικινδυνότητα κ.ά..

Η θεωρεία Ασαφών Συνόλων ορίζει κανόνες χειρισμού πάνω στα ασαφή σύνολα που μελετά. Συχνά οι κανόνες ορίζονται προσεγγιστικά, καθώς δεν υπάρχουν διακριτές τιμές προσδιορισμού των κανόνων. Γι αυτό το λόγο, η Ασαφής Λογική χρησιμοποιεί κανόνες τύπου IF-THEN στους αλγορίθμους. Οι κανόνες αυτοί εκφράζονται συχνά υπό την μορφή :

IF μεταβλητή IS τιμή THEN δράση

7.8.1 ΑΠΟΣΑΦΗΝΙΣΗ ΑΠΟΤΕΛΕΣΜΑΤΩΝ ΑΣΑΦΟΥΣ ΛΟΓΙΚΗΣ

Προκειμένου να υπάρξουν προσδιορίσιμα αποτελέσματα από την Ασαφή Λογική, γίνεται συχνά χρήση *Αποσαφήνισης* (*Defuzzification*). Αποσαφήνιση στη θεωρία Ασαφών συνόλων, είναι η διαδικασία παραγωγής ενός μετρήσιμου αποτελέσματος μέσα από την Ασαφή Λογική. Χρησιμοποιείται εκτενώς στα συστήματα ποσοτικών ελέγχων που στηρίζονται στα αποτελέσματα της Ασαφούς Λογικής και, γι αυτό το λόγο, χρησιμοποιείται και στην παρούσα διπλωματική. Η Αποσαφήνιση αποδέχεται ως είσοδο ένα σύνολο κανόνων που μεταφράζουν ένα αριθμό μεταβλητών σε ασαφή αποτελέσματα (δηλαδή αποτελέσματα που ορίζονται με βάση την ποσοστιαία μονάδα). Παραδείγματος χάριν, ένα σύστημα ασαφούς λογικής για έλεγχο παροχής ενέργειας θα είχε αποτελέσματα της μορφής «Χαμήλωσε Σημαντικότητα 20%» ή Διατήρησε Σημαντικότητα 50%» κ.τ.λ.). Η Αποσαφήνιση μεταφράζει τα ασαφή αποτελέσματα σε συγκεκριμένες τιμές και συγκεκριμένα αποτελέσματα.

Υπάρχει μια πληθώρα μαθηματικών μεθόδων Αποσαφήνισης, ανάλογα με τις ανάγκες του συστήματος Ασαφούς Λογικής που λειτουργεί κάθε φορά. Μια

συνηθισμένη και αποτελεσματική μέθοδος που υλοποιείται και στα αποτελέσματα του εργαλείου LogGIC, είναι η χρήση της μεθόδου *Κέντρου Βαρύτητας* (*Center of Gravity*).

7.8.1.1 Η ΜΕΘΟΔΟΣ ΑΠΟΣΑΦΗΝΙΣΗΣ «ΚΕΝΤΡΟΥ ΒΑΡΥΤΗΤΑΣ»

Η μέθοδος κέντρου βαρύτητας υπολογίζει την διακριτή τιμή των μεταβλητών εξόδου παίρνοντας την τομή του συνδυασμού των γεωμετρικών σχημάτων που καθορίζουν το εύρος των τιμών των μεταβλητών εισόδου στο σύστημα.

Αρχικά τα αποτελέσματα των κανόνων ασαφούς λογικής του LogGIC προστίθενται για να δημιουργήσουν μια ενιαία τιμή. Στην συνέχεια, η διαδικασία αποσαφήνισης δέχεται τα παραγόμενα τρίγωνα συμμετοχής ενός αποτελέσματος με βάση το μερίδιό του στο σύνολο (το τρίγωνο είναι η συνηθέστερη μορφή αναπαράστασης αποτελεσμάτων ασαφούς λογικής του τύπου: «Αν Α = Χ τότε Αύξησε 15%») και δημιουργεί τραπεζοειδή ή τριγωνικά σχήματα συμμετοχής κάθε αποτελέσματος στο σύνολο. Για παράδειγμα, αν η έξοδος ενός συστήματος Ασαφούς Λογικής ήταν «Μείωσε Παροχή 30%», τότε το τρίγωνο αναπαράστασης αυτής της τιμής θα τεμαχιζόταν στα δύο, 30% από την βάση του.

Στη συνέχεια, όλα τα τραπεζοειδή που παράγονται για κάθε αποτέλεσμα τοποθετούνται μαζί στο ίδιο διάγραμμα (υπέρθεση αποτελεσμάτων), δημιουργώντας ένα ενιαίο γεωμετρικό σχήμα.

Τέλος, η διαδικασία παίρνει το κέντρο των τιμών που αναπαριστά το τραπεζοειδές. Το κέντρο αυτό ονομάζεται *Ασαφές Κέντρο Τιμών* (*Fuzzy Centroid*). Η συντεταγμένη του άξονα Χ είναι η τιμή Αποσαφήνισης των αποτελεσμάτων.

7.8.2 ΑΣΑΦΗΣ ΛΟΓΙΚΗ ΚΑΙ ΕΡΓΑΛΕΙΟ LOGGIC

Προκειμένου να υπάρξει καλύτερη αξιοποίηση των αποτελεσμάτων από τον αναλυτή, το LogGIC κάνει χρήση ενός τέτοιου μοντέλου Ασαφούς Λογικής που αναφέραμε παραπάνω, προκειμένου να ορίσει διακριτούς κανόνες που δρουν ως διαχειριστές του μεγέθους κινδύνου των σημείων κώδικα στα οποία ενδεχομένως να έχουν εντοπιστεί αδυναμίες. Το μοντέλο χρησιμοποιείται προκειμένου να προσδιοριστεί η Κρισιμότητα των πιθανών λογικών αδυναμιών που έρχονται στην επιφάνεια από το εργαλείο LogGIC. Η διαδικασία κανόνων Ασαφούς Λογικής καθώς και η Αποσαφήνιση των αποτελεσμάτων με χρήση Κέντρου Βαρύτητας, υλοποιούνται ως τμήμα του LogGIC με την βοήθεια της βιβλιοθήκης κανόνων JFuzzyLogic για την γλώσσα JAVA [20].

Η κατηγοριοποίηση των ευρημάτων ορίζεται από τους κανόνες τύπου IF-THEN, μέσω μιας πενταβάθμιας κλίμακας Likert. Στο μοντέλο που υλοποιείται γίνεται χρήση δύο διανυσματικών μεταβλητών. Αρχικά, σε κάθε σημείο στον κώδικα που μπορεί να περιέχει λογικό λάθος, προσάπτεται ένας βαθμός Δριμύτητας από το ένα έως το πέντε. Για να γίνει κατανοητή η παραπάνω λειτουργία, ακολουθεί μια ομάδα κανόνων όπως αυτοί ορίζονται μέσα στη λογική του LogGIC. Η μεταβλητή Δριμύτητας (Severity) που φαίνεται στην ακόλουθη αναπαράσταση είναι μια τιμή που προσδιορίζει κατά πόσον επικίνδυνο είναι ένα σημείο του κώδικα και στο οποίο εφαρμόζονται οι κανόνες Fuzzy Logic. Για την τελική τιμή της κάθε μεταβλητής χρησιμοποιείται η προσέγγιση χειρότερης δυνατής περίπτωσης (worst-case scenario):

ΒΗΜΑ 1: ΥΠΟΛΟΓΙΣΜΟΣ ΒΑΣΙΚΗΣ ΔΡΙΜΥΤΗΤΑΣ (BASIC SEVERITY)

- Δριμύτητα *τυχαίας μεταβλητής*: **severity = 1**
- Δριμύτητα *μεταβλητής* με τιμή από *Δεδομένα εισόδου*: **severity = 2**
- Δριμύτητα *μεταβλητής* με τιμή *από διακλάδωση 'if':* **severity = 3**
- Δριμύτητα *μεταβλητής* με τιμή *από διακλάδωση* '*switch'*: **severity = 3**

ΒΗΜΑ 2: ΥΠΟΛΟΓΙΣΜΟΣ ΕΚΤΕΤΑΜΕΝΗΣ ΔΡΙΜΥΤΗΤΑΣ (EXTENDED SEVERITY)

- Δριμύτητα *μεταβλητής με τιμή από 'if'* και *εμφάνιση >= 2 φορές*: **severity = 4**
- Δριμύτητα *μεταβλητής με τιμή από 'switch'* και *εμφάνιση >= 2 φορές*: **severity = 4**
- Δριμύτητα *μεταβλητής* με τιμή από *Δεδομένα εισόδου* που *χρησιμοποιείται* ***ταυτόχρονα*** και σε *Διακλάδωση 'if' ή 'switch'*: **severity = 5**

ΒΗΜΑ 3: ΚΑΤΑΧΩΡΗΣΗ ΤΕΛΙΚΗΣ ΔΡΙΜΥΤΗΤΑΣ

Η δεύτερη μεταβλητή που χρησιμοποιείται ως είσοδος στο σύστημα Ασαφούς Λογικής που χρησιμοποιεί το LogGIC, είναι μια κατηγοριοποίηση του επιπέδου των αδυναμιών που βρέθηκαν από το εργαλείο μας. Προφανώς χρησιμοποιείται και στην δεύτερη μεταβλητή μια πενταβάθμια κλίμακα Likert για τον προσδιορισμό των αδυναμιών. Σύμφωνα με την μέθοδο που αναπτύχθηκε και υλοποιείται στο LogGIC, υπάρχουν αδυναμίες που είναι σίγουρα υπαρκτές και ιδιαίτερα επικίνδυνες για το λογισμικό υπό έλεγχο. Σ' αυτές τις αδυναμίες προσάπτεται ένας βαθμός 5 της κλίμακας. Κάποια άλλα αποτελέσματα εμφανίζουν στοιχεία που τα συνδέουν με αδυναμίες, αλλά δεν είναι απόλυτα σίγουρο ότι μπορούν να χρησιμοποιηθούν από ένα κακόβουλο χρήστη προκειμένου να δημιουργήσουν πρόβλημα στην εφαρμογή. Ως εκ τούτου, τέτοια είδη αδυναμιών παίρνουν χαμηλότερη τιμή στην κλίμακα 1-5, ανάλογα

με τον τρόπο εμφάνισής τους. Για παράδειγμα, ένα διάνυσμα εισόδου μπορεί να δέχεται δεδομένα από τον χρήστη και να μην κάνει κάποιο έλεγχο στα δεδομένα αυτά. Παρόλα αυτά όμως, μπορεί να διαχέει τα δεδομένα σε διάφορες μεθόδους της εφαρμογής, μερικές εκ των οποίων κάνουν ελέγχους των δεδομένων. Τέτοιου είδους κώδικας θα ενεργοποιήσει το LogGIC ως πιθανό λογικό λάθος, άλλα –ανάλογα τα συμπεράσματα που θα βγάλει το εργαλείο ακολουθώντας τα δεδομένα- θα προσάψει μια τιμή ανάμεσα από 2 και 4 στη συγκεκριμένη αδυναμία, ανάλογα με τους ελέγχους που γίνονται βαθύτερα στην εφαρμογή υπό έλεγχο από τις μεθόδους της.

Ακολουθεί μια ανάλυση του πώς λειτουργεί το σύνολο κανόνων JFuzzyLogic και του πώς έχουν οργανωθεί οι κανόνες μέσα στο εργαλείο LogGIC, σύμφωνα με την δομή και την υλοποίηση που παρέχεται από την βιβλιοθήκη JFuzzyLogic [20]:

Ο κορμός που αποτελεί την βάση του συστήματος Ασαφούς Λογικής που κάνει χρήση το εργαλείο LogGIC, υλοποιείται μέσα στο αρχείο *base.fcl* που βρίσκεται στο φάκελο του εργαλείου. Το αρχείο .fcl περιέχει ουσιαστικά ένα σύστημα υπολογισμού Ασαφής Λογικής. Η διαδικασία που περιγράφει το αρχείο, κάνει χρήση ενός ή περισσοτέρων FunctionBLocks της μορφής:

```
FUNCTION_BLOCK functionBlock1
...
END_FUNCTION_BLOCK

FUNCTION_BLOCK functionBlock2
...
END_FUNCTION_BLOCK
```

Κάθε FunctionBlock συντίθεται από ένα ή περισσότερα RuleBlocks κώδικα και από κάποιες μεταβλητές, καθώς επίσης από τους κανόνες Ασάφειας και Αποσαφήνισης των αποτελεσμάτων, π.χ. :

```
FUNCTION_BLOCK functionBlockName
VAR_INPUT
...
END_VAR

VAR_OUTPUT
...
END_VAR

FUZZIFY inputVariable
...
END_FUZZIFY

DEFUZZIFY outputVariable
...
END_DEFUZZIFY

RULEBLOCK No1
...
END_RULEBLOCK

END_FUNCTION_BLOCK
```

Κάθε σύνολο κανόνων συντίθεται από ένα μέρος κανόνων της μορφής IF και από ένα ακόλουθο μέρος κανόνων της μορφής THEN. Ένα παράδειγμα του δημιουργού της βιβλιοθήκης είναι το ακόλουθο :

RULE 1: IF Severity IS small OR Vulnerability IS trifle THEN Criticality IS trifle;

Προϋπόθεση: " Severity IS small OR Vulnerability IS trifle "

Συνέπεια: " Criticality IS trifle ".

Κάθε κανόνας τύπου RuleTerm κάνει χρήση μιας μεταβλητής και ενός γλωσσικού ονόματος (LinguisticTermName, στην δομή της βιβλιοθήκης). Παραδείγματος χάριν:

Severity IS small *Μεταβλητή*: severity

Γλωσσική τιμή μεταβλητής: 'small'

Σύνδεσμος: 'IS'

Το όνομα κάθε μεταβλητής που αποτελεί μεταβλητή εισόδου για το σύστημα, χρησιμοποιείται προκειμένου να οριστούν τα ποσοστιαία όρια Ασάφειας της μεταβλητής. Μέσα στο αρχείο .fcl, η βιβλιοθήκη περιμένει να βρει εγγραφές της μορφής:

```
FUZZIFY Severity
   TERM small := (0, 1) (2, 0) ;
   TERM medium := (1, 0) (2, 1) (3, 1) (5,0);
   TERM extended := (3.5, 0) (5, 1);
END_FUZZIFY
```

Αντίστοιχα για τις μεταβλητές εξόδου, οι οποίες υπόκεινται στην διαδικασία αποσαφήνισης, η βιβλιοθήκη αναμένει να βρει πληροφορίες της ακόλουθης μορφής στο αρχείο .fcl, προκειμένου να ορίσει τα σύνολα Ασάφειας:

```
// Defuzzify output variable 'Risk'
DEFUZZIFY Criticality

        TERM trifle := (0,0) (2.5,1) (5,0);
        TERM average := (5,0) (7.5,1) (10,0);
       TERM serious := (10,0) (13,1) (15,0);
       // Use 'Center Of Gravity' defuzzification method
       METHOD : COG;
       // Default value is 0 (if no rule activates defuzzifier)
        DEFAULT := 0;

END_DEFUZZIFY
```

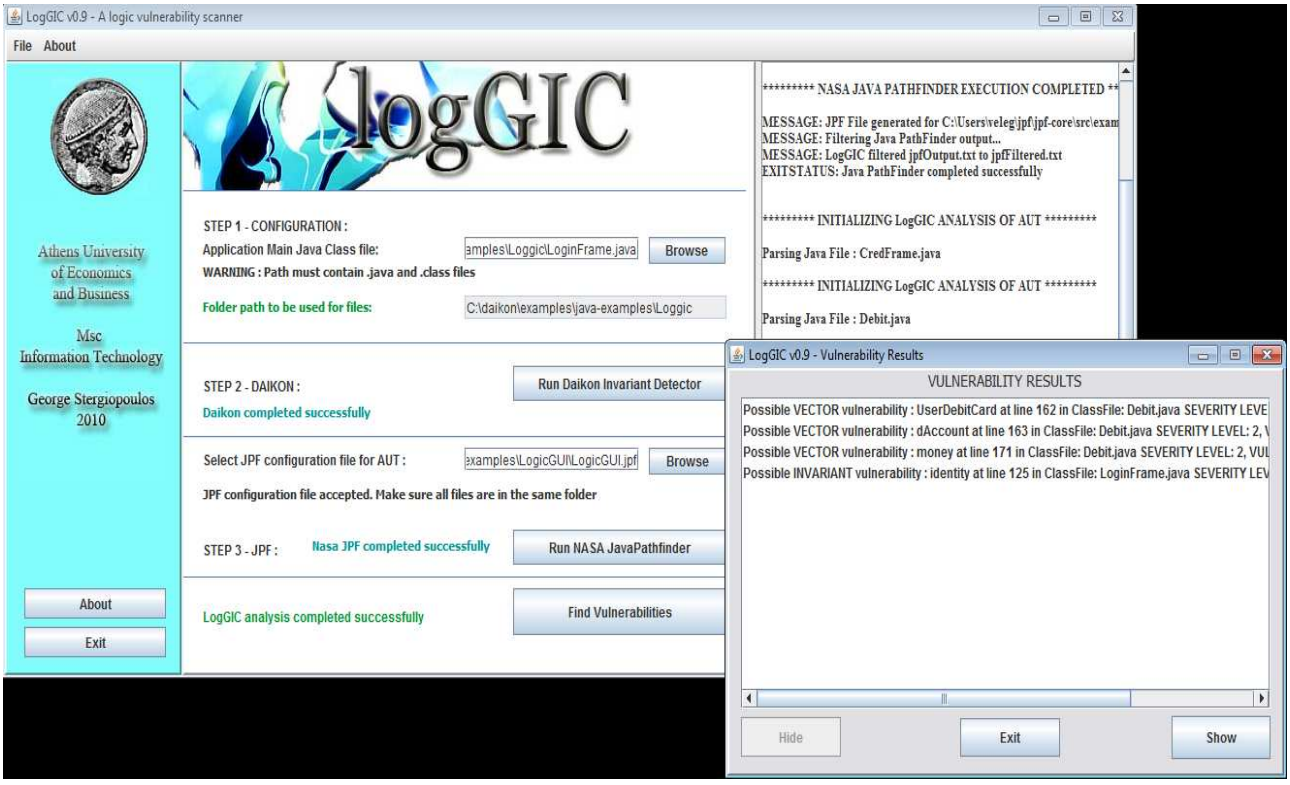
Παράδειγμα Αποσαφήνισης μεταβλητής εξόδου : {'trifle', 'average', 'serious'}

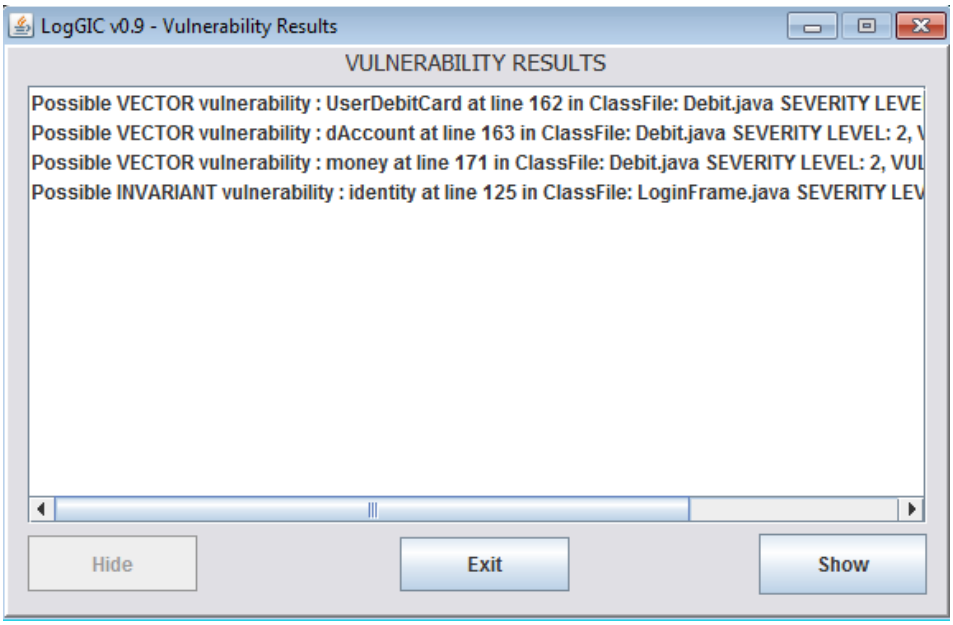
Περαιτέρω ανάλυση της λειτουργίας και του τρόπου υλοποίησης της βιβλιοθήκης JFuzzyLogic μπορεί κανείς να βρει στη διεύθυνση: http://jfuzzylogic.sourceforge.net/html/index.html.

7.9 ΠΑΡΟΥΣΙΑΣΗ ΤΩΝ ΑΠΟΤΕΛΕΣΜΑΤΩΝ ΛΕΙΤΟΥΡΓΙΑΣ ΤΟΥ ΕΡΓΑΛΕΙΟΥ ΚΑΙ ΤΩΝ ΠΙΘΑΝΩΝ ΑΔΥΝΑΜΙΩΝ

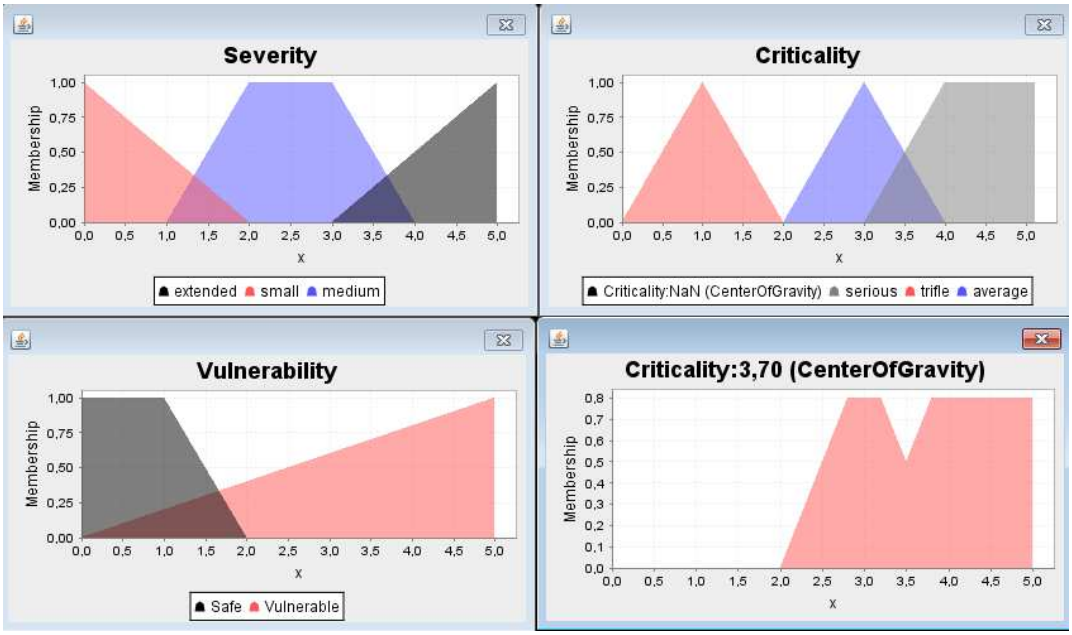
Για κάθε αποτέλεσμα που πιθανώς αντιστοιχεί σε μια λογική αδυναμία, το εργαλείο LogGIC δημιουργεί μια εγγραφή την οποία καταχωρεί σε μια λίστα αποτελεσμάτων. Αυτή η λίστα περιέχει όλες τις πιθανές αδυναμίες που έχει εντοπίσει το LogGIC, την μεταβλητή από την οποία εξαρτώνται, τον αριθμό της γραμμής του κώδικα που εμφανίζεται καθώς και τις πληροφορίες κρισιμότητάς της, όπως αυτές εξήχθησαν από το σύστημα ασαφούς λογικής. Αυτές οι πληροφορίες καθιστούν τον προσδιορισμό λογικών αδυναμιών στον πηγαίο κώδικα, μια εύκολη υπόθεση. Οι λογικές αδυναμίες κατατάσσονται στη λίστα με βάση το είδος τους. Αν δηλαδή προέρχονται από μολυσμένα δεδομένα διανυσμάτων εισόδου, έχουν την ένδειξη “VECTOR” στην αρχή της εγγραφής τους, αλλιώς έχουν την ένδειξη “INVARIANT” για αδυναμίες παραβίασης δυναμικών κανόνων.

Επειδή όλα τα ευρήματα του LogGIC δεν αποτελούν το ίδιο σοβαρές αδυναμίες, κάθε εγγραφή στην λίστα μπορεί να επιλεχθεί και να αναλυθεί περαιτέρω. Διαλέγοντας μια εγγραφή της λίστας και πατώντας το κουμπί “Show”, το LogGIC εμφανίζει μια διαγραμματική απεικόνιση των αποτελεσμάτων ανάλυσης της εκάστοτε αδυναμίας από το σύστημα ασαφούς λογικής. Τα πεδία “Severity”, “Vulnerability” και “Criticality” αποτελούν την γραφική απεικόνιση των κανόνων που διέπουν το σύστημα ασαφούς λογικής του LogGIC, ενώ το πεδίο «Center of Gravity» αποτελεί την τελική τιμή αξιολόγησης της αδυναμίας, από το πεδίο τιμών 1-5 της κλίμακας Likert.





Μια ενδεικτική παρουσίαση των αποτελεσμάτων Ασαφούς Λογικής για μια λογική αδυναμία που εντοπίστηκε, με την μορφή διαγραμμάτων:



8. ΠΕΡΙΟΡΙΣΜΟΙ ΛΕΙΤΟΥΡΓΙΑΣ ΚΑΙ ΜΕΛΛΟΝΤΙΚΗ ΕΡΕΥΝΑ

Ο απώτερος στόχος της μεθόδου που αναπτύχθηκε στη παρούσα διπλωματική, είναι ο αυτόματος εντοπισμός λογικών αδυναμιών σε ένα ευρύ φάσμα εφαρμογών με γραφική διεπαφή της γλώσσας προγραμματισμού Java. Παρόλα αυτά, η συγκεκριμένη υλοποίηση της μεθόδου στο εργαλείο LogGIC παρουσιάζει κάποιους περιορισμούς ως προς τα αποτελέσματα που είναι ικανή να παράξει, πολλοί από τους οποίους θα μπορούσαν να αρθούν με συστηματική δουλειά σε διδακτορικό επίπεδο, επίπεδο που είναι εκτός των ορίων της παρούσας διπλωματικής.

Κατ' αρχάς, τα είδη των αδυναμιών που μπορούν να εντοπιστούν από το LogGIC είναι περιορισμένα λόγω των αλγορίθμων που υλοποιούνται στο εργαλείο για τον εντοπισμό τους. Παραδείγματος χάριν, αν κάποιος προγραμματιστής χρησιμοποιήσει κάποιον αυτοσχέδιο μηχανισμό κώδικα για να εισάγει δεδομένα στην εφαρμογή του από την διεπαφή χρήστη, αντί να χρησιμοποιήσει παρεχόμενες εντολές της Java για ανάγνωση πεδίων διεπαφής όπως η "getText()", τότε το εργαλείο LogGIC δεν θα μπορούσε να αναγνωρίσει την ύπαρξη πιθανής αδυναμίας σε ένα τέτοιο σημείο κώδικα.

Επιπλέον, οι αδυναμίες που προκύπτουν από τα δεδομένα εισόδου πρέπει να κρύβονται στα σημεία διακλάδωσης και ελέγχου περιορισμών, δηλαδή σε if αποφάσεις εκτέλεσης. Αδυναμίες που προέρχονται από πολύπλοκες διακλαδώσεις switch ή από πολύπλοκες διαδικασίες, όπως λογική ένωση συγκρίσεων τύπου REGEX, δεν αναγνωρίζονται επιτυχώς από το LogGIC. Επίσης υπάρχει έλλειψη περαιτέρω ελέγχων πέραν του προσδιορισμού του τύπου ελέγχου που επιβάλλεται στα δεδομένα των διανυσμάτων εισόδου. Δεν υπάρχει αποτίμηση των πιθανών ελέγχων σημασιολογίας του περιεχομένου των δεδομένων στις μεταβλητές. Μια τέτοια μελέτη σημασιολογίας χρειάζεται πρόσβαση σε περαιτέρω πληροφορίες που να αντικατοπτρίζουν την επιχειρησιακή λογική του προγράμματος. Μια πιθανή λύση θα ήταν η ενσωμάτωση ανάγνωσης των προϋποθέσεων μια εταιρείας, από ένα δομημένο XBLR αρχείο.

Ένας άλλος περιορισμός είναι οι περιορισμοί που κληρονομούνται από τα εργαλεία που χρησιμοποιεί το LogGIC προκειμένου να εξάγει τα δικά του συμπεράσματα. Στην παρούσα διπλωματική -και ακολουθώντας τα βήματα της ομάδας ασφαλείας του G. Vigna στην δικιά τους υλοποίηση [2] - γίνεται χρήση του Daikon για την εκτέλεση της δυναμικής ανάλυσης. Παρόλο που το Daikon είναι ένα αξιόλογο εργαλείο δυναμικής ανάλυσης, έχει περιορισμένη υποστήριξη για κάποιες πολύπλοκες δομές δεδομένων [2], ενώ επίσης δεν υποστηρίζει και παραγωγή λογικών κανόνων για μεταβλητές που αξιοποιούνται μέσα σε βρόχους τύπου "for" και "while". Ως εκ τούτου, η έλλειψη υποστήριξης πολύπλοκων δομών και βρόχων

επανάληψης περιορίζει το εργαλείο LogGIC, το οποίο δεν εντοπίζει λογικά λάθη σε αντίστοιχα σημεία κώδικα. Για παράδειγμα, αν μια μεταβλητή συμβολοσειράς ‘username’ δεν δέχεται κάποια τιμή μέσα σε κάποια μέθοδο της εφαρμογής υπό έλεγχο αλλά ορίζεται μέσω μια εξωτερικής κλάσης στο εσωτερικό ενός βρόχου ‘while’, τότε το Daikon δεν θα μπορέσει να εξάγει τους λογικούς κανόνες για αυτή την μεταβλητή.

Ένας περιορισμός με την ευρύτερη έννοια ήταν η ανάγκη χρήσης της εφαρμογής υπό έλεγχο από κάποιον έμπειρο αναλυτή, προκειμένου να εξαχθούν οι απαραίτητες πληροφορίες από την δυναμική και την στατική ανάλυση. Η εκτέλεση του προγράμματος πρέπει να είναι όσο το δυνατόν πιο πλήρης κατά την δυναμική ανάλυση. Ο αναλυτής πρέπει να εξαντλήσει όσο το δυνατόν περισσότερες λειτουργίες της εφαρμογής υπό έλεγχο, καθώς και να εκτελέσει τις πιθανές υποκατηγορίες κάθε επιλογής που παίρνει στην εφαρμογή. Κατ’ επέκταση, το αρχείο παραμέτρων που θα ορίσει ο χρήστης για την στατική ανάλυση του JPF καθώς και το αρχείο προδιαγεγραμμένων βημάτων .ES που θα καθορίσει ο χρήστης για την στατική ανάλυση πρέπει να καλύπτουν τις ανωτέρω προϋποθέσεις που χρειάζεται και η δυναμική ανάλυση. Τα βήματα που περιγράφονται στο αρχείο .ES πρέπει να καλύπτουν όσο το δυνατόν περισσότερες λειτουργίες της εφαρμογής υπό έλεγχο, ευθύνες που βαραίνουν τον αναλυτή παρά την μέθοδο καθαυτή για την ορθή εξαγωγή συμπερασμάτων. Δυστυχώς, στη παρούσα φάση τα δύο αυτά βήματα ανάλυσης δεν μπορούν να αυτοματοποιηθούν πλήρως, αδυναμία η οποία προέρχεται τόσο από τον τρόπο λειτουργίας τον εργαλείων που χρησιμοποιεί το LogGIC, όσο και από την ίδια την ιδιομορφία της δυναμικής και στατικής ανάλυσης ως διαδικασίες.

Αναφερόμενοι στο JPF, ένας βασικός περιορισμός της μεθόδου εύρεσης λογικών αδυναμιών που βασίζεται στην στατική ανάλυση είναι οι περιορισμένες δυνατότητες της επέκτασης του JPF που χρησιμοποιείται εκτενώς στην ανάλυση εφαρμογών με γραφική διεπαφή, με το όνομα jpf-awt. Η συγκεκριμένη επέκταση υλοποιεί δικές τις εκδόσεις των γραφικών αντικειμένων και κλάσεων της Java, οπότε η υλοποίησή της είναι περιορισμένη όσον αφορά περίπλοκες γραφικές διεπαφές με σύγχρονα αντικείμενα απεικόνισης. Για παράδειγμα, είναι αναγκαία συνθήκη για την λειτουργία του jpf-awt και των προδιαγεγραμμένων βημάτων που ορίζει ο αναλυτής στο αρχείο .ES, όλα τα πεδία της γραφικής απεικόνισης (κουμπιά, λίστες, πεδία τιμών, διανύσματα εισόδου κτλ) να ορίζουν το όνομά τους μέσω της μεθόδου « Component.setName(“ “) ». Ειδάλλως, το jpf-awt δεν θα μπορέσει να αναγνωρίσει τα στοιχεία της γραφικής διεπαφής. Παρόλο που συνήθως αυτή η εντολή δηλώνεται αυτόματα από τους προγραμματιστές και από τα εργαλεία αυτόματης παραγωγής γραφικών διεπαφών, εντούτοις είναι κάτι που ο αναλυτής πρέπει να έχει πάντα κατά νου. Ένα άλλο παράδειγμα περιορισμένης ικανότητας του jpf-awt είναι το γεγονός ότι

δεν μπορεί να εκτελεστεί σε εφαρμογές που θέτουν τα αντικείμενα τύπου JLabel ως κληρονομικά αντικείμενα (inherit parent GUI object), δηλαδή όταν μπορούν να δεχτούν γραφική περιγραφή από άλλα αντικείμενα της εφαρμογής.

Ένας περιορισμός στο εύρος υλοποίησης της μεθόδου που μειώνει τον αριθμό των λογικών αδυναμιών που μπορούν να ανιχνευτούν, είναι η περιορισμένη κάλυψη πιθανής παράλληλης διασποράς μολυσμένων δεδομένων κατά την ανάλυση διανυσμάτων εισόδου και εντολών διακλάδωσης. Παραδείγματος χάριν, αν μια μεταβλητή δεχτεί δεδομένα από ένα διάνυσμα εισόδου και αμέσως μετά, μέσα στην ίδια μέθοδο, διαχύσει τα δεδομένα σε πολλαπλές μεταβλητές που η κάθε μια αποτελέσει παράμετρο σε διαφορετική μέθοδο, τότε το εργαλείο LogGIC αδυνατεί να παράξει μια γενική εκτίμηση για την ύπαρξη λογικού λάθους στο συγκεκριμένο διάνυσμα εισόδου. Ελέγχει διεξοδικά την πρώτη και κύρια διασπορά που θα εντοπίσει και θέτει μέτριες προς χαμηλές τιμές Δριμύτητας (severity) αν η πρώτη ανάλυση απεδείχθη ασφαλής.

Ένας ακόμη περιορισμός που περιορίζει το εύρος των πιθανών λογικών αδυναμιών που μπορούν να εντοπισθούν, είναι η έλλειψη συμβολικής εκτέλεσης των λειτουργιών της εφαρμογής υπό έλεγχο με ενδεικτικές τιμές εισόδου και επιθυμητές τιμές εξόδου. Οι εφαρμογές με γραφική διεπαφή, σε αντίθεση με την λειτουργία των Java servlets, δεν έχουν μια πεπερασμένη κατάσταση εκτέλεσης της λειτουργίας τους, καθώς η εκτέλεση διαδικασιών ορίζεται από την χρήση ακροατών και κουμπιών στην γραφική διεπαφή. Αυτή η ιδιοτροπία δημιουργεί μη-πεπερασμένα στον αριθμό μονοπάτια εκτέλεσης, ειδικά για μεγάλες εφαρμογές με πολύπλοκες γραφικές διεπαφές. Μια λύση θα ήταν η εκτέλεση της λειτουργίας της εφαρμογής μέσω στατικής ανάλυσης, κάνοντας ταυτόχρονα καταγραφή, βήμα-βήμα, του χώρου διευθύνσεων. Η καταγραφή τιμών, σειράς εκτέλεσης, μεταβλητών συστήματος και χώρων μνήμης θα μπορούσε να οδηγήσει στην άριστη αξιολόγηση των λογικών κανόνων που παράγει η δυναμική εκτέλεση. Μια τέτοια όμως υλοποίηση θα απαιτούσε επανεγγραφή τμημάτων του εργαλείου στατικής ανάλυσης JPF, καθώς και τροποποίηση της μεθόδου εξαγωγής πληροφοριών από την Ιδεατή Μηχανή JVM της γλώσσας Java, απαιτήσεις οι οποίες είναι εκτός των στόχων της παρούσας διπλωματικής.

Τέλος, ένας βασικός περιορισμός είναι ότι το εργαλείο Daikon πρέπει να εγκατασταθεί στη διαδρομή “ C:\ daikon “, ενώ το εργαλείο Java PathFinder στη διαδρομή της μεταβλητής συστήματος του χρήστη User_Home, δηλαδή στο $<user.home> \ jpf για τον πυρήνα του JPF και \ .jpf για το αρχείο εγκατάστασης του προγράμματος. Επίσης, το LogGIC δημιουργεί και εκτελεί προσωπικά εκτελέσιμα αρχεία για την χρήση των εργαλείων σε περιβάλλον Windows, καθώς τα εργαλεία δεν υποστηρίζουν την χρήση της καθέτου ‘\’ ως τρόπο ορισμού φακέλων και

υποφακέλων, γεγονός που αποτρέπει την χρήση των εργαλείων με καθαρά εμφωλευμένο τρόπο.

9. ΧΡΗΣΗ ΤΗΣ ΜΕΘΟΔΟΥ ΜΕ ΤΟ ΕΡΓΑΛΕΙΟ LOGGIC ΚΑΙ ΕΞΑΓΩΓΗ ΣΥΜΠΕΡΑΣΜΑΤΩΝ

Η χρήση του εργαλείου LogGIC είναι σχετικά απλή. Τα μοναδικά σημεία τα οποία ίσως φανούν δύσχρηστα στους πιθανούς χρήστες, είναι τα σημεία παραμετροποίησης των εργαλείων που χρησιμοποιεί το LogGIC, παρά η εγκατάσταση και η εκτέλεση του LogGIC αυτού καθεαυτού. Στα πλαίσια δοκιμής της αποδοτικότητας του εργαλείου LogGIC, αναπτύχθηκε μια ενδεικτική εφαρμογή γραφικής διεπαφής με το όνομα LogicBomb, η οποία περιέχει διάφορα λογικά λάθη διαφορετικών ειδών διάσπαρτα μέσα στις λειτουργίες της.

Εκ των έντεκα κριτηρίων που περιλαμβάνει η κλίμακα CCWAPSS -όπως παρουσιάστηκε σε προηγούμενο κεφάλαιο- το LogGIC βρίσκει υποσύνολα αδυναμιών που εμπίπτουν στις πρώτες τρεις κατηγορίες που φαίνονται στην παραπάνω εικόνα. Οι τρεις αυτές κατηγορίες καλύπτουν ένα ευρύτερο φάσμα εφαρμογών, το οποίο επεκτείνεται σε αδυναμίες που εμφανίζονται σε ευρύτερο σύνολο εφαρμογών, πέραν των διαδικτυακών που αξιολογεί η κλίμακα CCWAPSS. Συγκεκριμένα, τα κριτήρια της κλίμακας που περιγράφουν ένα μέρος των αδυναμιών που εντοπίζει το LogGIC, είναι τα ακόλουθα:

- Θέματα Αυθεντικοποίησης (Authentication)
- Θέματα Εξουσιοδότησης (Authorization)
- Θέματα ελέγχου και αποστείρωσης δεδομένων εισόδου του χρήστη (User input sanitization)

9.1 ΑΝΑΠΤΥΞΗ ΕΦΑΡΜΟΓΗΣ LOGICBOMB ΓΙΑ ΕΛΕΓΧΟ ΜΕ ΤΟ LOGGIC

Η εφαρμογή που χρησιμοποιείται ως κριτήριο αξιολόγησης του LogGIC περιέχει δύο βασικά και σχετικά περίπλοκα λογικά λάθη και ένα πιο απλό. Προκειμένου τα αποτελέσματα της διαδικασίας του LogGIC να έχουν αξιόπιστη βάση, υλοποιήθηκαν διάφορα είδη ελέγχου και τεχνικών ελέγχου δεδομένων μέσα στο LogicBomb. Εξάλλου, η αναζήτηση λογικών αδυναμιών σε μια εφαρμογή που δεν εφαρμόζει περιορισμούς σε κανένα της σημείο δεν έχει νόημα.

Έχοντας αυτό υπόψη, το LogicBomb περιέχει το πρώτο του λογικό λάθος στην διαδικασία αυθεντικοποίησης του χρήστη. Παρόλο που η διαδικασία αυθεντικοποίησης λειτουργεί ορθά, εντούτοις υπάρχει ένας τρόπος παράκαμψης της λειτουργίας. Αν κάποιος χρήστης χρησιμοποιήσει την διαδικασία Αποστολής αναγνωριστικών (Recover Credentials) προκειμένου να αποκτήσει το όνομα χρήστη και το συνθηματικό του με ορθό τρόπο, τότε μετά την επιτυχή ολοκλήρωση της διαδικασίας η εφαρμογή LogicBomb επιστρέφει τον χρήστη στη διεπαφή αυθεντικοποίησης, όπως θα έπρεπε. Παρόλα αυτά, από λογικό λάθος έχει περάσει ένα κατάλοιπο πληροφορίας από την προηγούμενη λειτουργία στη λειτουργία αυθεντικοποίησης. Όταν η διαδικασία αποστολής αναγνωριστικών ολοκληρώνεται, δεν αρχικοποιεί τις μεταβλητές ελέγχου της διαδικασίας αυθεντικοποίησης, αλλά, κατά λάθος, επιστρέφει το αναγνωριστικό του χρήστη που εκτέλεσε ορθά την προηγούμενη λειτουργία ως παράμετρο της διαδικασίας αυθεντικοποίησης. Αυτό έχει ως αποτέλεσμα πως, αν κάποιος εκτελέσει σωστά την αποστολή αναγνωριστικών (δηλαδή έχει το όνομα και το email ενός χρήστη), μετά να μπορεί να αυθεντικοποιήσει τον εαυτό του στην εφαρμογή δίχως να χρειάζεται να ξέρει επιπλέον το όνομα χρήστη και το συνθηματικό.

Η δεύτερη λογική αδυναμία υπάρχει στη λειτουργία αποστολής χρημάτων από μια κάρτα σε μια άλλη. Όταν ένας χρήστης αυθεντικοποιείται στην εφαρμογή LogicBomb, η κάρτα που του αντιστοιχεί συμπληρώνεται αυτόματα ως κάρτα πηγής κατά την λειτουργία αποστολής χρημάτων. Παρόλους τους υπάρχοντες ελέγχους όμως, ένας χρήστης είναι ικανός να αλλάξει την κάρτα πηγής και, αντί για την δικιά του, να δώσει μια άλλη κάρτα. Παρόλο που η δεύτερη κάρτα δεν αντιστοιχεί στον δικό του λογαριασμό, η συναλλαγή θα εκτελεστεί κανονικά. Η αδυναμία αυτή είναι κρυμμένη, καθώς το LogicBomb εκτελεί ελέγχους αρχικοποίησης στις κάρτες. Το πρόβλημα κρύβεται στο γεγονός ότι δεν υπάρχει έλεγχος ανάμεσα στην κάρτα που εισάγεται και στην κάρτα που αντιστοιχεί στο χρήστη, υπάρχει μόνο έλεγχος για το αν εισήχθη αριθμός με σωστή δομή (οι αριθμοί των καρτών τραπεζών αποτελούνται από τέσσερις τετράδες). Οι άλλη αδυναμία εμφανίζεται στον έλεγχο του ποσού χρημάτων που εισάγεται. Δεν ελέγχεται πουθενά στη λειτουργία του LogicBomb το απόθεμα

του λογαριασμού πηγής για το αν υπάρχει επαρκές ποσό για να αποσταλεί. Αυτή είναι μια απλή αδυναμία που δεν αντιστοιχεί σε ρεαλιστικά σενάρια και εισήχθη στο πρόγραμμα ως την βασική μορφή αδυναμίας που πρέπει να αναγνωρίζει οποιαδήποτε μέθοδος. Έχει το ρόλο της βάσης για την αξιολόγηση της υλοποίησης μας.

9.2 ΑΠΟΤΕΛΕΣΜΑΤΑ ΑΝΑΛΥΣΗΣ ΕΦΑΡΜΟΓΩΝ

Το LogicBomb κατάφερε να εντοπίσει επαρκώς όλα τα λογικά λάθη που ενσωματώθηκαν στην εφαρμογή LogicBomb. Η κατάταξη που πραγματοποίηση το σύστημα ασαφούς λογικής ήταν επαρκές αν και λίγο επιεικές σε σύγκριση με την σημαντικότητα των λογικών αδυναμιών που εντοπίσθηκαν. Παρόλα αυτά, είναι προτιμότερη η μετριοπαθής αξιολόγηση των αποτελεσμάτων παρά οι υπερβολικές δηλώσεις που θα έκριναν όλες τις μεταβλητές, ακόμα κι αυτές που δεν αποτελούν ιδιαίτερο πρόβλημα, ως επικίνδυνες για την εφαρμογή.

Προς μεγάλη μας ικανοποίηση, η μέθοδος είχε μηδενικά ψεύτικα αποτελέσματα ως αδυναμίες (το ποσοστό των false positives ήταν μηδέν). Παρόλα αυτά, η ελλιπής υλοποίηση της μεθόδου εύρεσης αδυναμιών μέσω της δυναμικής ανάλυσης λόγω των περιορισμών που αναφέρθηκαν νωρίτερα, οδήγησε στον ελλιπή εντοπισμό της αδυναμίας στην διαδικασία αυθεντικοποίησης. Παρόλο που η αδυναμία εντοπίστηκε, δεν υπήρξε σαφής αναγνώριση των λογικών κανόνων που παραβιάστηκαν κατά την εμφάνισή της. Αντίθετα, η αδυναμία εντοπίστηκε μέσω της σύγκρισης των κανόνων με τις αναθέσεις μεταβλητών που εξήγαγε η ευριστική ανάλυση του κώδικα, γεγονός που αποτελεί υποσύνολο των πραγματικών ικανοτήτων της μεθόδου που πατάει πάνω στα ευρήματα της έρευνας του Vigna και η οποία μπορούσε, λόγω εύρους δουλειάς, να εκτιμήσει ορθότερα τους λογικούς κανόνες.

Συνοψίζοντας, στα πλαίσια της παρούσας διπλωματικής έγιναν οι ακόλουθες συνεισφορές στην ήδη υπάρχουσα ακαδημαϊκή γνώση:

- Αναπτύχθηκε μια μέθοδος ανάλυσης πηγαίου κώδικα η οποία εντοπίζει και αναλύει τα διανύσματα εισόδου (με χρήση της θεωρητικής μεθόδου Ανάλυσης Μολυσμένων Αντικειμένων και της θεωρίας διανυσμάτων εισόδου). Η μέθοδος εντοπίζει, αναγνωρίζει και απομονώνει ένα σύνολο σημείων σε πηγαίο κώδικα, τα οποία αποτελούν πιθανή πηγή ύπαρξης λογικών αδυναμιών.
- Αναπτύχθηκε μια ευριστική μέθοδος που μελετά και απομονώνει την λογική των διακλαδώσεων και των αποφάσεων ροής εκτέλεσης στον πηγαίο κώδικα των εφαρμογών υπό έλεγχο.
- Υλοποιήθηκε μια παραλλαγή της μεθόδου εύρεσης λογικών αδυναμιών όπως παρουσιάζεται στη δημοσίευση της ομάδας Ασφαλείας της Καλιφόρνια [2], στην οποία γίνεται ταυτόχρονα δυναμική και στατική ανάλυση εφαρμογών με εκτέλεση προδιαγεγραμμένων σεναρίων σε μια εφαρμογή, προκειμένου να προσδιοριστούν οι λογικοί κανόνες και τα μονοπάτια εκτέλεσης της εφαρμογής υπό έλεγχο.
- Επεκτάθηκε η υπάρχουσα λογική αξιολόγησης των συμπερασμάτων από τις μεθόδους εύρεσης αδυναμιών. Ενσωματώθηκε ένα σύστημα κανόνων τύπου Fuzzy Logic προκειμένου να κατηγοριοποιούνται τα πιθανά λογικά λάθη και να τους προσάπτονται βάρη επικινδυνότητας χρησιμοποιώντας την μέθοδο ανάλυσης Κέντρου Βαρύτητας (Center of Gravity) της Ασαφούς Λογικής, ανάλογα με το αποτέλεσμα της έρευνας και την Δριμύτητα των ευρημάτων ως προς τις πιθανές επιπτώσεις τους στη λειτουργία της εφαρμογής.
- Στα πλαίσια ανάπτυξης της διπλωματικής, παραμετροποιήθηκε και βελτιώθηκε σε σημεία η λειτουργία της επέκτασης jpf-awt του εργαλείου Java NasaPathFinder, λύσεις προβλημάτων που ενσωματώθηκαν στην επίσημη διανομή από την ομάδα υποστήριξης του εργαλείου.
- Υλοποιήθηκε διαδικασία αλληλεπίδρασης με το δέντρο αφηρημένης σύνταξης του μεταγλωττιστή της Java, για τις ανάγκες της ευριστικής μεθόδου ανάλυσης κώδικα.
- Χρησιμοποιώντας και συνδυάζοντας τις υπάρχουσες μεθόδους, γίνεται μια πρωτότυπη προσπάθεια ανάλυσης εφαρμογών γραφικής διεπαφής (GUI) που κάνουν εκτενή χρήση ακροατών συστήματος και πολλαπλών δυνατών τρόπων εκτέλεσης.
- Η παραλλαγή της μεθόδου της ομάδας Ασφάλειας του πανεπιστημίου της Καλιφόρνια, η ευριστική μέθοδος ανάλυσης κώδικα καθώς και η μέθοδος εντοπισμού μολυσμένων αντικειμένων μαζί με τα μοντέλα Ασαφούς Λογικής

συνδυάστηκαν και ενσωματώθηκαν σε ένα καινούργιο ακαδημαϊκό εργαλείο για δοκιμές διείσδυσης και εύρεσης λογικών αδυναμιών σε εφαρμογές γραφικής διεπαφής με το όνομα **LogGIC**.

10. ΠΑΡΑΡΤΗΜΑΤΑ

10.1 ΠΑΡΑΡΤΗΜΑ Α - ΠΡΟΤΥΠΟ ΑΡΧΕΙΟ SITE.PROPERTIES ΓΙΑ ΤΗΝ ΕΓΚΑΤΑΣΤΑΣΗ ΤΟΥ JPF

Ακολουθεί το αρχείο "site.properties" που χρειάζεται το JPF προκειμένου να εντοπίσει τους φακέλους των αρχείων του. Στο αρχείο ορίζονται οι διαδρομές για τον βασικό πυρήνα του jpf: **jpf-core**, και για τις επεκτάσεις **jpf-numeric**, **jpf-aprop**, **jpf-trace-server** και **jpf-awt**. Οι παράμετροι του αρχείου που παρέχεται είναι δομημένοι έτσι ώστε να δουλεύουν σε όλα τα συστήματα, αρκεί να γίνει εγκατάσταση των αρχείων του JPF στη διαδρομή του αρχείου συστήματος που ορίζεται από την μεταβλητή συστήματος **$ user.home** (υπάρχει εξαρχής στα περισσότερα λειτουργικά συστήματα) :

```
# JPF site configuration
#
jpf-core = ${user.home}/jpf/jpf-core
#
# numeric extension
jpf-numeric = ${user.home}/jpf/jpf-numeric
extensions+=${jpf-numeric}
#
# annotation-based program properties extension
jpf-aprop = ${user.home}/jpf/jpf-aprop
extensions+=,${jpf-aprop}
#
# trace program to save executions
jpf-trace-server = ${user.home}/jpf/jpf-trace-
server
extensions+=,${jpf-trace-server}
#
# GUI input scripter
jpf-awt = ${user.home}/jpf/jpf-awt
extensions+=,${jpf-awt}
#
#... and all your other installed projects
```

10.2 ΠΑΡΑΡΤΗΜΑ Β – ΠΡΟΤΥΠΟ ΑΡΧΕΙΟ ΠΑΡΑΜΕΤΡΟΠΟΙΗΣΗΣ ΓΙΑ ΕΛΕΓΧΟ ΕΦΑΡΜΟΓΗΣ ΜΕ ΤΟ JPF

Ακολουθεί το αρχείο "LoginGUI.jpf" που πρέπει να ετοιμάσει ο χρήστης για την στατική ανάλυση της ενδεικτικής εφαρμογής LogicBomb που αναπτύχθηκε ως μέσο επίδειξης στα πλαίσια της διπλωματικής. Οι γραμμές που ουσιαστικά αλλάζουν από εφαρμογή σε εφαρμογή είναι οι υπογραμμισμένες:

```
target = LoginFrame

#--- which script to run
awt.script=${config_path}/LogicGUI.es

cg.enumerate_random = true

#--- what results to show
listener=.listener.ChoiceTracker
listener+=,.listener.ExecTracker

et.print_insn = false

choice.class=gov.nasa.jpf.awt.UIActionGenerator


report.console.property_violation=error,snapshot,trace



#--- example of how to run stuff that's not in
jpf.properties
classpath=${jpf-core}/src/examples/LogicGUI;${classpath}
sourcepath=${jpf-core}/src/examples/LogicGUI;${sourcepath}
```

10.3 ΠΑΡΑΡΤΗΜΑ Γ - ΠΑΡΑΜΕΤΡΟΠΟΙΗΣΗ ΛΕΙΤΟΥΡΓΙΑΣ ΤΟΥ DAIKON

Το εργαλείο Daikon είναι ανεπτυγμένο στη γλώσσα προγραμματισμού Java. Για την εκτέλεσή του, το μόνο που χρειάζεται κανείς είναι το εκτελέσιμο αρχείο "**daikon.jar**" το οποίο περιέχεται στην διανομή που παρέχεται στην ιστοσελίδα του εργαλείου: http://pag.csail.mit.edu/daikon/download/daikon.jar. Το αρχείο daikon.jar πρέπει να προστεθεί στην μεταβλητή συστήματος CLASSPATH προκειμένου να δουλέψει σωστά. Επίσης, είναι αναγκαία η ύπαρξη του πακέτου ανάπτυξης Java 5.0 (ή μεταγενέστερο) JVM (Java Virtual Machine). Το αρχείο tools.jar που υπάρχει μέσα στην διανομή πρέπει επίσης να βρίσκεται στην μεταβλητή συστήματος "classpath" [17]. Συνοπτικά, οι αναγκαίες ρυθμίσεις που πρέπει να γίνουν στο λειτουργικό σύστημα και που εξασφαλίζουν την ομαλή λειτουργία του εργαλείου Daikon σε περιβάλλον λειτουργίας Windows 7 -και XP- είναι οι ακόλουθες:

- Εγκατάσταση του πακέτου ανάπτυξης εφαρμογών Java SDK 1.6 και του πακέτου εκτέλεσης Java RE 6 από την Oracle: http://www.oracle.com/technetwork/java/javase/downloads/index.html
- Η διαδρομή στο σύστημα αρχείων πρέπει να είναι "**C:\daikon"** στα συστήματα με λειτουργικό σύστημα Windows, λόγω αδυναμίας του Daikon να χειριστεί διαδρομές και τοποθεσίες φακέλων που περιέχουν την κάθετο ' \ ' ή τον κενό χαρακτήρα ' ' στο μονοπάτι τους.
- Αντιγραφή του αρχείου QTJava.zip από τον φάκελο της Java και ανανέωση της μεταβλητής συστήματος CLASSPATH για την καινούργια του τοποθεσία.
- Προσθήκη στη μεταβλητή συστήματος CLASSPATH των αρχείων:
    - " daikon.jar " από το φάκελο εγκατάστασης C:\daikon.
    - dcomp_premain.jar (προαιρετικό για το LogGIC)
    - tools.jar από το φάκελο της εγκατάστασης του Java SDK 1.6.
- Όλα τα αρχεία πηγαίου κώδικα .java και τα αρχεία δυαδικού κώδικα .class πρέπει να βρίσκονται στον ίδιο φάκελο προκειμένου να λειτουργήσει η δυναμική ανάλυση.
- Ξεχωριστή εκτέλεση του Daikon για προσωπικές χρήσεις προϋποθέτει την μετάβαση μέσα στο φάκελο από την γραμμή εντολών, μέσω της εντολής 'CD' και έπειτα την εκτέλεση του προγράμματος με την εντολή :

Java daikon.Chicory --daikon Test

όπου 'Test' η κύρια κλάση της εφαρμογής υπό έλεγχο (π.χ. Test.class).

10.4 ΠΑΡΑΡΤΗΜΑ Δ - ΠΑΡΑΜΕΤΡΟΠΟΙΗΣΗ ΛΕΙΤΟΥΡΓΙΑΣ ΚΑΙ ΕΓΚΑΤΑΣΤΑΣΗ ΤΟΥ JAVA PATHFINDER

Ακολουθεί μια γενική παρουσίαση των βασικών χαρακτηριστικών και παραμέτρων προκειμένου να καταστεί δυνατή η ομαλή λειτουργία του εργαλείου JPF. Στα πλαίσια της διπλωματικής, το εργαλείο LogGIC παρέχεται μαζί με μια διαδικασία αυτόματης εγκατάστασης. Παρόλα αυτά, λόγω ύπαρξης πληθώρας παραμέτρων και διαφορετικών λειτουργικών συστημάτων, κρίνεται σκόπιμο να παρουσιαστεί η βασική δομή του JPF και οι κύριες παράμετροι της λειτουργίας του.

Η βασική δομή των φακέλων και των διαδρομών εγκατάστασης πρέπει να τηρηθεί αυστηρά, καθώς το JPF βασίζεται σε μια πληθώρα προκαθορισμένων παραμέτρων και διαδρομών του συστήματος αρχείων κάθε λογισμικού. Η παρούσα παρουσίαση εστιάζει στην εγκατάσταση του εργαλείου σε συστήματα υπολογιστών με λογισμικό Windows XP και νεώτερα. Το εργαλείο LogGIC καθώς και η χρήση του εργαλείου JPF έγιναν σε σύστημα με λογισμικό Windows 7. Η πλήρης τεκμηρίωση για την λειτουργία και την εγκατάσταση του προγράμματος NASA Java PathFinder, καθώς και πληθώρα οδηγών για υποστήριξη του προγράμματος μπορούν να βρεθούν στη διεύθυνση του ερευνητικού έργου του JPF: http://babelfish.arc.nasa.gov/trac/jpf/wiki.

Ο πυρήνας του JPF και οι περισσότερες από τις επεκτάσεις του είναι εφαρμογές ανεπτυγμένες στη γλώσσα Java, έκδοση 6, γεγονός που καθιστά το JPF ένα εργαλείο που μπορεί να δουλέψει πάνω από οποιοδήποτε λειτουργικό σύστημα, αρκεί να έχει υποστεί την κατάλληλη παραμετροποίηση. Ο κώδικας του εργαλείου είναι διαθέσιμος μέσα από μία διαδικτυακή αποθήκη που υποστηρίζει την δομή Mercurial. Για περισσότερες πληροφορίες και λήψη του προγράμματος αλληλεπίδρασης με αποθήκες Mercurial, μπορεί κανείς να επιστεφτεί την επίσημη ιστοσελίδα: http://mercurial.selenic.com/. Ο πυρήνας του JPF, το jpf-core αποτελεί αναγκαία προϋπόθεση προκειμένου να λειτουργήσει το εργαλείο. Όλες οι υπόλοιπες επεκτάσεις είναι προαιρετικές. Κάθε επέκταση είναι ένα ξεχωριστό ερευνητικό έργο που μπορεί να προσπελαστεί από τις πλατφόρμες ανάπτυξης προγραμμάτων Java, Netbeans ή Eclipse [18].

Η δομή του JPF ορίζεται σε ένα αρχείο με το όνομα site.properties, το οποίο ενημερώνει το JPF κατά την εκκίνησή του για το πού να κοιτάξει αυτό για τα αρχεία τις εγκατάστασης, ώστε να μπορέσει να θέσει τις σωστές μεταβλητές συστήματος αυτόματα. Ενδεικτική δομή των παραμέτρων μπορεί να δει κανείς στο Παράρτημα Α.

Η ενδεδειγμένη διαδρομή του συστήματος αρχείων για την εγκατάσταση του αρχείου παραμέτρων site.properties, είναι η διαδρομή **"<user.home> / .jpf / site.properties**", όπου user.home, η μεταβλητή συστήματος που ορίζει την διαδρομή

χρήστη. Αντίστοιχα, στην ίδια διαδρομή "**<user.home>** / **jpf** / " πρέπει να εγκατασταθεί ο πυρήνας του JPF, jpf-core. Για παράδειγμα σε ένα σύστημα με λειτουργικό Windows 7, η εγκατάσταση θα έπρεπε να γίνει στην διαδρομή φακέλου: " C:\Users\ONOMA_ΧΡΗΣΤΗ\jpf " για το jpf-core και όλες τις επεκτάσεις του και να δημιουργηθεί ένας φάκελος με όνομα **.jpf** στην ίδια διαδρομή που να περιέχει το αρχείο site.properties. Κάθε επέκταση, όπως και το jpf-core, καταλαμβάνει το δικό της φάκελο και δημιουργείται μέσω της χρήσης του Mercurial, όπως αναφέρθηκε παραπάνω [18].

Σε περίπτωση χρήσης του εργαλείου πάνω από λειτουργικό Windows 7, είναι απαραίτητο να σημειωθεί πως, όταν θέλει κανείς να κάνει χρήση απόλυτων μονοπατιών του συστήματος αρχείων, δεν πρέπει να χρησιμοποιεί τον χαρακτήρα '\' για τον διαχωρισμό των φακέλων όπως κάνει το λειτουργικό Windows. Αυτός ο χαρακτήρας μεταφράζεται διαφορετικά από τα προγράμματα Java και παραπέμπει στους ειδικούς χαρακτήρες, όπως για παράδειγμα η αλλαγή γραμμής που ορίζεται ως '**n**'. Κατά την παραμετροποίηση του JPF, πρέπει να χρησιμοποιούνται οι κάθετες '/'.

Η εγκατάσταση και δημιουργία των εκτελέσιμων αρχείων .jar για το εργαλείο JPF γίνεται μέσω των προδιαγεγραμμένων αρχείων εκτέλεσης εντολών build.xml που παρέχονται. Ο χρήστης πρέπει να έχει δικαιώματα διαχειριστή, προκειμένου να αποφευχθούν τυχόν μηνύματα λάθους. Η δημιουργία των εκτελέσιμων αρχείων γίνεται με την ακόλουθη εντολή ant μέσω της γραμμής εντολών, όπως φαίνεται και στην ακόλουθη εικόνα:

```
> cd jpf-core
> bin/ant test

... lots of output, at the end you should see something like:
BUILD SUCCESSFUL
Total time: 2 minutes 31 seconds
```

Κάθε εφαρμογή που θέλουμε να ελέγξει το JPF χρειάζεται ένα προσωπικό της αρχείο παραμετροποίησης. Αυτό το αρχείο το τοποθετούμε στον φάκελο με τα αρχεία πηγαίου κώδικα και τις κλάσεις του προγράμματος που θέλουμε να ελέγξουμε. Δημιουργούμε ένα αρχείο κειμένου με κατάληξη "**.jpf**" και χρησιμοποιούμε το όνομα της κύριας κλάσης του προγράμματος υπό έλεγχο ως όνομα για το αρχείο. Για παράδειγμα, ένα όνομα αρχείου θα μπορούσε να είναι "**MAIN.JPF**" αν η αρχική κλάση του προγράμματος υπό έλεγχο είναι main.java / main.class. Αυτό το αρχείο περιέχει τις διαδρομές των εκτελέσιμων κλάσεων για το JPF, καθώς και τις παραμέτρους για την ορθή λειτουργία του εργαλείου, ποιές επεκτάσεις χρησιμοποιεί, ποιούς ωτακουστές κτλ. Η εκτέλεση του JPF γίνεται με την παροχή του ανωτέρω αρχείου που δημιουργήθηκε από τον χρήστη για την εφαρμογή υπό έλεγχο, ως

παράμετρο στην εκτέλεση του JPF (Το παράδειγμα είναι για την εφαρμογή Racer.Java):

```
java -jar build/RunJPF.jar src/examples/Racer.jpf
```

Για περαιτέρω λεπτομέρειες όσων αφορά τη δημιουργία ενός τέτοιου αρχείου για κάθε εφαρμογή που θέλει κανείς να ελέγξει, επισκεφτείτε την επίσημη ιστοσελίδα: http://babelfish.arc.nasa.gov/trac/jpf/wiki/user/config. Παράδειγμα ενός αρχείου .jpf για την εφαρμογή LogicBomb -που ελέγχει το εργαλείο LogGIC στα πλαίσια της διπλωματικής- παρέχεται στο Παράρτημα Β.

11. ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] James A. Whittaker, May 2008, *Security Briefs: Penetration Testing*, MSDN, viewed 15 January 2011, http://msdn.microsoft.com/en-us/magazine/cc507646.aspx

[2] Felmetsger, Cavedon, Kruegel, Vigna, 'Toward Automated Detection of Logic Vulnerabilities in Web Applications', *USENIX Security Symposium Washington, DC, 12 August 2010*, Computer Security Group, Department of Computer Science, University of California, Santa Barbara

[3] Takanen, DeMott, Miller, 2008, *Fuzzing for Software Security Testing and Quality Assurance*, ARTECH HOUSE INC, Canton Street, Norwood

[4] Michael Sipser, 1997, *Introduction to the Theory of Computation*. PWS Publishing Co., Boston, ISBN 0-534-94728-X

[5] *Pairwise Testing*, DevelopSense, Version 1.5, November, 2007, http://www.developsense.com/pairwiseTesting.html, viewed 20 January 2011 at 17.00

[6] Dr. Daniel Geer and John Harthorne, 2002, 'Penetration Testing: A Duet', *Computer Security Applications Conference, 2002. Proceedings. 18th Annual,* STAKE

[7] Howard, A. Whittaker, 2005, 'Application Penetration Testing', *IEEE COMPUTER SOCIETY, IEEE SECURITY & PRIVACY*

[8] J. Tevis, A. Hamilton, 2006, Jr., 'Static Analysis of Anomalies and Security Vulnerabilities in Executable Files', *ACM SE'06, March 10-12, 2006, Melbourne, Florida, USA*

[9] Rothermel, Harroid**,** 1994, 'Performing data flow testing on classes', *SIGSOFT '94 Proceedings of the 2nd ACM SIGSOFT symposium on Foundations of software engineering, New York, NY, USA*, Department of Computer Science, Clemson University, Clemson, SC΄

[10] Livshits, S. Lam, September 2005, *'Finding Security Vulnerabilities in Java Applications with Static Analysis'*, Computer Science Department, Stanford University

[11] J. Zhou and G. Vigna, 'Detecting Attacks That Exploit Application-Logic Errors Through Application-Level Auditing', *Proceedings of the Annual Computer Security Applications Conference (ACSAC)* 168-178 Tucson, AZ December 2004, http://www.cs.ucsb.edu/~vigna/publications/2004_zhou_vigna_ACSAC04.pdf

[12] *M. Cova, V. Felmetsger, G. Banks, and G. Vigna,* December 2006*,* 'Static Detection of Vulnerabilities in x86 Executables'*, Proceedings of the Annual Computer Security Applications Conference (ACSAC)* Miami, Florida, http://www.cs.ucsb.edu/~vigna/publications/2006_cova_felmetsger_banks_vigna_ACSAC.pdf

[13] Gary McGraw, February 2006*, Building Security In: Software Penetration Testing,* Addison-Wesley Professional

[14] A. Doupe, M. Cova, and G. VignaProceedings , 'Why Johnny Can't Pentest: An Analysis of Black-box Web Vulnerability Scanners*', Detection of Intrusions and Malware and Vulnerability Assessment (DIMVA) Bonn, Germany July 2010, http://www.cs.ucsb.edu/~vigna/publications/2010_doupe_cova_vigna_dimva10.pdf*

[15] Halfond, Choudhary, Orso, 'Penetration Testing with Improved Input Vector Identification'*, ICST '09 Proceedings of the 2009 International Conference on Software Testing Verification and Validation - IEEE Computer Society Washington, DC, USA 2009,* College of Computing, Georgia Institute of Technology

[16] J. Zhou, G. Vigna, 2004*,* 'Detecting Attacks That Exploit Application-Logic Errors Through Application-Level Auditing'*, Proceedings of the Annual Computer Security Applications Conference (ACSAC) 168-178 Tucson, AZ December 2004,* Department of Computer Science, University of California, Santa Barbara

[17] Ernst, Perkins, Guo, McCamant, Pacheco, Tschantz, Xiao, *2007, The Daikon Invariant Detector User Manual,* Massachusetts Institute of Technology*,* *http://groups.csail.mit.edu/pag/daikon/download/doc/daikon.html* viewed on 20 January 2011

[18] *Java PathFinder*, Ames Research Center, NASA , *http://babelfish.arc.nasa.gov/trac/jpf/wiki,* viewed on 20 January 2011

[19] Halfond, Choudhary, Orso, 'Penetration Testing with Improved Input Vector Identification', *ICST '09 Proceedings of the 2009 International Conference on Software Testing Verification and Validation* - IEEE Computer Society Washington, DC, USA 2009, College of Computing, Georgia Institute of Technology

[20] Pablo Cingolani, *Open Source Fuzzy Logic library and FCL language implementation,* Pablo Cingolani, Sourceforge,net, http://jfuzzylogic.sourceforge.net/html/index.html viewed on 20 January 2011

[21] D'Anjou, Fairbrother, Kehn, 2004, *The Java developer's guide to Eclipse,* Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA 2003

[22] Newsome, Song, July 2005, 'Dynamic Taint Analysis for Automatic Detection, Analysis, and Signature Generation of Exploits on Commodity Software', *In Proceedings of the Network and Distributed System Security Symposium (NDSS 2005) 2005*, School of Computer Science, Carnegie Mellon University, Pittsburgh

[23] Dahl, Dijkstra, Hoare, 1972, *Structured Programming,* Academic Press

[24] *Decision Tables*, Saint Xavier University, Systems Analysis and Design, http://web.sxu.edu/rogers/sys/decision_tables.html viewed on 20 January 2011

[25] B. Buck, J. K. Hollingsworth, *An API for Runtime Code Patching*, The International Journal of High Performance Computing Applications, 14(4):317–329, Winter 2000.

[26] Jovanovic, Kruegel, Kirda, , 'Pixy: A Static Analysis Tool for Detecting Web Application Vulnerabilities (Short Paper)', *Security and Privacy, 2006 IEEE Symposium, Berkeley/Oakland, CA, 21-24 May 2006*, Secure Systems Lab, Technical University of Vienna

[27] Frédéric Charpentier, November 2007, *Common Criteria Web Application Security Scoring - CCWAPSS*, http://www.xmco.fr/whitepapers/ccwapss_1.1.pdf viewed 21 January 2011