Π Α Ν Ε Π Ι Σ Τ Η Μ Ι Ο Π Ε Ι Ρ Α Ι Ω Σ

ΤΜΗΜΑ ΨΗΦΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ
ΤΕΧΝΟΟΙΚΟΝΟΜΙΚΗ ΔΙΟΙΚΗΣΗ & ΑΣΦΑΛΕΙΑ ΨΗΦΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

Κατεύθυνση: ΑΣΦΑΛΕΙΑ ΨΗΦΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

Η ΧΡΗΣΗ ΤΗΣ ΣΤΑΤΙΚΗΣ ΑΝΑΛΥΣΗΣ ΣΤΗΝ ΠΑΡΑΓΩΓΗ ΑΣΦΑΛΟΥΣ ΛΟΓΙΣΜΙΚΟΥ

Δ Ι Π Λ Ω Μ Α Τ Ι Κ Η Ε Ρ Γ Α Σ Ι Α

ΓΕΩΡΓΙΟΣ Π. ΖΑΡΟΓΙΑΝΝΗΣ
ΜΤΕ 0906

ΕΠΙΒΛΕΠΩΝ**: ΑΝΑΠΛΗΡΩΤΗΣ ΚΑΘΗΓΗΤΗΣ ΚΩΝΣΤΑΝΤΙΝΟΣ ΛΑΜΠΡΙΝΟΥΔΑΚΗΣ**

ΑΘΗΝΑ 2013

ΠΕΡΙΛΗΨΗ

Τα περισσότερα περιστατικά ασφάλειας έχουν κάποια σχέση με το λογισμικό. Τα περιβόητα buffer overflows, SQL injections και άλλα αντίστοιχα προβλήματα που συχνά εκμεταλλεύονται οι κακόβουλοι χρήστες, είναι ευπάθειες του λογισμικού. Για την παραγωγή ασφαλούς λογισμικού, ένας οργανισμός χρειάζεται να υιοθετήσει μεθόδους και πρακτικές σε όλο τον κύκλο ζωής του λογισμικού. Μία από τις πρώτες μεθόδους που πρέπει να υιοθετηθούν και μια από τις πιο σημαντικές γενικά, είναι η στατική ανάλυση πηγαίου κώδικα. Σε αυτήν την εργασία παρουσιάζουμε συνοπτικά τις κρισιμότερες ευπάθειες του λογισμικού και περιγράφουμε τον τρόπο λειτουργίας των εργαλείων στατικής ανάλυσης. Κατόπιν χρησιμοποιούμε τρία ελεύθερα/ανοιχτού κώδικα τέτοια εργαλεία, για να αναλύσουμε καταρχήν μια δοκιμαστική εφαρμογή και έπειτα δύο πραγματικά έργα λογισμικού. Ένα ελεύθερο/ανοιχτού κώδικα και μία ιδιόκτητη επιχειρηματική εφαρμογή. Οι δοκιμαστικές αυτές εφαρμογές των εργαλείων στατικής ανάλυσης, έδειξαν ότι τα συγκεκριμένα εργαλεία έχουν περιορισμένες δυνατότητες και σε κάποιες περιπτώσεις παράγουν ένα μεγάλο πλήθος ψευδώς θετικών. Παρά τα όποια μειονεκτήματα, τα εργαλεία που δοκιμάστηκαν κατάφεραν να βρουν σοβαρά προβλήματα. Επίσης τα πολλά ψευδώς θετικά, κατέδειξαν τη χρήση κακών πρακτικών στον κώδικα. Σύμφωνα με αυτά τα ευρήματα και δεδομένου ότι κάποια εργαλεία εκτός από τα σφάλματα ασφάλειας, ψάχνουν και για λογικά, απόδοσης και άλλων ειδών σφάλματα, κρίνονται τελικά ως πολύ χρήσιμα σε κάθε προγραμματιστή.

ΠΙΝΑΚΑΣ ΠΕΡΙΟΧΟΜΕΝΩΝ

ΠΕΡΙΛΗΨΗ.....3

1 ΕΙΣΑΓΩΓΗ.....7

1.1 Ασφάλεια λογισμικού.....7
1.2 Σκοπός και δομή της εργασίας.....10

2 ΣΦΑΛΜΑΤΑ ΑΣΦΑΛΕΙΑΣ ΣΤΟ ΛΟΓΙΣΜΙΚΟ.....11

2.1 Ταξινομία των σφαλμάτων ασφάλειας.....11
2.2 CWE/SANS Top 25.....13
2.2.1 CWE-89. Ανεπαρκής απενεργοποίηση ειδικών στοιχείων σε μια εντολή SQL (SQL Injection).....15
2.2.2 CWE-78. Ανεπαρκής απενεργοποίηση ειδικών στοιχείων σε μια εντολή του λειτουργικού συστήματος (OS Injection).....16
2.2.3 CWE-120. Αντιγραφή προσωρινής αποθήκευσης, χωρίς έλεγχο του μεγέθους της εισόδου (Buffer Overflow).....18
2.2.4 CWE-79. Ανεπαρκής απενεργοποίηση εισόδου κατά την παραγωγή ιστοσελίδων (Cross-site Scripting).....20
2.2.5 CWE-306. Ελλιπής αυθεντικοποίηση σε κρίσιμη συνάρτηση.....22
2.2.6 CWE-862. Ελλιπής εξουσιοδότηση.....22
2.2.7 CWE-798. Ενσωματωμένα στον κώδικα διαπιστευτήρια..23
2.2.8 CWE-311. Ελλιπής κρυπτογράφηση ευαίσθητων δεδομένων 24
2.2.9 CWE-434. Ανέβασμα επικίνδυνων αρχείων χωρίς περιορισμούς.....24
2.2.10 CWE-807. Αποφάσεις ασφάλειας βασισμένες σε μη έμπιστη είσοδο.....24
2.2.11 CWE-250. Εκτέλεση με άχρηστα προνόμια.....24
2.2.12 CWE-352. Cross-Site Request Forgery (CSRF).....25
2.2.13 CWE-22. Ανεπαρκής έλεγχος πρόσβασης σε κατάλογο (Path Traversal).....25
2.2.14 CWE-494. Κατέβασμα κώδικα χωρίς έλεγχο ακεραιότητας.....26
2.2.15 CWE-863. Εσφαλμένη εξουσιοδότηση.....26
2.2.16 CWE-829. Προσθήκη λειτουργικότητας από μη έμπιστη πηγή.....27
2.2.17 CWE-732. Εσφαλμένη ανάθεση δικαιωμάτων σε κρίσιμο πόρο.....28
2.2.18 CWE-676. Χρήση εν δυνάμει επικίνδυνης συνάρτησης...28

2.2.19 CWE-327. Χρήση ανασφαλούς ή επικίνδυνου αλγορίθμου κρυπτογράφησης....29
2.2.20 CWE-131. Εσφαλμένος υπολογισμός μεγέθους προσωρινής μνήμης....29
2.2.21 CWE-307. Ελλιπής απαγόρευση απεριόριστων προσπαθειών αυθεντικοποίησης....29
2.2.22 CWE-601. Ανακατεύθυνση σε μη έμπιστο ιστότοπο (Open Redirect)....30
2.2.23 CWE- 134. Μη ελεγχόμενη συμβολοσειρά μορφοποίησης 31
2.2.24 CWE-190. Υπερχείλιση ακεραίου 31
2.2.25 CWE-759. Χρήση συνάρτησης κατακερματισμού χωρίς εισαγωγή τυχαίας συμβολοσειράς (salt)....31
2.3 Βέλτιστες πρακτικές για την αποφυγή σφαλμάτων ασφάλειας 32
2.3.1 Καθιέρωσε και συντήρησε ελέγχους σε όλες τις εισόδους (M1)....32
2.3.2 Καθιέρωσε και συντήρησε ελέγχους σε όλες τις εξόδους (M2)....33
2.3.3 Κλείδωσε το περιβάλλον (M3)....34
2.3.4 Υπόθεσε ότι τα εξωτερικά συστατικά μπορούν να τροποποιηθούν και ο κώδικάς σου μπορεί να διαβαστεί από οποιονδήποτε (M4)....36
2.3.5 Χρησιμοποίησε αποδεκτές από τη βιομηχανία λειτουργίες ασφάλειας και μην ανακαλύπτεις τις δικές σου (M5)....37
2.3.6 Χρησιμοποίησε βιβλιοθήκες και πλατφόρμες που διευκολύνουν την αποφυγή αδυναμιών (GP1)....38
2.3.7 Ενσωμάτωσε την ασφάλεια στον κύκλο ζωής του λογισμικού (GP2)....38
2.3.8 Χρησιμοποίησε ένα ευρύ φάσμα μεθόδων για να βρεις και να αποτρέψεις τις αδυναμίες (GP3)....39
2.3.9 Επίτρεψε στους κλειδωμένους πελάτες να αλληλεπιδράσουν με το λογισμικό σου (GP4)....41

3 ΣΤΑΤΙΚΗ ΑΝΑΛΥΣΗ....43

3.1 Εισαγωγή....43
3.1.1 Χρήσεις της στατικής ανάλυσης....43
3.1.2 Πλεονεκτήματα και μειονεκτήματα....46
3.1.3 Θεωρητικοί περιορισμοί....48
3.1.4 Ανάλυση πηγαίου/μεταγλωττισμένου κώδικα....49
3.2 Κατασκευή μοντέλου....50
3.2.1 Λεκτική ανάλυση....50
3.2.2 Συντακτική ανάλυση....51

3.2.3 Σημασιολογική ανάλυση.....53
3.2.4 Ιχνηλάτηση ροής ελέγχου.....55
3.2.5 Ιχνηλάτηση ροής δεδομένων.....57
3.2.6 Διάδοση μόλυνσης.....58
3.2.7 Ψευδώνυμα δεικτών.....59
3.3 Αλγόριθμοι ανάλυσης.....59
3.3.1 Έλεγχος ισχυρισμών.....60
3.3.2 Τοπική ανάλυση.....62
3.3.3 Καθολική ανάλυση.....63
3.4 Κανόνες.....64

4 ΠΕΡΙΠΤΩΣΕΙΣ ΧΡΗΣΗΣ.....67
4.1 Παρουσίαση των εργαλείων στατικής ανάλυσης.....67
4.1.1 Findbugs.....67
4.1.2 Lightweight Analysis for Program Security in Eclipse (LAPSE).....69
4.1.3 Yet Another Source Code Analyzer (YASCA).....70
4.2 Αποτελέσματα εφαρμογής των εργαλείων στατικής ανάλυσης 72
4.2.1 Securibench micro.....72
4.2.2 Hipergate.....76
4.2.3 Πραγματική εμπορική εφαρμογή.....80
4.3 Αξιολόγηση αποτελεσμάτων και γενικά συμπεράσματα.....82

ΒΙΒΛΙΟΓΡΑΦΙΚΕΣ ΑΝΑΦΟΡΕΣ.....85

ΚΕΦΑΛΑΙΟ 1

1 ΕΙΣΑΓΩΓΗ

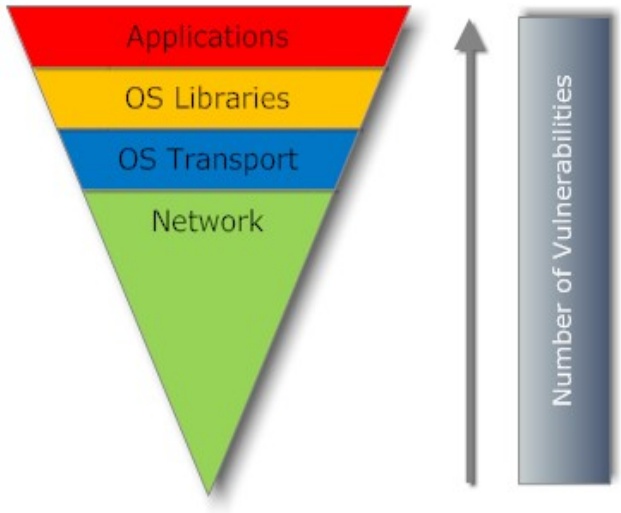
1.1 Ασφάλεια λογισμικού

Παρόλο που αλληλεπιδρούμε με το λογισμικό σε καθημερινή βάση, το μεταφέρουμε στα κινητά τηλέφωνά μας, οδηγούμε με αυτό στα αυτοκίνητά μας, πετάμε με αυτό στα αεροπλάνα μας και το χρησιμοποιούμε στους προσωπικούς και επαγγελματικούς υπολογιστές μας, το ίδιο το λογισμικό παραμένει ουσιαστικά ένα φάντασμα στο μηχάνημα. Ένα μυστήριο που λειτουργεί, μα όχι όλες τις φορές. Και εκεί βρίσκεται το πρόβλημά μας. Το λογισμικό είναι η ουσία των σύγχρονων υποδομών. Δεν είναι μόνο ότι το λογισμικό εμπεριέχεται σε έναν αυξανόμενο αριθμό εμπορικών προϊόντων που αγοράζουμε και χρησιμοποιούμε, αλλά οι κυβερνήσεις το χρησιμοποιούν ολοένα και περισσότερο για να διαχειριστούν λεπτομέρειες της ζωής μας, να διαθέσουν τα οφέλη και τις δημόσιες υπηρεσίες που απολαμβάνουμε ως πολίτες και να διαχειριστεί και να υπερασπιστεί το κράτος στο σύνολό του. Πώς και πότε θα αλληλεπιδράσουμε με το λογισμικό, είναι όλο και λιγότερο επιλογή μας κάθε μέρα. Η ποιότητα αυτού του λογισμικού έχει σημασία σε μεγάλο βαθμό. Το επίπεδο προστασίας που μας παρέχει αυτό το λογισμικό, από τη ζημία και την εκμετάλλευση, έχει σημασία ακόμη περισσότερο. [1]

Από τη σκοπιά της ασφάλειας πληροφοριακών συστημάτων, το λογισμικό έχει γίνει πιο σημαντικό από τα παραδοσιακά στοιχεία του δικτύου. Ο Bruce Schneier έχει δηλώσει: «*Σκεφτείτε την πιο πρόσφατη ευπάθεια ασφάλειας που έχετε διαβάσει σχετικά. Ίσως είναι ένα πακέτο δολοφόνος, το οποίο επιτρέπει σε έναν επιτιθέμενο να προκαλέσει κατάρρευση σε έναν διακομιστή, στέλνοντας του ένα συγκεκριμένο πακέτο. Ίσως είναι μία από τις άπειρες υπερχειλίσεις στοίβας (buffer overflows), η οποία επιτρέπει*

σε έναν εισβολέα να πάρει τον έλεγχο ενός υπολογιστή, με την αποστολή ενός συγκεκριμένου δύσμορφου μηνύματος. Ίσως είναι μια ευπάθεια στην κρυπτογράφηση, η οποία επιτρέπει σε έναν εισβολέα να διαβάσει ένα κρυπτογραφημένο μήνυμα, ή να ξεγελάσει ένα σύστημα ελέγχου ταυτότητας. Όλα αυτά είναι θέματα του λογισμικού». [2]

Σύμφωνα με το SANS [3], κατά τη διάρκεια των τελευταίων ετών, ο αριθμός των ευπαθειών που ανακαλύφθηκε στις εφαρμογές λογισμικού, είναι πολύ μεγαλύτερος από τον αριθμό των ευπαθειών που ανακαλύφθηκαν σε λειτουργικά συστήματα. Ως αποτέλεσμα, οι περισσότερες προσπάθειες εκμετάλλευσης καταγράφονται στις εφαρμογές λογισμικού:

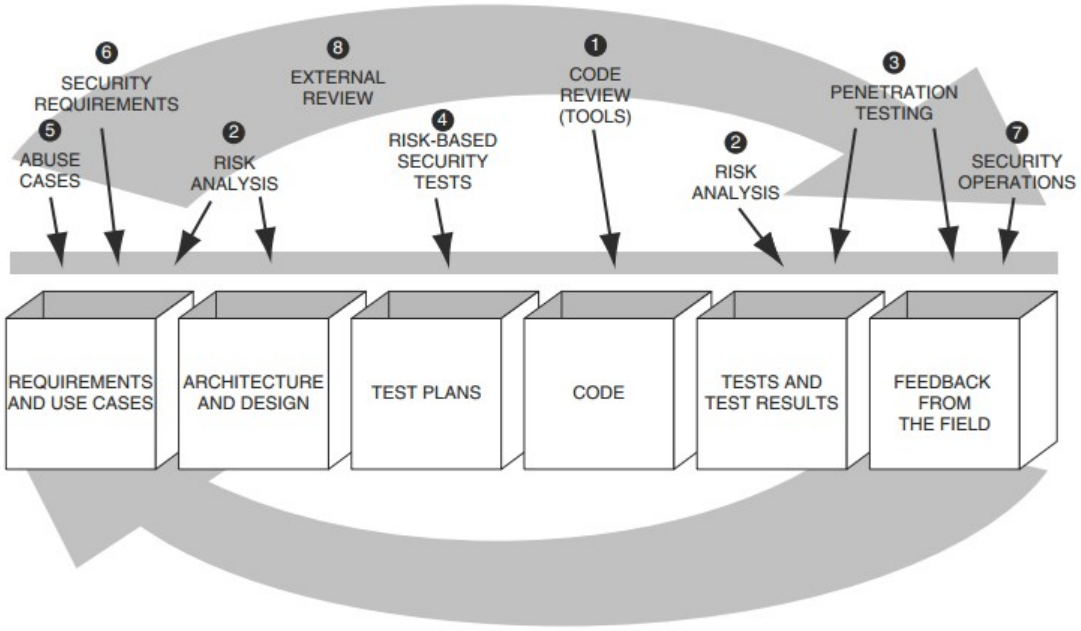


Εικόνα 1: Πλήθος ευπαθειών στα επίπεδα του συστήματος

Τα εργαλεία ασφάλειας δικτύων (firewall, IPS κλπ), δεν μπορούν να προστατεύσουν τις εφαρμογές λογισμικού, επειδή είναι δύσκολο να γίνει διάκριση μεταξύ νόμιμης και κακόβουλης κυκλοφορίας. Οι οργανισμοί πρέπει να ενσωματώσουν διαδικασίες ασφάλειας (αξιολόγηση του κινδύνου, μοντελοποίηση απειλών, ελέγχους διείσδυσης,...) καθ 'όλη τη διάρκεια του κύκλου ζωής του λογισμικού.

Ο Gary McGraw [4] παρουσίασε 7 βέλτιστες πρακτικές για την παραγωγή ασφαλούς λογισμικού, τις οποίες ονόμασε σημεία επαφής

(touchpoints). Στην εικόνα 2 παρουσιάζονται αυτά τα σημεία επαφής, ανάλογα με τα υποπροϊόντα του κύκλου ζωής του λογισμικού στα οποία εφαρμόζονται. Επίσης έχουν ταξινομηθεί (1-7) ανάλογα με τη σημασία τους και συνεπώς με τη σειρά που προτείνεται να εφαρμοστούν από έναν οργανισμό που παράγει λογισμικό.



Εικόνα 2: Μέτρα ασφάλειας στα υποπροϊόντα λογισμικού

Όπως φαίνεται παραπάνω, η χρήση εργαλείων στατικής ανάλυσης πηγαίου κώδικα, έχει το μεγαλύτερο βάρος και θεωρείται το 1ο σημείο επαφής που πρέπει να υιοθετηθεί για την ανάπτυξη ασφαλούς λογισμικού. Αυτό συμβαίνει γιατί όλα τα έργα λογισμικού, παράγουν τουλάχιστον ένα προϊόν: πηγαίο κώδικα. Σε επίπεδο κώδικα, η εστίαση γίνεται στα σφάλματα ανάπτυξης (bugs) και ειδικότερα σε αυτά που μπορούν να ανιχνεύσουν τα εργαλεία στατικής ανάλυσης. Τα σφάλματα ανάπτυξης είναι πολλά και κοινά και περιλαμβάνουν σοβαρές ευπάθειες, όπως την περιβόητη υπερχείλιση στοίβας (buffer overflow), η οποία οφείλει την ύπαρξή της στη χρήση (ή κατάχρηση) ευάλωτων APIs (π.χ. gets(), strcpy(), και ούτω καθεξής στη γλώσσα C). Οι διαδικασίες επισκόπησης κώδικα, η χειροκίνητη και (σημαντικότερα) η αυτοματοποιημένη, με ένα εργαλείο στατικής ανάλυσης, προσπαθεί να εντοπίσει σφάλματα ασφαλείας πριν από την παράδοση του λογισμικού.

Φυσικά, μία μόνο τεχνική, όποια κι αν είναι αυτή, δεν μπορεί να λύσει όλα τα προβλήματα. Η στατική ανάλυση πηγαίου κώδικα είναι μια αναγκαία, αλλά όχι επαρκή πρακτική, για την παραγωγή ασφαλούς λογισμικού. Τα σφάλματα ανάπτυξης είναι ένα πραγματικό πρόβλημα (ειδικά στις γλώσσες C και C++), αλλά τα σφάλματα αρχιτεκτονικής είναι εξίσου σημαντικά. Η εφαρμογή της στατικής ανάλυσης κώδικα είναι μια εξαιρετικά χρήσιμη δραστηριότητα, αλλά δεδομένου ότι αυτού του είδους ο έλεγχος μπορεί να προσδιορίσει μόνο σφάλματα ανάπτυξης, στην καλύτερη περίπτωση μπορεί να αποκαλύψει περίπου το 50% των προβλημάτων ασφάλειας. Τα προβλήματα αρχιτεκτονικής είναι πολύ δύσκολο (και συνήθως αδύνατο) να βρεθούν, εξετάζοντας μόνο τον πηγαίο κώδικα. Αυτό είναι ιδιαίτερα αληθές στα σύγχρονα συστήματα, τα οποία αποτελούνται από εκατοντάδες χιλιάδες γραμμές κώδικα. Μια ολοκληρωμένη προσέγγιση για την ασφάλεια του λογισμικού, περιλαμβάνει ολιστικά την επισκόπηση κώδικα και την ανάλυση της αρχιτεκτονικής.

Από τη φύση της, η επισκόπηση πηγαίου κώδικα απαιτεί τη γνώση του κώδικα. Ένας επαγγελματίας της ασφάλειας πληροφοριών, με μικρή εμπειρία στο γράψιμο και τη μεταγλώττιση λογισμικού, θα είναι χρήσιμος ελάχιστα κατά την επισκόπηση κώδικα. Αυτό το στάδιο είναι καλύτερα να αφεθεί στα χέρια των μελών της ομάδας ανάπτυξης, ειδικά αν έχουν στη διάθεσή τους ένα σύγχρονο εργαλείο στατικής ανάλυσης. Με εξαίρεση επαγγελματίες της ασφάλειας πληροφοριών, οι οποίοι έχουν μεγάλη εμπειρία σε γλώσσες προγραμματισμού και ανάλυσης ευπαθειών στο επίπεδο του κώδικα, δεν υπάρχει ρόλος για τον ειδικό της ασφάλειας δικτύων στη φάση της επισκόπησης κώδικα. Αυτό μπορεί να προκαλέσει μεγάλη έκπληξη στους οργανισμούς που προσπαθούν να εφαρμόσουν ασφάλεια λογισμικού, μέσω του τμήματος ασφάλειας πληροφοριακών συστημάτων. Ακόμα κι αν η ιδέα της επιβολής ασφάλειας είναι στέρεα, για να είναι επιτυχής η επιβολή της ασφάλειας σε επίπεδο κώδικα, απαιτείται πρακτική εμπειρία στον ίδιο τον κώδικα. Το πρόβλημα είναι ότι οι περισσότεροι προγραμματιστές έχουν μικρή ιδέα για το ποια σφάλματα πρέπει να ψάξουν και τι να κάνουν ώστε να τα διορθώσουν, αφού τα ανακαλύψουν.

1.2 Σκοπός και δομή της εργασίας

Όπως παρουσιάστηκε παραπάνω, η ασφάλεια λογισμικού είναι πολύ κρίσιμο τμήμα της ασφάλειας πληροφοριακών συστημάτων. Σκοπός αυτής της εργασίας είναι η διερεύνηση καταρχήν των σημαντικότερων και συχνότερων σφαλμάτων ασφάλειας στο λογισμικό και η περιγραφή της πιο διαδεδομένης μεθόδου εύρεσής τους, δηλαδή της στατικής ανάλυσης πηγαίου κώδικα. Τέλος, γίνεται μια αξιολόγηση της χρήσης της στατικής ανάλυσης, εφαρμόζοντας κάποια σχετικά εργαλεία, πρώτα σε κάποια δοκιμαστικά προβλήματα και έπειτα σε πραγματικές εφαρμογές λογισμικού.

Αντίστοιχα, στο επόμενο κεφάλαιο ακολουθεί ο κατάλογος με τα κρισιμότερα σφάλματα ασφάλειας στο λογισμικό και στο κεφάλαιο 3 αναλύεται η μέθοδος της στατικής ανάλυσης. Τέλος, στο κεφάλαιο 4 παρουσιάζονται τα αποτελέσματα της εκτέλεσης των εργαλείων στις εφαρμογές λογισμικού και τα γενικά συμπεράσματα.

ΚΕΦΑΛΑΙΟ 2

2 ΣΦΑΛΜΑΤΑ ΑΣΦΑΛΕΙΑΣ ΣΤΟ ΛΟΓΙΣΜΙΚΟ

2.1 Ταξινομία των σφαλμάτων ασφάλειας

Όλοι οι τομείς της επιστήμης, ωφελούνται από την ταξινόμηση του πεδίου έρευνας. Έτσι και στην ασφάλεια λογισμικού, έχουν προταθεί αρκετά συστήματα ταξινόμησης από τη δεκαετία του 1980 [5]. Μία γενικά αποδεκτή σύγχρονη προσέγγιση είναι οι επτά (και μία) κατηγορίες σφαλμάτων που παρουσίασαν οι Tsipenyuk, Chess και McGraw [6]:

- **Επικύρωση των εισόδων και αναπαράσταση.** Αυτού του είδους τα προβλήματα προκαλούνται από μετα-χαρακτήρες, εναλλακτικές κωδικοποιήσεις και αριθμητικές παραστάσεις. Οι ευπάθειες προκύπτουν από την εμπιστοσύνη στις εισόδους του λογισμικού. Οι σχετικές αδυναμίες περιλαμβάνουν: “Buffer Overflows,” “Cross-Site Scripting” επιθέσεις,“SQL Injection” και πολλά άλλα.

- **Κατάχρηση προγραμματιστικών διεπαφών (API).** Ένα API είναι μια σύμβαση μεταξύ καλούντος και καλούμενου. Οι πιο κοινές μορφές κατάχρησης API προκαλούνται από τον καλούντα, παραλείποντας να τιμήσει το δικό του μέρος της σύμβασης. Για παράδειγμα, εάν ένα πρόγραμμα αποτύχει να καλέσει τη chdir() μετά την κλήση της chroot(), παραβιάζει τη σύμβαση που καθορίζει το πώς αλλάζει ο ενεργός κατάλογος με ασφαλή τρόπο. Ένα άλλο καλό παράδειγμα κατάχρησης βιβλιοθήκης, είναι η αναμονή του καλούντα μιας αξιόπιστης εγγραφής DNS από τον καλούμενο. Σε αυτήν την περίπτωση ο καλών κάνει κακή χρήση της API, κάνοντας υποθέσεις για τη συμπεριφορά της (π.χ. ότι μπορεί να χρησιμοποιηθεί για αυθεντικοποίηση). Κάποιος μπορεί επίσης να παραβιάσει το συμβόλαιο καλούντος-καλούμενου, από την άλλη πλευρά. Για παράδειγμα, εάν ένας προγραμματιστής φτιάξει μια κλάση η

οποία κληρονομεί τη SecureRandom και αυτή επιστρέφει μια μη-τυχαία τιμή, η σύμβαση παραβιάζεται.

- **Λειτουργικότητα ασφάλειας.** Η ασφάλεια λογισμικού δεν είναι το λογισμικό ασφάλειας. Σε αυτήν την κατηγορία ανήκουν προβλήματα σχετικά με την αυθεντικοποίηση, τον έλεγχο πρόσβασης, την εμπιστευτικότητα, την κρυπτογράφηση και τη διαχείριση προνομίων.

- **Χρόνος και κατάσταση.** Η κατανεμημένη επεξεργασία σχετίζεται με τον χρόνο και την κατάσταση. Δηλαδή, για να είναι δυνατή η επικοινωνία περισσοτέρων από ένα συστατικών, η κατάσταση πρέπει να διαμοιραστεί και αυτό απαιτεί χρόνο. Οι περισσότεροι προγραμματιστές νομίζουν ότι ένα νήμα ελέγχου εκτελεί το σύνολο του προγράμματος, με τον ίδιο τρόπο που θα έκαναν τη δουλειά οι ίδιοι. Οι σύγχρονοι υπολογιστές ωστόσο, αλλάζουν μεταξύ των καθηκόντων πολύ γρήγορα και στα πολυπύρηνα, πολυεπεξεργαστικά ή τα κατανεμημένα συστήματα, δύο γεγονότα μπορούν να λάβουν χώρα ακριβώς την ίδια στιγμή. Τα ελαττώματα αναπληρώνουν το χάσμα μεταξύ του μοντέλου του προγραμματιστή για το πώς ένα πρόγραμμα εκτελείται και του τι συμβαίνει στην πραγματικότητα. Αυτά τα ελαττώματα σχετίζονται με απρόσμενες αλληλεπιδράσεις μεταξύ των νημάτων, των διεργασιών, του χρόνου και των πληροφοριών. Αυτές οι αλληλεπιδράσεις συμβαίνουν μέσω της κοινής κατάστασης: Σηματοφόρων, μεταβλητών, συστήματος αρχείων, και βασικά όποιου μέσου μπορεί να αποθηκεύσει πληροφορίες.

- **Σφάλματα.** Τα σφάλματα και η διαχείρισή τους, αντιπροσωπεύουν μια κλάση API. Τα προβλήματα σχετικά με τη διαχείριση σφαλμάτων είναι τόσο συχνά, ώστε τους αξίζει μια ξεχωριστή κατηγορία. Όπως και στην κατάχρηση API, υπάρχουν δύο τρόποι να προκύψουν σχετικές ευπάθειες. Ο πιο συχνός είναι η ελλιπής (ή καθόλου) διαχείριση σφαλμάτων. Ο δεύτερος τρόπος είναι να προκαλούνται σφάλματα που ή παρουσιάζουν πάρα πολλές πληροφορίες (στον επιτιθέμενο) ή είναι δύσκολο να διαχειριστούν.

- **Ποιότητα κώδικα.** Η χαμηλή ποιότητα κώδικα οδηγεί σε απρόβλεπτη συμπεριφορά. Από τη σκοπιά του χρήστη, αυτό συνήθως εμφανίζει φτωχή χρηστικότητα. Στον επιτιθέμενο, αυτό παρέχει μια ευκαιρία να πιέσει το σύστημα με απρόβλεπτους τρόπους.

- **Ενθυλάκωση.** Η ενθυλάκωση είναι η χάραξη ισχυρών ορίων. Σε έναν περιηγητή web, αυτό μπορεί να είναι η εξασφάλιση ότι ένας κινητός κώδικας, δεν μπορεί να καταχραστεί τον δικό σας κινητό κώδικα. Στο διακομιστή μπορεί να είναι η διαφοροποίηση μεταξύ επικυρωμένων και μη, δεδομένων διαφορετικών χρηστών ή μεταξύ δεδομένων τα οποία ένας χρήστης μπορεί να προσπελάσει και αυτών που δεν μπορεί.

- **Περιβάλλον.** Αυτή η κατηγορία περιλαμβάνει οτιδήποτε είναι εκτός του πηγαίου κώδικα, αλλά είναι κρίσιμο στην ασφάλεια του προϊόντος λογισμικού.

2.2 CWE/SANS Top 25

Σε αυτήν την ενότητα θα γίνει μια συνοπτική παρουσίαση των περισσότερο κρίσιμων σφαλμάτων λογισμικού, όπως αυτά εμφανίζονται στην λίστα του 2011 CWE/SANS Top 25 Most Dangerous Software Errors [7]. Σε αυτήν τη λίστα περιέχονται τα πιο διαδεδομένα και κρίσιμα λάθη, τα οποία μπορούν να οδηγήσουν σε σοβαρές ευπάθειες στο λογισμικό. Συνήθως είναι εύκολο να τα βρει κανείς και εύκολο να τα εκμεταλλευτεί. Είναι επικίνδυνα, διότι συχνά επιτρέπουν στους επιτιθέμενους να ελέγχουν πλήρως το λογισμικό, να κλέψουν τα δεδομένα, ή να αποτρέψουν τελείως τη λειτουργία του λογισμικού.

Η λίστα των 25 κρισιμότερων σφαλμάτων είναι ένα εργαλείο για την εκπαίδευση και την ευαισθητοποίηση, ώστε να βοηθήσει τους προγραμματιστές να αποτρέψουν τα είδη των ευπαθειών που μαστίζουν τη βιομηχανία λογισμικού, με τον εντοπισμό και την αποφυγή τους, πριν καν την παράδοσή του λογισμικού. Οι πελάτες του λογισμικού μπορούν να χρησιμοποιήσουν την ίδια λίστα για να τους βοηθήσει να ζητήσουν πιο ασφαλές λογισμικό. Οι ερευνητές

στον τομέα της ασφάλειας λογισμικού, μπορούν να χρησιμοποιήσουν τη λίστα για να επικεντρωθούν σε ένα στενό, αλλά σημαντικό υποσύνολο όλων των γνωστών προβλημάτων ασφάλειας. Τέλος, οι διαχειριστές του λογισμικού και οι υπεύθυνοι ασφάλειας σε έναν οργανισμό, μπορούν να χρησιμοποιήσουν τον κατάλογο ως μέτρο σύγκρισης της προόδου των προσπαθειών τους, να εξασφαλίσουν το λογισμικό τους.

Ο κατάλογος είναι το αποτέλεσμα της συνεργασίας μεταξύ του ινστιτούτου SANS, της εταιρείας MITRE και πολλών κορυφαίων ειδικών της ασφάλειας λογισμικού στις ΗΠΑ και την Ευρώπη. Αξιοποιεί την εμπειρία στην ανάπτυξη των κορυφαίων 20 ελέγχων ασφάλειας του SANS [8] και της κοινής καταμέτρησης αδυναμιών (Common Weakness Enumeration) του MITRE [9]. Η MITRE διατηρεί την ιστοσελίδα του CWE, με την υποστήριξη του τμήματος κυβερνοασφάλειας του αμερικανικού υπουργείου εθνικής ασφάλειας, παρουσιάζοντας λεπτομερείς περιγραφές από τα κορυφαία 25 λάθη προγραμματισμού, μαζί με την έγκυρη καθοδήγηση για τον μετριασμό και την αποφυγή τους. Η ιστοσελίδα CWE περιέχει στοιχεία για περισσότερα από 800 προγραμματιστικά λάθη, λάθη σχεδιασμού και αρχιτεκτονικής, λάθη που μπορούν να οδηγήσουν σε εκμεταλλεύσιμες ευπάθειες.

Η λίστα των 25 κρισιμότερων του 2011 βελτιώνει τον κατάλογο του 2010, αλλά το πνεύμα και οι στόχοι παραμένουν οι ίδιοι. Ο νέος κατάλογος είναι ταξινομημένος χρησιμοποιώντας στοιχεία από πάνω από 20 διαφορετικούς οργανισμούς, οι οποίοι αξιολόγησαν κάθε αδυναμία με βάση την επικράτηση, τη σημασία και την πιθανότητα εκμετάλλευσης. Χρησιμοποιεί το κοινό σύστημα βαθμολόγησης αδυναμίας (Common Weakness Scoring System) για να αποτιμήσει και να κατατάξει τα τελικά αποτελέσματα.

Μια άλλη διάσημη αντίστοιχη λίστα, η οποία επικεντρώνεται στις εφαρμογές web και είναι υποσύνολο της CWE/SANS Top 25, είναι το top 10 του οργανισμού Open Web Application Security Project (OWASP) [10]. Στον πίνακα που ακολουθεί, παρουσιάζεται η σχέση των κρισιμότερων σφαλμάτων των δύο αυτών λιστών:

OWASP Top 10 2013	CWE/SANS Top 25 2011
A1-Injection	1. CWE-89, 2. CWE-78
A2-Broken Authentication and Session Management	5. CWE-306, 21. CWE-307, 7. CWE-798, 10. CWE-807
A3-Cross-Site Scripting (XSS)	4. CWE-79
A4-Insecure Direct Object References	6. CWE-862, 15. CWE-863, 13. CWE-22, 9. CWE-434, 16. CWE-829
A5-Security Misconfiguration	11. CWE-250, 17. CWE-732, 25. CWE-759
A6-Sensitive Data Exposure	8. CWE-311, 19. CWE-327
A7-Missing Function Level Access Control	5. CWE-306
A8-Cross-Site Request Forgery (CSRF)	12. CWE-352
A9-Using Components with Known Vulnerabilities	-
A10-Unvalidated Redirects and Forwards	22. CWE-601
-	3. CWE-120, 14. CWE-494, 18. CWE-676, 20. CWE-131, 23. CWE-134, 24. CWE-190

2.2.1 CWE-89. Ανεπαρκής απενεργοποίηση ειδικών στοιχείων σε μια εντολή SQL (SQL Injection)

Χωρίς επαρκή απομάκρυνση ή απενεργοποίηση του SQL κώδικα από τις εισόδους που ελέγχονται από τον χρήστη, το παραγόμενο ερώτημα SQL μπορεί να μεταφράσει αυτές τις εισόδους σε εκτελέσιμο κώδικα, αντί για απλά δεδομένα χρήστη. Αυτό μπορεί να προκαλέσει την αλλαγή της λογικής του ερωτήματος για να παρακαμφθούν οι έλεγχοι ασφάλειας, ή για να εισαχθούν πρόσθετες εντολές οι οποίες τροποποιούν τη βάση δεδομένων και ενδεχομένως, να εκτελεστούν εντολές στο σύστημα.

Το SQL injection έχει γίνει ένα κοινό θέμα στις εφαρμογές web οι οποίες στηρίζονται σε μια βάση δεδομένων. Η ευπάθεια αυτή εντοπίζεται εύκολα και εκμεταλλεύεται εύκολα, οπότε οποιαδήποτε ιστοσελίδα ή πακέτο λογισμικού με ακόμη και ελάχιστη βάση

χρηστών, είναι πιθανό να υποστεί μια απόπειρα επίθεσης αυτού του είδους. Αυτό το σφάλμα βασίζεται στο γεγονός ότι η γλώσσα SQL, δεν κάνει καμία πραγματική διάκριση μεταξύ του κώδικα και των δεδομένων.

Παράδειγμα:
Ο ακόλουθος κώδικας C# σχηματίζει δυναμικά και εκτελεί ένα ερώτημα SQL που ψάχνει για στοιχεία που ταιριάζουν με ένα συγκεκριμένο όνομα. Το ερώτημα περιορίζει τα στοιχεία που εμφανίζονται, σε αυτά που σχετίζονται με το όνομα χρήστη που έχει αυθεντικοποιηθεί.

```
...
string userName = ctx.getAuthenticatedUserName();
string query = "SELECT * FROM items WHERE owner =
'" + userName + "' AND itemname = '" +
ItemName.Text + "'";
sda = new SqlDataAdapter(query, conn);
DataTable dt = new DataTable();
sda.Fill(dt);
...
```

Το SQL ερώτημα που θέλει να εκτελέσει ο παραπάνω κώδικας είναι το:

```
SELECT * FROM items WHERE owner = <userName> AND
itemname = <itemName>;
```

Ωστόσο, επειδή το ερώτημα παράγεται δυναμικά συνενώνοντας μια σταθερή συμβολοσειρά με την είσοδο από το χρήστη, το ερώτημα θα λειτουργήσει σωστά, μόνο αν το itemName δεν περιέχει τον χαρακτήρα των μονών εισαγωγικών. Αν ένας επιτιθέμενος με το όνομα χρήστη wiley, εισάγει τη συμβολοσειρά:

```
name' OR 'a'='a
```

στο itemName, το ερώτημα θα γίνει ως εξής:

```
SELECT * FROM items WHERE owner = 'wiley' AND
itemname = 'name' OR 'a'='a';
```

Το αποτέλεσμα της εκτέλεσης θα είναι να παρακαμφθεί η απαίτηση, ότι το ερώτημα πρέπει να επιστρέφει μόνο τα στοιχεία που σχετίζονται με τον αυθεντικοποιημένο χρήστη.

2.2.2 CWE-78. Ανεπαρκής απενεργοποίηση ειδικών στοιχείων σε μια εντολή του λειτουργικού συστήματος (OS Injection)

Αυτό θα μπορούσε να επιτρέψει στους επιτιθέμενους να εκτελέσουν απροσδόκητες, επικίνδυνες εντολές, απευθείας στο λειτουργικό σύστημα. Η αδυναμία αυτή μπορεί να οδηγήσει σε μια ευπάθεια σε περιβάλλοντα, στα οποία ο επιτιθέμενος δεν έχει άμεση πρόσβαση στο λειτουργικό σύστημα, όπως στις εφαρμογές web. Εναλλακτικά, εάν η αδυναμία εμφανίζεται σε ένα προνομιακό πρόγραμμα, θα μπορούσε να επιτρέψει στον εισβολέα να εκτελέσει εντολές που κανονικά δεν είναι προσβάσιμες ή να εκτελέσει εναλλακτικές εντολές με προνόμια τα οποία δεν διαθέτει. Το πρόβλημα επιδεινώνεται αν η ευπαθής διαδικασία δεν εφαρμόζει το μέτρο των ελαχίστων απαιτούμενων δικαιωμάτων, γιατί έτσι οι κακόβουλες εντολές μπορεί να τρέξουν με ειδικά προνόμια συστήματος και να προκαλέσουν πολύ μεγάλη ζημιά.

Υπάρχουν τουλάχιστον δύο υπότυποι αυτού του σφάλματος:

1. Η εφαρμογή προτίθεται να εκτελέσει ένα ενιαίο, σταθερό πρόγραμμα που είναι υπό τον έλεγχό της. Σκοπεύει να χρησιμοποιήσει εξωτερικά παρεχόμενες εισόδους ως παραμέτρους στο εν λόγω πρόγραμμα. Για παράδειγμα, η εφαρμογή θα μπορούσε να χρησιμοποιήσει την εντολή nslookup [HOSTNAME] για να καλέσει το nslookup, επιτρέποντας στο χρήστη να εισάγει ένα όνομα, το οποίο θα χρησιμοποιηθεί ως παράμετρος. Οι επιτιθέμενοι, δεν μπορούν να αποτρέψουν την εκτέλεση του nslookup. Ωστόσο, εάν η εφαρμογή δεν αφαιρεί τους διαχωριστές εντολών από την παράμετρο του ονόματος του μηχανήματος, ένας κακόβουλος χρήστης θα μπορούσε να εισάγει διαχωριστικά στην παράμετρο και αυτό να του επιτρέψει να εκτελέσει οποιοδήποτε άλλο πρόγραμμα, μετά την εκτέλεση του nslookup.

2. Η εφαρμογή δέχεται μια είσοδο την οποία χρησιμοποιεί για την πλήρη επιλογή του προγράμματος που θα εκτελέσει, καθώς και ποιες εντολές θα χρησιμοποιήσει. Η εφαρμογή ανακατευθύνει απλά όλη την εντολή στο λειτουργικό σύστημα. Για παράδειγμα, η εφαρμογή θα μπορούσε να χρησιμοποιήσει το exec([εντολή]), για να εκτελέσει την [εντολή] που παρέχεται από το χρήστη. Εάν η εντολή είναι υπό τον έλεγχο του εισβολέα, τότε αυτός μπορεί να εκτελέσει αυθαίρετες εντολές ή προγράμματα. Εάν η εντολή εκτελείται με συναρτήσεις όπως η exec() και η CreateProcess(), μπορεί να μην είναι σε θέση να συνδυάσει πολλαπλές εντολές μαζί στην ίδια γραμμή.

Από τη σκοπιά της ευπάθειας, αυτές οι παραλλαγές αντιπροσωπεύουν διακριτά προγραμματιστικά σφάλματα. Στην πρώτη παραλλαγή, ο προγραμματιστής προτίθεται σαφώς να χρησιμοποιήσει μη έμπιστη είσοδο, ως παράμετρο σε κάποια εντολή. Στη δεύτερη παραλλαγή, ο προγραμματιστής δεν θέλει να είναι προσβάσιμη η εντολή από τα μη έμπιστα μέλη, αλλά προφανώς δεν προέβλεψε τους εναλλακτικούς τρόπους, με τους οποίους ένας κακόβουλος χρήστης, μπορεί να παρέχει είσοδο.

Παράδειγμα:
Αυτός ο κώδικας σκοπεύει να λάβει το όνομα χρήστη ως παράμετρο και να παρουσιάσει τα περιεχόμενα του καταλόγου του χρήστη. Ανήκει στην πρώτη υποκατηγορία του σφάλματος.

```
$userName = $_POST["user"];
$command = 'ls -l /home/' . $userName;
system($command);
```

Η μεταβλητή $userName δεν ελέγχεται για κακόβουλη είσοδο. Ένας επιτιθέμενος, θα μπορούσε να εισάγει οποιαδήποτε εντολή του λειτουργικού συστήματος, όπως την:

```
;rm -rf /
```

Το αποτέλεσμα είναι η μεταβλητή $command να λάβει την τιμή:

```
ls -l /home/;rm -rf /
```

Επειδή στο λειτουργικό σύστημα UNIX, το ερωτηματικό είναι διαχωριστικό εντολών, θα εκτελεστεί πρώτα η εντολή ls και μετά η εντολή rm, η οποία θα διαγράψει ολόκληρο το σύστημα αρχείων.

2.2.3 CWE-120. Αντιγραφή προσωρινής αποθήκευσης, χωρίς έλεγχο του μεγέθους της εισόδου (Buffer Overflow)

Μια κατάσταση υπερχείλισης μιας προσωρινής αποθήκευσης, υπάρχει όταν ένα πρόγραμμα προσπαθεί να βάλει περισσότερα δεδομένα σε αυτήν από ότι μπορεί να κρατήσει, ή όταν ένα πρόγραμμα προσπαθεί να εισάγει τα δεδομένα σε μια περιοχή μνήμης έξω από τα όρια μιας προσωρινής αποθήκευσης. Το απλούστερο είδος του λάθους και η πιο κοινή αιτία των υπερχειλίσεων, είναι η «κλασική» περίπτωση κατά την οποία το πρόγραμμα αντιγράφει την προσωρινή αποθήκευση, χωρίς να περιορίζει τον όγκο της αντιγραφής. Και άλλες παραλλαγές υπάρχουν, αλλά η ύπαρξη μιας κλασικής υπερχείλισης, υποδηλώνει σαφώς ότι ο προγραμματιστής δεν εξετάζει καν τους πιο βασικούς ελέγχους ασφάλειας.

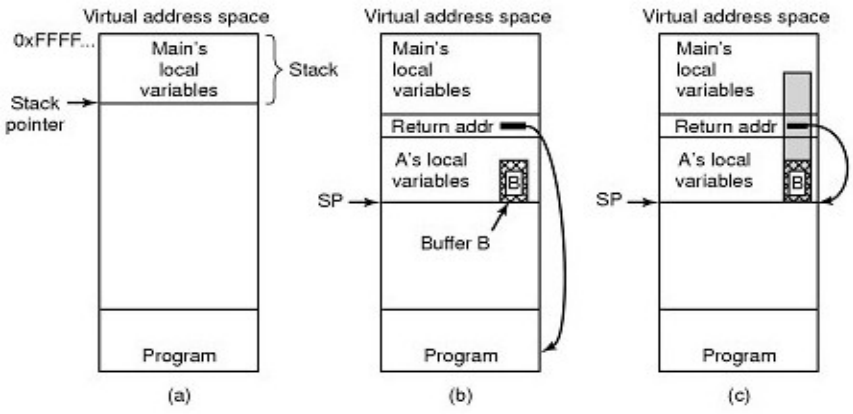
Αυτή είναι η πιο κοινή και μία από τις πιο επικίνδυνες ευπάθειες που υπάρχουν. Το αποτέλεσμα μιας υπερχείλισης είναι οτιδήποτε, από μια κατάρρευση, στον πλήρη έλεγχο της εφαρμογής από τον εισβολέα και αν η εφαρμογή εκτελείται με προνομιακά δικαιώματα χρήστη (root, administrator, ή local system), τότε στον πλήρη έλεγχο ολόκληρου του λειτουργικού συστήματος και όλων των χρηστών που είναι συνδεδεμένοι εκείνη τη στιγμή, ή θα συνδεθούν αργότερα. Εάν η εν λόγω εφαρμογή είναι μια υπηρεσία δικτύου, το αποτέλεσμα της ευπάθειας θα μπορούσε να είναι ένα σκουλήκι (worm). Το πρώτο γνωστό σκουλήκι στο διαδίκτυο, αξιοποιούσε μια υπερχείλιση στην υπηρεσία finger του διακομιστή και είναι γνωστό ως το Robert T. Morris (ή απλά Morris) finger worm. Αν και θα περίμενε κανείς πως θα είχαμε μάθει να αποφεύγουμε αυτού του είδους τα προβλήματα, μετά από μια σχεδόν κατάρρευση του διαδικτύου το 1988, εξακολουθούν να εμφανίζονται συχνές αναφορές για υπερχειλίσεις, σε διαφόρων ειδών λογισμικό. Η C είναι η πιο κοινή γλώσσα που χρησιμοποιείται για την δημιουργία

υπερχειλίσεων και ακολουθεί στενά η C++. Οι πιο σύγχρονες γλώσσες προγραμματισμού υψηλότερου επιπέδου, δεν επιτρέπουν την άμεση προσπέλαση της μνήμης από τον προγραμματιστή, γενικά με ένα σεβαστό κόστος απόδοσης [11].

Παράδειγμα:
Για την εξήγηση της ευπάθειας θα χρησιμοποιήσουμε τον παρακάτω κώδικα C:

```
#include <stdio.h>
void DontDoThis(char* input)
{
	char buf[16];
	strcpy(buf, input);
	printf("%s\n", buf);
}
int main(int argc, char* argv[])
{
	// So we're not checking arguments
	// What do you expect from an app that uses strcpy?
	DontDoThis(argv[1]);
	return 0;
}
```

Όταν εκτελεστεί το παραπάνω πρόγραμμα και κληθεί η συνάρτηση DontDoThis, η στοίβα θα περιέχει (από κάτω προς τα πάνω) τις τοπικές μεταβλητές της main (a), μετά τη διεύθυνση επιστροφής εκτέλεσης και στο τέλος τις τοπικές μεταβλητές της συνάρτησης DontDoThis (b). Αν strcpy προσπαθήσει να γράψει περισσότερα bytes από το μέγεθος της buf, θα αντικατασταθεί η διεύθυνση επιστροφής (c).



Εικόνα 3: Η στοίβα κατά την κλήση της συνάρτησης DontDoThis

Χρησιμοποιώντας αυτήν την τεχνική, ένας εισβολέας μπορεί να ανακατευθύνει την εκτέλεση και να τρέξει κακόβουλο κώδικα. Εάν το αρχικό πρόγραμμα τρέχει με δικαιώματα διαχειριστή, τότε ο εισβολέας θα αποκτήσει τα δικαιώματα πρόσβασης του διαχειριστή του συστήματος.

2.2.4 CWE-79. Ανεπαρκής απενεργοποίηση εισόδου κατά την παραγωγή ιστοσελίδων (Cross-site Scripting)

Το Cross- site scripting (XSS) μπορεί να συμβεί όταν:

1. Μη αξιόπιστα δεδομένα εισέρχονται σε μια εφαρμογή web , τυπικά από μια αίτηση web.
2. Η web εφαρμογή παράγει δυναμικά μια ιστοσελίδα, που περιέχει αυτά τα αναξιόπιστα δεδομένα.
3. Κατά τη διάρκεια της παραγωγής της σελίδας, η εφαρμογή δεν εμποδίζει τα δεδομένα που περιέχουν εκτελέσιμο από ένα πρόγραμμα περιήγησης περιεχόμενο, όπως η JavaScript, τα HTML tags, τα HTML attributes, τα γεγονότα του ποντικιού, Flash, ActiveX, κλπ.
4. Ένα θύμα επισκέπτεται την παραγόμενη ιστοσελίδα μέσω ενός περιηγητή, το οποίο περιέχει κακόβουλο κώδικα που εισήχθηκε χρησιμοποιώντας τα αναξιόπιστα δεδομένα.
5. Δεδομένου ότι ο κώδικας προέρχεται από μια ιστοσελίδα που στάλθηκε από τον διακομιστή, ο περιηγητής του θύματος εκτελεί τον κακόβουλο κώδικα στο πλαίσιο του τομέα (domain) του διακομιστή.
6. Αυτό παραβιάζει ουσιαστικά την πολιτική ίδιας καταγωγής του περιηγητή, η οποία ορίζει ότι ο κώδικας ενός τομέα, δεν μπορεί να έχει πρόσβαση σε πόρους ή να τρέξει κώδικα άλλου τομέα.

Υπάρχουν τρία κύρια είδη XSS:

Τύπος 1 : Reflected XSS (ή Non-Persistent)
Ο διακομιστής διαβάζει δεδομένα απευθείας από την αίτηση HTTP και τα αντανακλά πίσω στην HTTP απάντηση. Οι εκμεταλλεύσεις αυτών των προβλημάτων συμβαίνουν όταν ένας εισβολέας

προκαλεί το θύμα να παρέχει επικίνδυνο περιεχόμενο σε μια ευάλωτη εφαρμογή web, το οποίο στη συνέχεια αντανακλάται πίσω στο θύμα και εκτελείται από τον περιηγητή. Ο πιο κοινός μηχανισμός για την παροχή κακόβουλου περιεχομένου, είναι να συμπεριληφθεί ως παράμετρος σε μια διεύθυνση URL που έχει αναρτηθεί δημόσια ή έχει σταλεί με e-mail απευθείας στο θύμα. Τα URLs που έχουν κατασκευαστεί με αυτόν τον τρόπο, αποτελούν τον πυρήνα πολλών συστημάτων phishing.

Τύπος 2 : Stored XSS (ή Persistent)

Η εφαρμογή αποθηκεύει επικίνδυνα δεδομένα σε μια βάση δεδομένων ή άλλο έμπιστο μέσο αποθήκευσης δεδομένων. Σε μεταγενέστερο χρόνο, τα επικίνδυνα δεδομένα διαβάζονται πίσω στην εφαρμογή και περιλαμβάνονται στο δυναμικό περιεχόμενο. Από την πλευρά του επιτιθέμενου, το ιδανικό μέρος για να κάνει την εισαγωγή κακόβουλου περιεχομένου, είναι σε μια περιοχή που εμφανίζεται είτε σε πολλούς χρήστες ή σε ιδιαίτερα ενδιαφέροντες χρήστες. Ενδιαφέροντες είναι συνήθως οι χρήστες που έχουν αυξημένα προνόμια στην εφαρμογή ή αλληλεπιδρούν με ευαίσθητα δεδομένα που είναι πολύτιμα για τον εισβολέα. Αν κάποιος από αυτούς τους χρήστες εκτελέσει το κακόβουλο περιεχόμενο, ο εισβολέας μπορεί να είναι σε θέση να εκτελέσει τις προνομιακές λειτουργίες για λογαριασμό του χρήστη ή να αποκτήσει πρόσβαση σε ευαίσθητα δεδομένα που ανήκουν στον χρήστη. Για παράδειγμα, ο εισβολέας θα μπορούσε να εισάγει XSS σε ένα μήνυμα καταγραφής, ώστε να εκτελεστεί όταν ένας διαχειριστής ανοίξει τα αρχεία καταγραφής.

Τύπος 0 : DOM-Based XSS

Στο DOM-based XSS, ο πελάτης πραγματοποιεί την εισαγωγή του XSS στη σελίδα. Στα άλλα είδη, ο διακομιστής εκτελεί την εισαγωγή. Αυτή η κατηγορία περιλαμβάνει γενικά έμπιστο κώδικα, ελεγχόμενο από τον διακομιστή, ο οποίος στέλνεται στον πελάτη (π.χ. Javascript για την επικύρωση της εισόδου σε μια φόρμα). Αν αυτός ο κώδικας επεξεργάζεται την είσοδο του χρήστη και την αντανακλά στην ιστοσελίδα, τότε είναι πιθανή αυτή η ευπάθεια.

Μόλις ο κακόβουλος κώδικας εγχέεται, ο εισβολέας μπορεί να εκτελέσει μια ποικιλία από κακόβουλες δραστηριότητες. θα μπορούσε να μεταφέρει προσωπικές πληροφορίες, όπως τα cookies

που μπορεί να περιλαμβάνουν πληροφορίες της συνόδου, από το μηχάνημα του θύματος στον εισβολέα. Θα μπορούσε να στείλει κακόβουλα αιτήματα σε μια ιστοσελίδα για λογαριασμό του θύματος, τα οποία θα μπορούσαν να είναι ιδιαίτερα επικίνδυνα για τον ιστότοπο, εάν το θύμα έχει δικαιώματα διαχειριστή. Phishing επιθέσεις θα μπορούσαν να χρησιμοποιηθούν για τη μίμηση αξιόπιστων ιστοσελίδων, ώστε να εξαπατηθεί το θύμα και να εισάγει κωδικούς πρόσβασης, επιτρέποντας στον εισβολέα να παραβιάσει το λογαριασμό του θύματος. Τέλος, ο επιτιθέμενος θα μπορούσε να εκμεταλλευτεί μια ευπάθεια στο πρόγραμμα περιήγησης του θύματος και να πάρει τον έλεγχό του μηχανήματός του (drive-by hacking).

Σε πολλές περιπτώσεις, η επίθεση μπορεί να ξεκινήσει χωρίς το θύμα καν να το γνωρίζει. Ακόμη και με προσεκτικούς χρήστες, οι εισβολείς χρησιμοποιούν συχνά μια ποικιλία μεθόδων για να κωδικοποιήσουν το κακόβουλο τμήμα της επίθεσης, όπως την κωδικοποίηση URL ή Unicode, οπότε η αίτηση φαίνεται λιγότερο ύποπτη.

Παράδειγμα:
Το ακόλουθο τμήμα κώδικα JSP, διαβάζει τον κωδικό υπαλλήλου στη μεταβλητή eid από την HTTP αίτηση και την εμφανίζει στο χρήστη.

```
<% String eid = request.getParameter("eid"); %>
...
Employee ID: <%= eid %>
```

Αντίστοιχα στη γλώσσα ASP.NET:

```
protected System.Web.UI.WebControls.TextBox Login;
protected          System.Web.UI.WebControls.Label
EmployeeID;
...
EmployeeID.Text = Login.Text;
... (HTML follows) ...
<p><asp:label    id="EmployeeID"    runat="server"
/></p>
```

Αυτά τα τμήματα θα λειτουργήσουν σωστά, μόνο αν η παράμετρος του κωδικού υπαλλήλου, είναι τυπικό κείμενο. Αν όμως η παράμετρος περιέχει μετα-χαρακτήρες ή πηγαίο κώδικα, αυτός θα εκτελεστεί στον περιηγητή του χρήστη, όταν αυτός εμφανίσει την HTTP απάντηση.

2.2.5 CWE-306. Ελλιπής αυθεντικοποίηση σε κρίσιμη συνάρτηση

Το λογισμικό δεν αυθεντικοποιεί το χρήστη πριν εκτελέσει κάποια λειτουργία, η οποία απαιτεί μια ταυτότητα χρήστη ή καταναλώνει συγκεκριμένους πόρους.

2.2.6 CWE-862. Ελλιπής εξουσιοδότηση

Δεδομένου ενός αυθεντικοποιημένου χρήστη, εξουσιοδότηση είναι η διαδικασία που καθορίζει αν αυτός ο χρήστης έχει δικαίωμα πρόσβασης ενός συγκεκριμένου πόρου. Όταν αυτοί οι έλεγχοι δεν εφαρμόζονται, οι χρήστες μπορεί να αποκτήσουν πρόσβαση σε δεδομένα ή να εκτελέσουν λειτουργίες, χωρίς να έχουν αυτό το δικαίωμα.

2.2.7 CWE-798. Ενσωματωμένα στον κώδικα διαπιστευτήρια

Τα ενσωματωμένα διαπιστευτήρια, δημιουργούν συνήθως μια σημαντική τρύπα που επιτρέπει σε έναν εισβολέα να παρακάμψει τον έλεγχο ταυτότητας, που έχει διαμορφωθεί από το διαχειριστή λογισμικού. Αυτή η τρύπα μπορεί να είναι δύσκολο να ανιχνευθεί από το διαχειριστή του συστήματος. Ακόμη και αν ανιχνευθεί, μπορεί να είναι δύσκολο να διορθωθεί, οπότε ο διαχειριστής πρέπει να απενεργοποιήσει το προϊόν εντελώς. Υπάρχουν δύο βασικές παραλλαγές:

Εισερχόμενη: Το λογισμικό περιέχει έναν μηχανισμό ελέγχου ταυτότητας ελέγχοντας τα διαπιστευτήρια εισόδου, συγκρίνοντάς τα με αυτά που έχουν ενσωματωθεί στον κώδικα.

Εξερχόμενη: Το λογισμικό συνδέεται με ένα άλλο σύστημα ή στοιχείο λογισμικού, χρησιμοποιώντας ενσωματωμένα διαπιστευτήρια για τη σύνδεση σε αυτό το στοιχείο.

Στην εισερχόμενη παραλλαγή, ένας προεπιλεγμένος λογαριασμός διαχείρισης έχει δημιουργηθεί και ένας απλός κωδικός πρόσβασης είναι κωδικοποιημένος μέσα στο προϊόν και συσχετισμένος με αυτόν τον λογαριασμό. Αυτός ο κωδικός είναι ο ίδιος για κάθε εγκατάσταση του προϊόντος και συνήθως δεν μπορεί να αλλάξει ή να απενεργοποιηθεί από τους διαχειριστές του συστήματος, χωρίς να χρειαστεί τροποποίηση του προγράμματος, ή αλλιώς την επιδιόρθωση του λογισμικού. Εάν ο κωδικός πρόσβασης έχει ανακαλυφθεί ή δημοσιευτεί (σύνηθες φαινόμενο στο διαδίκτυο), τότε ο καθένας με τη γνώση αυτού του κωδικού μπορεί να έχει πρόσβαση στο προϊόν. Τέλος, δεδομένου ότι όλες οι εγκαταστάσεις του λογισμικού θα έχουν τον ίδιο κωδικό πρόσβασης, ακόμα και μέσω διαφόρων οργανισμών, αυτό επιτρέπει μαζικές επιθέσεις, όπως τα σκουλήκια.

Η εξερχόμενη παραλλαγή ισχύει για συστήματα διεπαφής με τον χρήστη τα οποία αυθεντικοποιούνται σε μια back-end υπηρεσία. Αυτή η υπηρεσία μπορεί να απαιτεί ένα σταθερό κωδικό πρόσβασης που μπορεί εύκολα να ανακαλυφθεί. Ο προγραμματιστής μπορεί απλά να ενσωματώσει αυτόν τον κωδικό στο λογισμικό διεπαφής. Κάθε χρήστης του προγράμματος μπορεί να είναι σε θέση να εξάγει τον κωδικό πρόσβασης. Συστήματα πελάτη με ενσωματωμένους κωδικούς πρόσβασης αποτελούν ακόμη μεγαλύτερη απειλή, δεδομένου ότι η εξαγωγή ενός κωδικού πρόσβασης από ένα δυαδικό αρχείο είναι συνήθως πολύ απλή.

2.2.8 CWE-311. Ελλιπής κρυπτογράφηση ευαίσθητων δεδομένων

Το λογισμικό δεν κρυπτογραφεί ευαίσθητα ή κρίσιμα δεδομένα πριν την αποθήκευση ή τη μετάδοσή τους. Η έλλειψη σωστής κρυπτογράφησης απειλεί την εμπιστευτικότητα και ενδεχομένως την ακεραιότητα των δεδομένων.

2.2.9 CWE-434. Ανέβασμα επικίνδυνων αρχείων χωρίς περιορισμούς

Το λογισμικό επιτρέπει σε έναν επιτιθέμενο να ανεβάσει (upload) ή να μεταφέρει επικίνδυνα αρχεία, τα οποία μπορεί να εκτελεστούν αυτόματα στο περιβάλλον του λογισμικού.

2.2.10 CWE-807. Αποφάσεις ασφάλειας βασισμένες σε μη έμπιστη είσοδο

Οι προγραμματιστές μπορεί να υποθέσουν ότι οι είσοδοι, όπως τα cookies, οι μεταβλητές περιβάλλοντος και τα κρυφά πεδία μιας φόρμας, δεν μπορούν να τροποποιηθούν. Ωστόσο, ένας εισβολέας θα μπορούσε να αλλάξει αυτά τα δεδομένα, χρησιμοποιώντας προσαρμοσμένους πελάτες ή άλλες επιθέσεις. Η αλλαγή αυτή μπορεί να μην είναι δυνατό να ανιχνευτεί. Όταν οι αποφάσεις της ασφάλειας, όπως η αυθεντικοποίηση και η εξουσιοδότηση γίνεται με βάση τις τιμές των εισόδων αυτών, οι επιτιθέμενοι μπορούν να παρακάμψουν την ασφάλεια του λογισμικού. Χωρίς επαρκή κρυπτογράφηση, έλεγχο της ακεραιότητας, ή άλλο μηχανισμό, οποιαδήποτε είσοδος που προέρχεται από έναν ξένο δεν μπορεί να θεωρηθεί έμπιστη.

2.2.11 CWE-250. Εκτέλεση με άχρηστα προνόμια

Το λογισμικό εκτελεί μια λειτουργία με προνόμια υψηλότερου επιπέδου, απ' ότι πραγματικά χρειάζεται. Αυτό μπορεί να απενεργοποιήσει τους κανονικούς ελέγχους ασφάλειας που θα μπορούσε να εκτελέσει το λειτουργικό σύστημα ή το περιβάλλον του λογισμικού.

2.2.12 CWE-352. Cross-Site Request Forgery (CSRF)

Όταν ένας διακομιστής web έχει σχεδιαστεί να λαμβάνει αίτησεις από έναν πελάτη, χωρίς κανένα μηχανισμό για την επαλήθευση ότι έχουν σταλεί εσκεμμένα, τότε μπορεί ένας επιτιθέμενος να ξεγελάσει έναν πελάτη σε μια ακούσια αίτηση προς τον διακομιστή

web, η οποία θα αντιμετωπιστεί ως μια αυθεντική αίτηση. Αυτό μπορεί να γίνει μέσω ενός URL, το φόρτωμα μιας εικόνας, ενός XMLHttpRequest κλπ., και μπορεί να οδηγήσει σε έκθεση δεδομένων ή ακούσια εκτέλεση κώδικα.

Παράδειγμα:
Έστω ότι ο χρήστης Αλίκη θέλει να μεταφέρει 100$ στον Βασίλη, χρησιμοποιώντας το bank.com. Η Http αίτηση που θα στείλει ο περιηγητής της αλίκης θα έχει τη μορφή:

```
GET http://bank.com/transfer.do?
acct=BOB&amount=100 HTTP/1.1
```

Η Μαρία σκέφτεται ότι μπορεί να το εκμεταλλευτεί αυτό και κατασκευάζει το ακόλουθο URL, ώστε να μεταφέρει χρήματα στο δικό της λογαριασμό:

```
http://bank.com/transfer.do?
acct=MARIA&amount=100000
```

Τώρα αρκεί να πείσει την Αλίκη να στείλει αυτήν την αίτηση. Θα της στείλει ένα μήνυμα, αλλά για να μην εισάγει τον σύνδεσμο και δει η Αλίκη τη μεταφορά αφού τον πατήσει, εισάγει στο μήνυμά της μια HTML εικόνα ενός pixel:

```
<img src="http://bank.com/transfer.do?
acct=MARIA&amount=100000" width="1" height="1"
border="0" />
```

Όταν το λογισμικό ανάγνωσης του e-mail κάνει αυτήν την αίτηση για να φορτώσει την εικόνα, θα γίνει η μεταφορά στο bank.com και η Αλίκη δεν θα δει τίποτα.

2.2.13 CWE-22. Ανεπαρκής έλεγχος πρόσβασης σε κατάλογο (Path Traversal)

Το λογισμικό χρησιμοποιεί εξωτερική είσοδο για να δημιουργήσει ένα μονοπάτι (path) στο δίσκο, για την πρόσβαση ενός αρχείου ή καταλόγου. Αυτό μπορεί να βρίσκεται σε έναν περιορισμένο

κατάλογο, αλλά το λογισμικό να μην ελέγχει την είσοδο για ειδικούς χαρακτήρες, οπότε ένας επιτιθέμενος μπορεί να αποκτήσει πρόσβαση έξω από τον περιορισμένο κατάλογο.

Παράδειγμα:
Ο παρακάτω κώδικας, παίρνει σαν είσοδο ένα μονοπάτι σε κάποιο αρχείο, ελέγχει αν αυτό είναι κάτω από τον κατάλογο safe_dir και αν ναι, το σβήνει.

```
String path = getInputPath();
if (path.startsWith("/safe_dir/"))
{
      File f = new File(path);
      f.delete()
}
```

Ένας επιτιθέμενος όμως θα μπορούσε να δώσει ως είσοδο το παρακάτω:

```
/safe_dir/../important.dat
```

Ο κώδικας θα επιτρέψει τη διαγραφή, γιατί ελέγχει την αρχή του μονοπατιού, αλλά το αποτέλεσμα θα είναι να διαγραφεί το αρχείο important.dat, το οποίο βρίσκεται έναν κατάλογο πάνω από το safe_dir.

2.2.14 CWE-494. Κατέβασμα κώδικα χωρίς έλεγχο ακεραιότητας

Το λογισμικό κατεβάζει έναν απομακρυσμένο πηγαίο ή εκτελέσιμο κώδικα και τον εκτελεί χωρίς να ελέγξει την αυθεντικότητα και ακεραιότητά του. Ένας επιτιθέμενος θα μπορούσε να εκτελέσει κακόβουλο κώδικα αν αποκτήσει τον έλεγχο της πηγής ή αλλοιώσει τον κώδικα κατά τη μεταφορά του.

2.2.15 CWE-863. Εσφαλμένη εξουσιοδότηση

Το λογισμικό κάνει έλεγχο εξουσιοδότησης όταν ένας χρήστης ζητάει πρόσβαση σε έναν πόρο, αλλά δεν τον κάνει σωστά. Αυτό επιτρέπει σε έναν επιτιθέμενο να προσπεράσει τον έλεγχο.

Παράδειγμα:
Στον παρακάτω κώδικα PHP, επιτρέπεται η πρόσβαση στο ιατρικό ιστορικό (DisplayMedicalHistory), μόνο αν ο χρήστης έχει αυθεντικοποιηθεί πρώτα και κατέχει το ρόλο Reader:

```
$role = $_COOKIES['role'];
if (!$role) {
$role = getRole('user');
      if ($role) {
           // save the cookie to send out in
future               responses
           setcookie("role",     $role,     time()
+60*60*2);
      } else {
           ShowLoginScreen();
           die("\n");
      }
}
if ($role == 'Reader') {
      DisplayMedicalHistory($_POST['patient_ID']);
} else {
      die("You are not Authorized to view this
           record\n");
}
```

Το πρόβλημα είναι ότι διαβάζει το ρόλο από το cookie του αυθεντικοποιημένου χρήστη. Ένας κακόβουλος χρήστης όμως, μπορεί να θέσει εύκολα την τιμή Reader στο cookie, οπότε να του επιτραπεί η πρόσβαση.

2.2.16 CWE-829. Προσθήκη λειτουργικότητας από μη έμπιστη πηγή

Όταν συμπεριλαμβάνεται λειτουργικότητα από τρίτους, όπως ένα web widget, μια βιβλιοθήκη ή άλλη πηγή λειτουργικότητας, το λογισμικό πρέπει να εμπιστεύεται αυτή τη λειτουργικότητα. Χωρίς μηχανισμούς προστασίας, η λειτουργικότητα μπορεί να είναι κακόβουλη. Αυτό μπορεί να συμβεί είτε από μη έμπιστη πηγή ή με τροποποίηση κατά τη μεταφορά. Επίσης, η λειτουργικότητα, μπορεί να περιέχει τις δικές τις ευπάθειες ή να παρέχει πρόσβαση σε πρόσθετες λειτουργίες ή πληροφορίες κατάστασης, οι οποίες θα έπρεπε να είναι ιδιωτικές.

2.2.17 CWE-732. Εσφαλμένη ανάθεση δικαιωμάτων σε κρίσιμο πόρο

Όταν ορίζονται τα δικαιώματα σε έναν πόρο, ώστε να παρέχεται πρόσβαση σε περισσότερους χρήστες απ' ότι θα έπρεπε, κινδυνεύει η εμπιστευτικότητα και η ακεραιότητα αυτού του πόρου.

Παράδειγμα:
Ο παρακάτω κώδικας C, θέτει το umask ίσο με μηδέν και μετά γράφει το αρχείο hello.out στο δίσκο.

```
#define OUTFILE "hello.out"

umask(0);
FILE *out;
/* Ignore CWE-59 (link following) for brevity */
out = fopen(OUTFILE, "w");
if (out) {
      fprintf(out, "hello world!\n");
      fclose(out);
}
```

Αν τρέξουμε την εντολή "ls -l" στο UNIX, το αποτέλεσμα θα είναι κάπως έτσι:

```
-rw-rw-rw- 1 username 13 Nov 24 17:58 hello.out
```

Δηλαδή, όλοι οι χρήστες του συστήματος έχουν δικαιώματα ανάγνωσης και εγγραφής στο αρχείο hello.out.

2.2.18 CWE-676. Χρήση εν δυνάμει επικίνδυνης συνάρτησης

Το πρόγραμμα καλεί μια επικίνδυνη συνάρτηση, η οποία θα μπορούσε να εισάγει μια ευπάθεια αν χρησιμοποιηθεί εσφαλμένα, αλλά μπορεί επίσης να χρησιμοποιηθεί με ασφάλεια.

Παράδειγμα:
Ο παρακάτω κώδικας C, χρησιμοποιεί την επικίνδυνη strcpy χωρίς να ελέγχει το μέγεθος της παραμέτρου string. Αν ένας επιτιθέμενος μπορεί να ελέγξει την τιμή της, αυτό θα προκαλέσει μια υπερχείλιση στοίβας.

```
void manipulate_string(char * string){
      char buf[24];
      strcpy(buf, string);
      ...
}
```

2.2.19 CWE-327. Χρήση ανασφαλούς ή επικίνδυνου αλγορίθμου κρυπτογράφησης

Η χρήση ενός μη-τυπικού αλγορίθμου είναι επικίνδυνη, διότι ένας αποφασισμένος εισβολέας μπορεί να είναι σε θέση να σπάσει τον αλγόριθμο και να απειλήσει την εμπιστευτικότητα των δεδομένων. Επίσης μπορεί να υπάρχουν γνωστές τεχνικές σπασίματος ενός τυπικού ή μη αλγορίθμου.

2.2.20 CWE-131. Εσφαλμένος υπολογισμός μεγέθους προσωρινής μνήμης

Το λογισμικό δεν υπολογίζει σωστά το μέγεθος της προσωρινής μνήμης και αυτό μπορεί να οδηγήσει σε μια ευπάθεια υπερχείλισης στοίβας.

Παράδειγμα:
Ο παρακάτω κώδικας C, δεσμεύει έναν πίνακα για εικόνες:

```
img_t table_ptr; /*struct containing img data, 10kB
each*/
int num_imgs;
...
num_imgs = get_num_imgs();
table_ptr = (img_t*)malloc(sizeof(img_t)*num_imgs);
```

Ο κώδικας σκοπεύει να υπολογίσει σωστά το num_imgs, αλλά όσο αυτό μεγαλώνει, κάποια στιγμή μπορεί να υπερχειλίσει. Οπότε τελικά, θα δεσμευτεί ένας πολύ μικρός πίνακας και αυτό θα οδηγήσει στην ευπάθεια.

2.2.21 CWE-307. Ελλιπής απαγόρευση απεριόριστων προσπαθειών αυθεντικοποίησης

Το λογισμικό δεν υλοποιεί επαρκή μέτρα ώστε να αποτρέψει πολλές εσφαλμένες προσπάθειες αυθεντικοποίησης σε μικρό χρονικό διάστημα και αυτό μπορεί να επιτρέψει επιθέσεις brute-force.

Παράδειγμα:
Ο παρακάτω κώδικας Java προσπαθεί να αυθεντικοποιήσει έναν χρήστη χωρίς κανένα περιορισμό.

```
String username = request.getParameter("username");
String password = request.getParameter("password");

int authResult = authenticateUser(username, password);
```

Αν αυτός ο κώδικας καλείται για παράδειγμα στη μέθοδο doPost() ενός servlet, ένας επιτιθέμενος θα μπορεί να εκτελέσει απεριόριστες προσπάθειες από το διαδίκτυο.

2.2.22 CWE-601. Ανακατεύθυνση σε μη έμπιστο ιστότοπο (Open Redirect)

Μια Http παράμετρος μπορεί να περιέχει ένα URL και να προκαλεί μια ανακατεύθυνση της εφαρμογής web σε αυτό το URL. Αν ένας επιτιθέμενος αλλάξει αυτήν την παράμετρο, μπορεί να ανακατευθύνει έναν χρήστη σε ένα κακόβουλο ιστότοπο, για να κλέψει τα διαπιστευτήρια (phising). Επειδή το τμήμα του διακομιστή στο κακόβουλο URL θα είναι ίδιο με το έμπιστο ιστότοπο, η επίθεση αυτή έχει περισσότερες πιθανότητες να επιτύχει.

Παράδειγμα:
Ο παρακάτω κώδικας PHP ανακατευθύνει τον περιηγητή στην τιμή της μεταβλητής url:

```
$redirect_url = $_GET['url'];
header("Location: " . $redirect_url);
```

Ένας επιτιθέμενος που θέλει να ανακατευθύνει τον χρήστη στο κακόβουλο malicious.example.com μπορεί να δημιουργήσει αυτό το URL:

```
http://example.com/example.php?
url=http://malicious.example.com
```

2.2.23 CWE- 134. Μη ελεγχόμενη συμβολοσειρά μορφοποίησης

Το πρόγραμμα χρησιμοποιεί μια εξωτερική είσοδο ως παράμετρο μορφοποίησης σε μια συνάρτηση σαν την printf της γλώσσας C. Αυτό μπορεί να οδηγήσει σε ευπάθεια υπερχείλισης στοίβας ή προβλήματα παρουσίασης δεδομένων.

2.2.24 CWE-190. Υπερχείλιση ακεραίου

Μια υπερχείλιση ακεραίου συμβαίνει όταν μια ακέραια τιμή αυξάνεται πάνω από το όριο της αντίστοιχης αναπαράστασης.

Όταν αυτό συμβαίνει η τιμή μπορεί να αναδιπλωθεί και τελικά να αποθηκευτεί μια αρνητική ή πολύ μικρή θετική τιμή. Αν και αυτή η συμπεριφορά μπορεί να έχει προβλεφθεί, μπορεί επίσης να οδηγήσει σε προβλήματα ασφάλειας αν είναι απρόβλεπτη. Το σφάλμα αυτό είναι κρισιμότερο αν η υπερχείλιση μπορεί να προκληθεί από είσοδο του χρήστη και αν στην τιμή του ακεραίου βασίζεται ο έλεγχος ενός βρόχου, μια απόφαση ασφάλειας ή το μέγεθος μιας προσωρινής μνήμης (CWE-131).

2.2.25 CWE-759. Χρήση συνάρτησης κατακερματισμού χωρίς εισαγωγή τυχαίας συμβολοσειράς (salt)

Το λογισμικό χρησιμοποιεί μια συνάρτηση κατακερματισμού σε μια είσοδο που δεν πρέπει να είναι αναστρέψιμη, όπως έναν κωδικό πρόσβασης, αλλά δεν την συνενώνει με μια τυχαία συμβολοσειρά. Αυτό διευκολύνει έναν επιτιθέμενο να προϋπολογίσει τις τιμές κατακερματισμού, χρησιμοποιώντας μια τεχνική βασισμένη σε λεξικά, όπως τα rainbow tables.

Παράδειγμα:
Ο παρακάτω κώδικας Java ελέγχει το αποτέλεσμα της συνάρτησης κατακερματισμού με την αποθηκευμένη τιμή, η οποία προέρχεται μόνο από τον κωδικό του χρήστη:

```
String plainText = new String(plainTextIn);
MessageDigest                encer                =
MessageDigest.getInstance("SHA");
encer.update(plainTextIn);
byte[] digest = password.digest();
//Login if hash matches stored hash
if (equal(digest,secret_password())) {
      login_user();
}
```

2.3 Βέλτιστες πρακτικές για την αποφυγή σφαλμάτων ασφάλειας

Οι οργανισμοί CWE/SANS εκτός από τη λίστα των κρισιμότερων σφαλμάτων, έχει επίσης δημοσιεύσει εννέα βέλτιστες πρακτικές

[12] για την αποφυγή των top 25 αλλά και ως γενικές οδηγίες για την προστασία του λογισμικού. Ακολουθεί μια σύντομη περιγραφή τους.

2.3.1 Καθιέρωσε και συντήρησε ελέγχους σε όλες τις εισόδους (M1)

Η ανεπαρκής επικύρωση των εισόδων είναι ο υπ 'αριθμόν ένα δολοφόνος του υγιούς λογισμικού. Είναι επικίνδυνο να μην υπάρχει έλεγχος ότι η είσοδος είναι σύμφωνη με τις προσδοκίες του προγραμματιστή. Για παράδειγμα, ένα αναγνωριστικό που αναμένεται να είναι αριθμητικό δεν πρέπει να περιέχει ποτέ γράμματα. Ούτε θα πρέπει η τιμή ενός καινούργιου αυτοκινήτου να επιτρέπεται να είναι ένα δολάριο, ούτε καν στη σημερινή οικονομία. Φυσικά, οι εφαρμογές έχουν συχνά πιο πολύπλοκες απαιτήσεις επικύρωσης από αυτά τα απλά παραδείγματα. Η λανθασμένη επικύρωση των εισόδων μπορεί να οδηγήσει σε ευπάθειες, όταν οι επιτιθέμενοι μπορούν να τροποποιήσουν τις εισόδους με απροσδόκητους τρόπους. Πολλές από τις πιο κοινές τρέχουσες ευπάθειες μπορούν να εξαλειφθούν, ή τουλάχιστον να μειωθούν, με σωστή επικύρωση των εισόδων.

Χρησιμοποιήστε ένα τυπικό μηχανισμό επικύρωσης των εισόδων για:

- Το μήκος
- Τον τύπο της εισόδου
- Τη σύνταξη
- Την έλλειψη ή τις επιπλέον εισόδους
- Τη συνέπεια μεταξύ συναφών πεδίων
- Τους επιχειρηματικούς κανόνες

Ως παράδειγμα της λογικής των επιχειρηματικών κανόνων, το "σκάφος" μπορεί να είναι συντακτικά έγκυρο επειδή περιέχει μόνο αλφαριθμητικούς χαρακτήρες, αλλά δεν είναι έγκυρο, αν περιμένετε ονόματα χρωμάτων όπως το "κόκκινο" ή "μπλε".

Όπου είναι δυνατόν, χρησιμοποιήστε αυστηρές λευκές λίστες που περιορίζουν το σύνολο χαρακτήρων, με βάση την αναμενόμενη τιμή της παραμέτρου στην αίτηση. Αυτό μπορεί να έχει έμμεσα οφέλη, όπως τη μείωση ή την εξάλειψη των αδυναμιών που μπορεί να υπάρχουν σε άλλα μέρη του προϊόντος.

Δεν πρέπει να δέχεστε καμία είσοδο που παραβιάζει τους κανόνες αυτούς, ή να τις μετατρέπετε σε ασφαλείς τιμές.

Κατανοήστε όλες τις πιθανές περιοχές όπου μη έμπιστες είσοδοι μπορούν να εισέλθουν στο λογισμικό σας: παραμέτρους, cookies, οτιδήποτε διαβάσετε από το δίκτυο, μεταβλητές περιβάλλοντος, αντίστροφες αναζητήσεις DNS, τα αποτελέσματα ερωτημάτων, κεφαλίδες αίτησης, στοιχεία της διεύθυνσης URL, e-mail, αρχεία, βάσεις δεδομένων και οποιαδήποτε εξωτερικά συστήματα παρέχουν δεδομένα στην εφαρμογή. Να θυμάστε ότι η εισαγωγή μπορεί να επιτευχθεί έμμεσα μέσω κλήσεων API.

Να είστε προσεκτικοί ώστε να αποκωδικοποιήσετε σωστά τις εισόδους και να τις μετατρέψετε στην εσωτερική αναπαράστασή σας, πριν από την εκτέλεση της επικύρωσης.

2.3.2 Καθιέρωσε και συντήρησε ελέγχους σε όλες τις εξόδους (M2)

Οι υπολογιστές έχουν μια παράξενη συνήθεια να κάνουν ότι τους λέτε, όχι ότι εννοείτε. Η ανεπαρκής κωδικοποίηση της εξόδου είναι ο συχνά αγνοούμενος αδελφός της ακατάλληλης επικύρωσης των εισόδων, αλλά είναι στη ρίζα των περισσότερων επιθέσεων ένεσης, οι οποίες κάνουν θραύση αυτές τις μέρες. Ένας εισβολέας μπορεί να τροποποιήσει τις εντολές που σκοπεύετε να στείλετε σε άλλα συστατικά και αυτό ενδέχεται να οδηγήσει σε μια πλήρη κατάληψη της εφαρμογής σας. Αυτό μετατρέπει το "κάνε ότι εννοώ" σε "κάνε ότι ο επιτιθέμενος λέει". Όταν το πρόγραμμά σας παράγει εξόδους σε άλλα συστατικά, με τη μορφή δομημένων μηνυμάτων όπως ερωτήματα ή αιτήσεις, πρέπει να διαχωρίσετε τις πληροφορίες ελέγχου και τα μετα-δεδομένα από τα πραγματικά δεδομένα. Αυτό είναι εύκολο να ξεχαστεί, γιατί πολλά παραδείγματα μεταφέρουν δεδομένα και εντολές ομαδοποιημένα στο ίδιο ρεύμα, με λίγους

μόνο ειδικούς χαρακτήρες επιβολής των ορίων. Ένα παράδειγμα είναι το Web 2.0 και άλλα πλαίσια που λειτουργούν θολώνοντας αυτές τις γραμμές. Αυτό τα εκθέτει περαιτέρω.

Εάν υπάρχουν, χρησιμοποιήστε δομημένους μηχανισμούς που επιβάλλουν αυτόματα το διαχωρισμό μεταξύ δεδομένων και κώδικα. Οι μηχανισμοί αυτοί μπορεί να είναι σε θέση να παράσχουν τις σχετικές κωδικοποιήσεις και επικυρώσεις αυτόματα, αντί να βασίζονται στον προγραμματιστή. Για παράδειγμα, οι stored procedures μπορούν να επιβάλουν τη δομή των ερωτημάτων στη βάση δεδομένων και να μειώσει την πιθανότητα του SQL injection.

Καταλάβετε το πλαίσιο στο οποίο τα δεδομένα σας θα χρησιμοποιηθούν και την κωδικοποίηση που θα πρέπει να αναμένεται. Αυτό είναι ιδιαίτερα σημαντικό κατά τη μετάδοση δεδομένων μεταξύ των διαφόρων τμημάτων, ή όταν παράγονται έξοδοι που μπορούν να περιέχουν πολλαπλές κωδικοποιήσεις ταυτόχρονα, όπως ιστοσελίδες ή πολλαπλών τμημάτων μηνύματα. Μελετήστε όλα τα αναμενόμενα πρωτόκολλα επικοινωνίας και τις αναπαραστάσεις δεδομένων, ώστε να καθορίσετε τις απαιτούμενες στρατηγικές κωδικοποίησης.

2.3.3 Κλείδωσε το περιβάλλον (M3)

Είναι ατυχές γεγονός της ζωής: οι ευπάθειες συμβαίνουν, ακόμη και με τις καλύτερες προσπάθειές σας. Με τη χρήση της άμυνας σε βάθος, μπορείτε να περιορίσετε το εύρος και τη σοβαρότητα μιας επιτυχημένης επίθεσης σε μια απροσδόκητη αδυναμία, μερικές φορές ακόμη και τη μείωση των προβλημάτων σε μια περιστασιακή ενόχληση. Η βασική πτυχή της άμυνας σε βάθος είναι να αποφευχθεί η εξάρτηση από ένα μόνο χαρακτηριστικό, λύση, μετριασμό ή εμπόδιο (π.χ. firewall) για την παροχή ασφάλειας. Το κλείδωμα του περιβάλλοντος σας βοηθά να το επιτύχετε αυτό.

Εκτελέστε το λογισμικό σας σε ένα περιορισμένο περιβάλλον, χρησιμοποιώντας τις διαθέσιμες δυνατότητες ασφαλείας, για να περιορίσετε τι μπορεί να κάνει στο σύστημα που εκτελείται και στα παρακείμενα δίκτυα. Ακόμα και αν ένας εισβολέας καταφέρει να βρει και να εκμεταλλευτεί μια ευπάθεια, το κλείδωμα

περιβάλλοντος μπορεί να περιορίσει τι ο επιτιθέμενος μπορεί να κάνει, μειώνοντας τη συνολική σοβαρότητα του προβλήματος και να κάνει το έργο του εισβολέα πιο δύσκολο. Σε ορισμένες περιπτώσεις, το κλείδωμα μπορεί να αποτρέψει τελείως το πρόβλημα να μετατραπεί σε ευπάθεια.

Ορισμένες από τις πτυχές του κλειδώματος του περιβάλλοντος είναι :

- Εκτέλεση με προνόμια χρήστη. Εκτελέστε το λογισμικό σας με τα χαμηλότερα δυνατά προνόμια. Ακόμα κι αν το λογισμικό σας καταληφθεί, μέσω μιας αδυναμίας που δεν γνωρίζετε, θα καταστήσει πιο δύσκολο για έναν εισβολέα να προκαλέσει περαιτέρω ζημία. Ο Spider Man, το γνωστό κόμικ υπερήρωα, ζει με το σύνθημα «Με τη μεγάλη δύναμη έρχεται η μεγάλη ευθύνη". Το λογισμικό σας μπορεί να χρειαστεί ειδικά προνόμια για την εκτέλεση ορισμένων εργασιών, αλλά κρατώντας τα προνόμια περισσότερο από όσο χρειάζεται μπορεί προκύψει μεγάλο ρίσκο. Όταν τρέχει με επιπλέον προνόμια, η εφαρμογή σας έχει πρόσβαση σε πόρους που ο χρήστης της εφαρμογής δεν μπορεί άμεσα να φτάσει. Για παράδειγμα, μπορεί να εκκινήσετε εκ προθέσεως ένα ξεχωριστό πρόγραμμα και αυτό το πρόγραμμα να επιτρέπει στους χρήστες να ορίσουν το αρχείο που θα ανοίξει. Αυτό το χαρακτηριστικό παρουσιάζεται συχνά σε αναγνώστες ή συντάκτες βοήθειας. Ο χρήστης μπορεί να έχει πρόσβαση σε αρχεία χωρίς εξουσιοδότηση, μέσω του προγράμματος που ξεκίνησε, χάρη σε αυτά τα επιπλέον προνόμια. Αντίστοιχα μπορεί να γίνει εκτέλεση εντολών.

- Σκεφτείτε να απενεργοποιήσετε τα λεπτομερή μηνύματα λάθους, για τα μηνύματα που εμφανίζονται στους χρήστες χωρίς ειδικά προνόμια. Εάν χρησιμοποιείτε περιγραφικά μηνύματα λάθους, είναι πιθανό να αποκαλυφθούν μυστικά σε κάποιον εισβολέα που προσπαθεί να επιτεθεί στο λογισμικό σας. Τα μυστικά μπορούν να περιλαμβάνουν ένα ευρύ φάσμα πολύτιμων δεδομένων, όπως προσωπικών πληροφοριών ταυτοποίησης, διαπιστευτήρια αυθεντικοποίησης και παραμέτρους του διακομιστή. Μερικές φορές, μπορεί να

φαίνονται σαν ακίνδυνα μυστικά που βοηθούν τους χρήστες και τους διαχειριστές σας, όπως την πλήρη διαδρομή εγκατάστασης του λογισμικού σας. Ακόμα και αυτά τα μικρά μυστικά μπορούν να απλοποιήσουν σε μεγάλο βαθμό, μια πιο συντονισμένη επίθεση με μεγαλύτερες ανταμοιβές.

- Όπου είναι δυνατόν, αν πρέπει να χρησιμοποιήσετε βιβλιοθήκες τρίτων, επιλέξτε τις καλύτερες και αυτές με ένα αποδεδειγμένο ιστορικό υψηλής ασφάλειας.

- Μεταγλωττίστε τον κώδικά σας χρησιμοποιώντας επιλογές σχετικές με την ασφάλεια. Για παράδειγμα, ορισμένοι μεταγλωττιστές παρέχουν ενσωματωμένη προστασία έναντι στις υπερχειλίσεις στοίβας.

- Χρησιμοποιήστε χαρακτηριστικά του λειτουργικού συστήματος και του υλισμικού, όταν είναι διαθέσιμα, τα οποία περιορίζουν τον αντίκτυπο μιας επιτυχημένης επίθεσης. Για παράδειγμα, ορισμένες πλατφόρμες, λειτουργικά συστήματα, και βιβλιοθήκες, καθιστούν πιο δύσκολη την εκμετάλλευση των υπερχειλίσεων στοίβας. Κάποια λειτουργικά συστήματα, σας επιτρέπουν να καθορίσετε τις ποσοστώσεις για τη χρήση του δίσκου, της CPU , της μνήμης , των ανοιχτών αρχείων και άλλων.

- Χρησιμοποιήστε εικονικές μηχανές, sandboxes, φυλακές, ή παρόμοιες τεχνολογίες που περιορίζουν την πρόσβαση στο περιβάλλον. Ακόμα και αν ένας εισβολέας καταλάβει εντελώς το λογισμικό σας, η ζημιά στο υπόλοιπο περιβάλλον μπορεί να ελαχιστοποιηθεί.

- Εξετάστε την υιοθέτηση ασφαλών διαμορφώσεων χρησιμοποιώντας αξιολογήσεις ασφάλειας ή οδηγούς ασφαλούς παραμετροποίησης.

- Χρησιμοποιήστε σαρωτές ευπαθειών και εφαρμόζετε τακτικά τις διορθώσεις ασφάλειας (security patches), ώστε να εξασφαλίσετε ότι το σύστημά σας δεν έχει γνωστές αδυναμίες. Ακόμα κι αν η εφαρμογή σας δεν έχει κανένα

πρόβλημα, ένας εισβολέας μπορεί να τη θέσει σε κίνδυνο, μέσα από μια επίθεση στο ίδιο το σύστημα.

2.3.4 Υπόθεσε ότι τα εξωτερικά συστατικά μπορούν να τροποποιηθούν και ο κώδικάς σου μπορεί να διαβαστεί από οποιονδήποτε (M4)

Όσον αφορά την ασφάλεια, ο κώδικάς σας λειτουργεί σε ένα εχθρικό περιβάλλον. Οι επιτιθέμενοι μπορούν να αναλύσουν το λογισμικό σας - ακόμα κι αν έχει χρησιμοποιηθεί obfuscation - και να καταλάβουν τι συμβαίνει. Μπορούν να τροποποιήσουν εξωτερικά εξαρτήματα για να αγνοήσουν τους ελέγχους ασφάλειας. Μπορούν να δοκιμάσουν οποιοδήποτε είδος επίθεσης, ενάντια σε κάθε στοιχείο του συστήματος. Αν ο κώδικας διανέμεται στους καταναλωτές σας, μπορούν να τρέξουν το λογισμικό σας και να του επιτεθούν στο δικό τους περιβάλλον, χωρίς ποτέ να το μάθετε.

Πολλοί προγραμματιστές θεωρούν ότι ο μεταγλωττισμένος κώδικάς τους, ή το προσαρμοσμένο πρωτόκολλο τους, δεν μπορεί να το καταλάβει ένας επιτιθέμενος. Ωστόσο, οι σύγχρονες τεχνικές αντίστροφης μηχανικής είναι πολύ ώριμη και οι επιτιθέμενοι μπορούν να καταλάβουν την εσωτερική λειτουργία του λογισμικού σας γρήγορα, ακόμα και με ιδιωτικά πρωτόκολλα, έλλειψη πρόσβασης σε προδιαγραφές και χωρίς τον πηγαίο κώδικα. Όταν οι μηχανισμοί προστασίας του λογισμικού βασίζονται σε αυτήν τη μυστικότητα, οι εν λόγω μηχανισμοί μπορεί συχνά να ηττηθούν. Μπορείτε να δοκιμάσετε να κρυπτογραφήσετε τον ίδιο σας τον κώδικα, αλλά ακόμα κι έτσι, είναι μια απλή τακτική καθυστέρησης στα περισσότερα περιβάλλοντα.

Δεν μπορείτε επίσης, να εμπιστεύεστε έναν πελάτη για να κάνει ελέγχους ασφάλειας για λογαριασμό του διακομιστή σας, ακόμα κι αν γράψατε εσείς τον πελάτη. Να θυμάστε ότι κάτω από αυτό το φανταχτερό GUI, είναι απλός κώδικας. Οι επιτιθέμενοι μπορούν να χρησιμοποιήσουν τεχνικές αντίστροφης μηχανικής στον πελάτη σας και να γράψουν τους δικούς τους προσαρμοσμένους πελάτες, που αφήνουν έξω ορισμένα ενοχλητικά χαρακτηριστικά, όπως όλους αυτούς τους ενοχλητικούς ελέγχους ασφάλειας.

2.3.5 Χρησιμοποίησε αποδεκτές από τη βιομηχανία λειτουργίες ασφάλειας και μην ανακαλύπτεις τις δικές σου (M5)

Μπορεί να μπείτε στον πειρασμό να δημιουργήσετε το δικό σας αλγορίθμο ασφάλειας, αλλά σοβαρά, αυτό είναι δύσκολο να το κάνετε σωστά. Μπορεί να νομίζετε ότι έχετε δημιουργήσει έναν ολοκαίνουργιο αλγόριθμο που κανείς δεν θα τον καταλάβει, αλλά είναι πιο πιθανό ότι επανανακαλύψατε τον τροχό που πέφτει λίγο πριν την εκκίνηση της παρέλασης.

Αυτό ισχύει για χαρακτηριστικά, όπως η κρυπτογράφηση, η αυθεντικοποίηση, η εξουσιοδότηση, η τυχαιότητα, η διαχείριση συνόδου, τα αρχεία καταγραφών, και άλλα.

- Διερευνήστε ποιος από τους διαθέσιμους αλγορίθμους ασφάλειας είναι ισχυρότερος για την κρυπτογραφία, αυθεντικοποίηση και εξουσιοδότηση και χρησιμοποιήστε τον από προεπιλογή. Χρησιμοποιήστε αποδεκτούς από τη βιομηχανία αλγορίθμους που σήμερα θεωρούνται ισχυροί από εμπειρογνώμονες στον τομέα και επιλέξτε καλά δοκιμασμένες υλοποιήσεις. Μείνετε μακριά από ιδιόκτητους και μυστικούς αλγορίθμους. Σημειώστε το μήκος των κλειδιών συχνά δεν αποτελεί αξιόπιστη ένδειξη. Σε περίπτωση αμφιβολίας, ζητήστε συμβουλές.

- Όταν χρησιμοποιείτε εγκεκριμένες από τη βιομηχανία τεχνικές, θα πρέπει να τις χρησιμοποιείτε σωστά. Μην κόβετε τις γωνίες παρακάμπτοντας βήματα με υπολογιστικό κόστος. Τα βήματα αυτά είναι συχνά απαραίτητα για την πρόληψη κοινών επιθέσεων.

2.3.6 Χρησιμοποίησε βιβλιοθήκες και πλατφόρμες που διευκολύνουν την αποφυγή αδυναμιών (GP1)

Τα τελευταία χρόνια, έχουν αναπτυχθεί διάφορα πλαίσια και βιβλιοθήκες ώστε να είναι ευκολότερο για σας να εισαγάγετε χαρακτηριστικά ασφάλειας και να αποφευχθεί η κατά λάθος εισαγωγή αδυναμιών. Η χρήση των στερεών, σχετικών με την

ασφάλεια βιβλιοθηκών μπορεί να σώσει πόρους ανάπτυξης και να καθιερώσει ένα καλά κατανοητό επίπεδο ασφάλειας. Αν ένα πρόβλημα ασφάλειας βρεθεί, η επιδιόρθωση θα μπορούσε να είναι τοπική στη βιβλιοθήκη, αντί να απαιτούνται μεγάλες αλλαγές σε όλο τον κώδικα.

Τελικά, μπορείτε να επιλέξετε να αναπτύξετε τις δικές σας προσαρμοσμένες πλατφόρμες ή βιβλιοθήκες. Αλλά εν τω μεταξύ, υπάρχουν ορισμένοι πόροι που διατίθενται για να σας δώσουν ένα προβάδισμα. Οι επιλογές κυμαίνονται από βιβλιοθήκες ασφαλούς χειρισμού συμβολοσειρών στη C/C++, παραμετροποιήσιμους μηχανισμούς ερωτημάτων, όπως εκείνες που είναι διαθέσιμες για την SQL, πλαίσια επικύρωσης εισόδων, όπως το Struts, συστήματα API, όπως το ESAPI, και ούτω καθεξής.

Αν και συχνά υποστηρίζεται ότι θα μπορούσατε να επιλέξετε μια ασφαλέστερη γλώσσα, αυτό σπάνια είναι εφικτό στη μέση ενός εν εξελίξει έργου. Επιπλέον, κάθε γλώσσα έχει χαρακτηριστικά που θα μπορούσαν να χρησιμοποιηθούν κατά τρόπο που να δημιουργούν αδυναμίες. Τέλος, πολύ λίγες αδυναμίες αφορούν συγκεκριμένες γλώσσες, δηλαδή, οι περισσότερες αδυναμίες έχουν εφαρμογή σε ένα ευρύ φάσμα γλωσσών.

2.3.7 Ενσωμάτωσε την ασφάλεια στον κύκλο ζωής του λογισμικού (GP2)

Το Top 25 είναι μόνο το πρώτο βήμα στο δρόμο για πιο ασφαλές λογισμικό. Οι αδυναμίες και τα τρωτά σημεία μπορεί να αποφευχθούν μόνο όταν ασφαλείς πρακτικές ανάπτυξης ενσωματωθούν σε όλες τις φάσεις του κύκλου ζωής του λογισμικού, συμπεριλαμβανομένων (αλλά όχι περιοριστικά) των απαιτήσεων και των προδιαγραφών, του σχεδιασμού και της αρχιτεκτονικής, της υλοποίησης, την κατασκευής, δοκιμής, εγκατάστασης, λειτουργίας και συντήρησης. Η δημιουργία ομάδας ασφάλειας λογισμικού, μπορεί να είναι το πιο κρίσιμο βήμα.

Έργα όπως τα BSIMM, SAFEcode και OpenSAMM μπορούν να σας βοηθήσουν να καταλάβετε τι κάνουν οι κορυφαίες οργανώσεις για την ασφαλή ανάπτυξη.

2.3.8 Χρησιμοποίησε ένα ευρύ φάσμα μεθόδων για να βρεις και να αποτρέψεις τις αδυναμίες (GP3)

Δεν υπάρχει ένα μοναδικό εργαλείο, τεχνική, ή διαδικασία που θα εγγυηθεί ότι το λογισμικό σας είναι ασφαλές. Κάθε προσέγγιση έχει τα δικά της πλεονεκτήματα και τους περιορισμούς της, έτσι ένας κατάλληλος συνδυασμός θα βελτιώσει την ασφάλεια περισσότερο από οποιαδήποτε τεχνική.

Οι προσεγγίσεις περιλαμβάνουν, αλλά δεν περιορίζονται σε :

- Αυτοματοποιημένη στατική ανάλυση κώδικα: είτε πρόκειται για τον πηγαίο κώδικα ή δυαδικό κώδικα. Η σύγχρονη αυτοματοποιημένη στατική ανάλυση κώδικα μπορεί να παρέχει εκτεταμένη κάλυψη κώδικα, με υψηλά ποσοστά επιτυχίας για ορισμένα είδη αδυναμιών , ειδικά για τα λάθη υλοποίησης. Ωστόσο, σε ορισμένα πλαίσια, οι τεχνικές αυτές μπορούν να αναφέρουν ένα μεγάλο αριθμό θεμάτων και ένας προγραμματιστής μπορεί να μην επιθυμεί να τα αντιμετωπίσει. Έχουν επίσης δυσκολία με διάφορα προβλήματα επιπέδου σχεδιασμού, τα οποία συχνά απαιτούν την χειροκίνητη ανάλυση.

- Χειροκίνητη στατική ανάλυση κώδικα: Μπορεί να είναι χρήσιμη για την εύρεση λεπτών λαθών και ατελειών του σχεδιασμού και για να δοθεί μια συνολική αξιολόγηση των πρακτικών ανάπτυξης. Είναι επίσης χρήσιμη για την ανίχνευση ελαττωμάτων στην επιχειρηματική λογική. Ωστόσο, μπορεί να είναι ακριβή στη διεξαγωγή, η κάλυψη κώδικα είναι μια πρόκληση και μπορεί να μην καλύψει όλους τους φορείς επιθέσεων.

- Αυτοματοποιημένη δυναμική ανάλυση κώδικα: Fuzzers και σαρωτές. Αυτά μπορούν να βρουν κάποια θέματα που είναι δύσκολο να ανιχνευθούν με στατική ανάλυση, όπως τα περιβαλλοντικά προβλήματα και να δημιουργήσουν ένα μεγάλο αριθμό εισόδων ή αλληλεπιδράσεων και να βρουν

λεπτές ρωγμές που μπορεί να χαθούν από τη στατική ανάλυση. Ωστόσο, η κάλυψη του κώδικα είναι ένα πρόβλημα.

- Χειροκίνητη δυναμική ανάλυση κώδικα: Αυτή μπορεί να είναι αποτελεσματική για τον εντοπισμό ατελειών στην επιχειρηματική λογική, για γρήγορη εύρεση ορισμένων τύπων θεμάτων που δεν είναι εύκολο να αυτοματοποιηθούν και για την κατανόηση της συνολικής ασφάλειας του λογισμικού.

- Μοντελοποίηση απειλών: Αυτό είναι χρήσιμο για την εύρεση προβλημάτων πριν την ανάπτυξη του κώδικα. Ενώ οι τεχνικές αυτές είναι ακόμα υπό ανάπτυξη, σήμερα η προσέγγιση αυτή είναι δύσκολο να διδαχθεί, τα διαθέσιμα εργαλεία έχουν περιορισμένη ισχύ και ωφελείται σημαντικά από τη συμμετοχή εμπειρογνωμόνων.

- Τείχη προστασίας εφαρμογών και πλαίσια εξωτερικής παρακολούθησης/ελέγχου (web app firewalls, proxies, IPS, Struts κ.λπ.): Η χρήση αυτών των μηχανισμών μπορεί να είναι κατάλληλη για την προστασία του λογισμικού που είναι γνωστό ότι περιέχει αδυναμίες, αλλά δεν μπορεί να τροποποιηθεί. Μπορεί επίσης να είναι χρήσιμη για την πρόληψη επιθέσεων εναντίον λανθανουσών αδυναμιών που δεν έχουν ακόμη ανακαλυφθεί. Ωστόσο, η εφαρμογή τους θα μπορούσε να βλάψει τη νόμιμη λειτουργικότητα, μπορεί να χρειαστούν προσαρμογή και δεν μπορούν να ανιχνεύσουν ή να αποτρέψουν όλες τις επιθέσεις.

- Εκπαίδευση και κατάρτιση: Η κατάλληλη εκπαίδευση θα βοηθήσει τους προγραμματιστές να αποφεύγουν την εισαγωγή σφαλμάτων σε κώδικα, μειώνοντας το κόστος διόρθωσης. Ωστόσο, η ίδια η κατάρτιση μπορεί να είναι ακριβή, να αντιμετωπίσει προκλήσεις για την έγκριση και απαιτεί ενημέρωση, όπως οι μέθοδοι λογισμικού και ανίχνευσης εξελίσσονται.

- Επισκόπηση αρχιτεκτονικής και σχεδιασμού: Αυτά μπορεί να είναι σημαντικά για τον εντοπισμό προβλημάτων που θα ήταν πάρα πολύ δύσκολο, χρονοβόρο, ή ακριβό να καθοριστούν

αφού το προϊόν έχει αναπτυχθεί. Μπορεί να απαιτήσουν βοήθεια από εμπειρογνώμονες.

- Πρότυπα κωδικοποίησης: Τα κατάλληλα πρότυπα κωδικοποίησης μπορεί να μειώσουν τον αριθμό των αδυναμιών που εισάγονται στον κώδικα, το οποίο μπορεί να μειώσει το κόστος συντήρησης. Ωστόσο, αυτά είναι δύσκολο να ενταχθούν σε υφιστάμενα έργα , ιδιαίτερα στα μεγάλα.

2.3.9 Επίτρεψε στους κλειδωμένους πελάτες να αλληλεπιδράσουν με το λογισμικό σου (GP4)

Εάν ένας χρήστης έχει έναν κλειδωμένο πελάτη, που δεν υποστηρίζει όλες τις λειτουργίες που χρησιμοποιούνται από την εφαρμογή σας, σκεφτείτε να επιτρέψετε κάποιου είδους πρόσβαση, αντί της πλήρους απαγόρευσης πρόσβασης, η οποία μπορεί να αναγκάσει τον χρήστη να υποβαθμίσει την ασφάλεια του πελάτη του. Αυτό μπορεί να βοηθήσει στην προστασία των χρηστών και να επιβραδύνει την εξάπλωση του κακόβουλου λογισμικού. Για παράδειγμα, σε ένα διαδικτυακό περιβάλλον, ορισμένοι χρήστες μπορεί να έχουν απενεργοποιήσει τη Javascript ή τα plug-ins και απαιτώντας αυτά τα χαρακτηριστικά τους εκθέτετε σε επιθέσεις από άλλους δικτυακούς τόπους.

ΚΕΦΑΛΑΙΟ 3

3 ΣΤΑΤΙΚΗ ΑΝΑΛΥΣΗ

3.1 Εισαγωγή

Όπως χαρακτηριστικά αναφέρουν οι Chees & West [13], οποιοδήποτε εργαλείο αναλύει κώδικα χωρίς να τον εκτελέσει, εφαρμόζει στατική ανάλυση. Για το σκοπό του εντοπισμού των προβλημάτων ασφάλειας, η ποικιλία των στατικών εργαλείων ανάλυσης που μας ενδιαφέρει, είναι αυτά που συμπεριφέρονται λίγο-πολύ σαν ένας ορθογράφος. Εμποδίζουν καλά κατανοητές ποικιλίες λαθών, από το να περάσουν απαρατήρητες. Ακόμη και αυτοί που γνωρίζουν ορθογραφία πολύ καλά, χρησιμοποιούν έναν ορθογράφο διότι πάντοτε, ορθογραφικά λάθη θα εμφανιστούν σε ένα κείμενο. Ο ορθογράφος όμως δεν θα πιάσει κάθε λάθος: Εάν πληκτρολογήσετε *λύπη* και εννοείτε *λείπει*, ο ορθογράφος δεν θα βοηθήσει. Τα εργαλεία στατικής ανάλυσης λειτουργούν με παρόμοιο τρόπο. Ένα καθαρό τρέξιμο δεν εγγυάται ότι ο κώδικας σας είναι τέλειος. Δείχνει απλώς ότι είναι απαλλαγμένος από ορισμένα είδη κοινών προβλημάτων. Οι περισσότεροι έμπειροι και επαγγελματίες συγγραφείς βρίσκουν ότι ο ορθογράφος είναι ένα χρήσιμο εργαλείο. Οι κακοί συγγραφείς βρίσκουν χρήσιμο τον ορθογράφο επίσης, αλλά το εργαλείο δεν τους μετατρέπει σε εξαιρετικούς συγγραφείς. Το ίδιο ισχύει και για τη στατική ανάλυση: Οι καλοί προγραμματιστές μπορούν να αξιοποιήσουν τα εργαλεία στατικής ανάλυσης και να έχουν ένα εξαιρετικό αποτέλεσμα, αλλά οι κακοί προγραμματιστές θα εξακολουθούν να παράγουν κακά προγράμματα, ανεξάρτητα από τα εργαλεία που χρησιμοποιούν.

3.1.1 Χρήσεις της στατικής ανάλυσης

Αν και η συγκεκριμένη εργασία επικεντρώνεται στη χρήση της στατικής ανάλυσης στην εύρεση σφαλμάτων ασφάλειας, αυτή η μέθοδος μπορεί να λύσει και άλλων ειδών προβλήματα. Μάλιστα η στατική ανάλυση χρησιμοποιείται από πολλούς χωρίς να το καταλαβαίνουν, κυρίως επειδή υπάρχουν πολλών ειδών εργαλεία, με διαφορετικούς στόχους το καθένα. Ενδεικτικά, η στατική ανάλυση κώδικα χρησιμοποιείται για:

Έλεγχο τύπου
Η πιο διαδεδομένη μορφή της στατικής ανάλυσης και αυτή με την οποία είναι εξοικειωμένοι οι περισσότεροι προγραμματιστές, είναι ο έλεγχος τύπου. Πολλοί προγραμματιστές δεν σκέφτονται πολύ τον έλεγχο τύπου. Έτσι κι αλλιώς, οι κανόνες του παιχνιδιού συνήθως ορίζονται από τη γλώσσα προγραμματισμού και επιβάλλονται από τον μεταγλωττιστή. Έτσι ο προγραμματιστής ασχολείται ελάχιστα με το πότε εφαρμόζεται η ανάλυση ή πώς αυτή λειτουργεί. Ωστόσο ο έλεγχος τύπου είναι στατική ανάλυση. Ο έλεγχος τύπου εξαλείφει ολόκληρες κατηγορίες λαθών προγραμματισμού. Για παράδειγμα, αποτρέπει προγραμματιστές από την τυχαία ανάθεση ακέραιων τιμών σε μια μεταβλητή αντικειμένου. Αλιεύοντας σφάλματα κατά τη μεταγλώττιση, ο έλεγχος τύπου αποτρέπει σφάλματα εκτέλεσης. Ο έλεγχος τύπου έχει περιορισμένη ικανότητά και πάσχει από ψευδώς θετικά και ψευδώς αρνητικά αποτελέσματα, ακριβώς όπως όλα τα άλλα είδη στατικής ανάλυσης.

Έλεγχος προγραμματιστικού στυλ
Τα εργαλεία που ελέγχουν το στυλ είναι επίσης εργαλεία στατικής ανάλυσης. Γενικά επιβάλουν ένα σύνολο πιο επιφανειακών κανόνων από τον έλεγχο τύπου. Επιβάλουν κανόνες σχετικά με τα κενά, την ονοματολογία, καταργημένες συναρτήσεις, τα σχόλια, τη δομή του προγράμματος και άλλα αντίστοιχα. Επειδή οι περισσότεροι προγραμματιστές είναι προσκολλημένοι στο προσωπικό τους στυλ, τα περισσότερα εργαλεία του είδους είναι πολύ ευέλικτα. Τα σφάλματα που αναφέρονται από αυτήν τη διαδικασία, συνήθως σχετίζονται με την αναγνωσιμότητα και τη συντηρησιμότητα του κώδικα αλλά δεν δείχνουν κάποιο σφάλμα που θα προκύψει κατά την

εκτέλεσή του. Με τον καιρό, αρκετοί μεταγλωττιστές έχουν υλοποιήσει προαιρετικό έλεγχο στυλ.

Κατανόηση προγράμματος

Τα εργαλεία κατανόησης προγράμματος, βοηθούν τους χρήστες να καταλάβουν μια μεγάλη βάση κώδικα. Τα ολοκληρωμένα περιβάλλοντα ανάπτυξης περιέχουν τουλάχιστον ένα εργαλείο κατανόησης προγράμματος. Απλά παραδείγματα τέτοιων λειτουργιών είναι οι “βρες όλες της κλήσεις αυτής της μεθόδου” και “βρες τη δήλωση αυτής της καθολικής μεταβλητής”. Πιο εξελιγμένες μέθοδοι ανάλυσης επιτρέπουν αυτόματο factoring, όπως τη μετονομασία μεταβλητών ή τη διάσπαση μιας συνάρτησης σε πολλές μικρότερες. Τα εργαλεία κατανόησης προγράμματος υψηλότερου επιπέδου, βοηθούν τον προγραμματιστή να σχηματήτισει μια ιδέα για το πως δουλεύει ένα πρόγραμμα. Μερικά προσπαθούν να εφαρμόσουν αντίστροφη μηχανική και να δώσουν μια γενικότερη εικόνα της αρχιτεκτονικής του λογισμικού. Αυτό είναι ιδιαίτερα χρήσιμο όταν οι προγραμματιστές προσπαθούν να καταλάβουν μια μεγάλη βάση κώδικα, τον οποίο δεν έχουν γράψει οι ίδιοι, αλλά αυτό δεν μπορεί να αντικαταστήσει το αρχικό σχέδιο.

Επαλήθευση προγράμματος και έλεγχος ιδιοτήτων

Ένα εργαλείο επαλήθευσης προγράμματος, δέχεται μια προδιαγραφή και ένα σώμα κώδικα και προσπαθεί να αποδείξει ότι ο κώδικας είναι πιστή υλοποίηση της προδιαγραφής. Αν η προδιαγραφή είναι μια πλήρης περιγραφή όλων όσων ένα πρόγραμμα πρέπει να κάνει, το εργαλείο επαλήθευσης μπορεί να εκτελέσει έλεγχο ισοδυναμίας ώστε να επιβεβαιώσει ότι ο κώδικας και οι προδιαγραφές συμφωνούν απόλυτα. Σπάνια οι προγραμματιστές έχουν τόσο λεπτομερείς προδιαγραφές, ώστε να χρησιμοποιηθούν σε έλεγχο ισοδυναμίας και ο κόπος που απαιτείται για να φτιαχτούν αυτές οι προδιαγραφές, μπορεί να είναι μεγαλύτερος από τη συγγραφή του προγράμματος. Ένα άλλο πρόβλημα είναι ότι ιστορικά, ο έλεγχος ισοδυναμίας δεν έχει καταφέρει να χειριστεί μεγάλες βάσεις κώδικα. Γι' αυτό είναι πιο διαδεδομένο αυτά τα εργαλεία να ελέγχουν ένα μέρος των προδιαγραφών και αυτή η μέθοδος μερικές φορές περιγράφεται ως έλεγχος ιδιοτήτων. Πολλά τέτοια εργαλεία εστιάζονται στις χρονικά ασφαλείς ιδιότητες (temporal safety properties). Μια

χρονικά ασφαλής ιδιότητα ορίζει μια σειρά γεγονότων, η οποία δεν πρέπει να συμβεί σε ένα πρόγραμμα. Ένα παράδειγμα μιας τέτοιας ιδιότητας είναι το “μια θέση μνήμης δεν πρέπει να διαβαστεί, αφού έχει αποδεσμευτεί”.

Εύρεση σφαλμάτων

Ο σκοπός ενός εργαλείου εύρεσης σφαλμάτων δεν είναι να αναφέρει θέματα μορφοποίησης, όπως ο ελεγκτής στυλ, ούτε να κάνει εξαντλητικό έλεγχο των προδιαγραφών, όπως ο ελεγκτής ιδιοτήτων. Η εύρεση σφαλμάτων απλά δείχνει σημεία στα οποία το πρόγραμμα θα συμπεριφερθεί με διαφορετικό τρόπο, από αυτόν που θα ήθελε ο προγραμματιστής. Τα περισσότερα εργαλεία του είδους, είναι εύκολα στη χρήση γιατί περιέχουν ένα σύνολο ιδιωμάτων (κανόνων), τα οποία περιγράφουν πρότυπα σφαλμάτων. Τα πιο εξελιγμένα εργαλεία εύρεσης σφαλμάτων, μπορούν να επεκτείνουν τα πρότυπα τους, εξάγοντας προδιαγραφές από τον ίδιο τον κώδικα. Για παράδειγμα, αν ένα πρόγραμμα java χρησιμοποιεί το ίδιο κλείδωμα συγχρονισμού για να περιορίσει την πρόσβαση σε μια συγκεκριμένη μεταβλητή στις 99 από τις 100 φορές, είναι πιθανό ότι το ίδιο κλείδωμα θα έπρεπε να χρησιμοποιηθεί και την εκατοστή.

Επισκόπηση ασφάλειας

Τα εργαλεία στατικής ανάλυσης που επικεντρώνονται στην ασφάλεια, χρησιμοποιούν ίδιες τεχνικές με άλλα εργαλεία, αλλά ο διαφορετικός τους στόχος σημαίνει ότι τις εφαρμόζουν διαφορετικά. Τα πρώτα εργαλεία ασφάλειας ITS4 [14], RATS [15] και Flawfinder [16], δεν ήταν πολλά περισσότερα από μια αποθεωμένη grep. Στο μεγαλύτερο κομμάτι τους, έψαχναν των κώδικα για να βρουν κλήσεις σε συναρτήσεις όπως οι strcpy(), οι οποίες είναι εύκολο να χρησιμοποιηθούν εσφαλμένα και πρέπει να ελεγχθούν στη φάση της χειροκίνητης επισκόπησης κώδικα. Με αυτήν τη λογική, ήταν πιο κοντά στα εργαλεία ελέγχου στυλ, γιατί τα προβλήματα που έβρισκαν δεν ήταν απαραίτητα σφάλματα ασφάλειας, αλλά ήταν ενδεικτικές προειδοποιήσεις για έλεγχο. Αυτού του είδους τα εργαλεία κατηγορήθηκαν ότι είχαν υψηλό ρυθμό ψευδώς θετικών, γιατί οι χρήστες μετέφραζαν τις προειδοποιήσεις ως λίστα σφαλμάτων και όχι ενδείξεις για χειροκίνητη επισκόπηση. Τα πιο σύγχρονα εργαλεία είναι περισσότερο ένα υβρίδιο ελεγκτών ιδιοτήτων και ανιχνευτών

σφαλμάτων. Πολλές ιδιότητες ασφάλειας, μπορούν να εκφραστούν ως ιδιότητες προγράμματος. Για παράδειγμα, η αναζήτηση για υπερχειλίσεις στοίβας, μπορούν να εκφραστούν ως “το πρόγραμμα δεν θα προσπελάσει μια διεύθυνση μνήμης, έξω από τα όρια της δεσμευμένης μνήμης”. Από τη σκοπιά της εύρεσης σφαλμάτων, τα εργαλεία ασφάλειας υιοθετούν την ιδέα ότι οι προγραμματιστές συχνά επαν-ανακαλύπτουν την ίδια ανασφαλή μέθοδο επίλυσης ενός προβλήματος, το οποίο μπορεί να περιγραφεί ως ανασφαλές προγραμματιστικό ιδιώμα.

3.1.2 Πλεονεκτήματα και μειονεκτήματα

Τα προβλήματα ασφάλειας σε ένα πρόγραμμα, προκύπτουν αντίστοιχα με τα ορθογραφικά λάθη που κάνει ένας συγγραφέας. Μπορούν όμως να είναι το αποτέλεσμα της άγνοιας του προγραμματιστή σχετικά με την ασφάλεια λογισμικού. Δεν είναι σπάνιο ένας προγραμματιστής να μην έχει ιδέα για τους τρόπους με τους οποίους οι επιτιθέμενοι προσπαθούν να εκμεταλλευτούν ένα κομμάτι κώδικα. Με αυτά τα δεδομένα, η στατική ανάλυση είναι χρήσιμη στην εύρεση προβλημάτων ασφάλειας, για τους εξής λόγους:

- Τα εργαλεία στατικής ανάλυσης εφαρμόζουν ελέγχους διεξοδικά και με συνέπεια, χωρίς την προκατάληψη που μπορεί να έχει ένας προγραμματιστής για το ποια τμήματα του κώδικα είναι ενδιαφέροντα από άποψη ασφάλειας ή για το ποια τμήματα είναι εύκολο να ελεγχθούν με δυναμική ανάλυση.

- Εξετάζοντας τον ίδιο τον κώδικα, τα εργαλεία στατικής ανάλυσης συχνά καταδεικνύουν την πηγή του προβλήματος ασφάλειας και όχι απλά ένα σύμπτωμα. Αυτό είναι ιδιαίτερα σημαντικό γιατί εξασφαλίζει την πλήρη διόρθωση ενός προβλήματος.

- Η στατική ανάλυση μπορεί να βρει σφάλματα στα πρώτα στάδια ανάπτυξης, ακόμα και πριν εκτελεστεί το πρόγραμμα για πρώτη φορά. Η έγκαιρη εύρεση ενός σφάλματος, δεν μειώνει απλά το κόστος διόρθωσης, αλλά η άμεση ανάδραση

βοηθάει στην καθοδήγηση του προγραμματιστή. Ο προγραμματιστής μπορεί να έχει την ευκαιρία να διορθώσει προβλήματα, τα οποία δεν γνώριζε εξαρχής ότι μπορούν να προκύψουν. Τα σενάρια επίθεσης και οι πληροφορίες για τον κώδικα που παρέχουν τα εργαλεία στατικής ανάλυσης, μπορούν να μεταφέρουν γνώση στους προγραμματιστές.

- Όταν ένας ερευνητής ασφάλειας ανακαλύψει μια νέα ποικιλία επίθεσης, τα εργαλεία στατικής ανάλυσης κάνουν εύκολο τον επανέλεγχο ενός μεγάλου σώματος κώδικα, ώστε να βρεθεί που μπορεί να επιτύχει το νέο είδος επίθεσης. Κάποια σφάλματα ασφάλειας υπάρχουν για χρόνια πριν ανακαλυφθούν και αυτό καθιστά την δυνατότητα επανελέγχου παλαιού κώδικα για νέου είδους προβλήματα, ανεκτίμητη.

Οι μεγαλύτερες αντιρρήσεις σχετικά με τα εργαλεία στατικής ανάλυσης, είναι ότι παράγουν πολύ θόρυβο. Συγκεκριμένα, παράγουν πολλά ψευδώς θετικά, τα οποία αναφέρονται και ως ψευδείς προειδοποιήσεις. Το ψευδώς θετικό, είναι ένα αναφερόμενο πρόβλημα, χωρίς αυτό πραγματικά να υπάρχει. Ένα μεγάλο πλήθος ψευδών θετικών, μπορεί να προκαλέσει πραγματικές δυσκολίες. Όχι μόνο καθιστά τον έλεγχο δυσκολότερο, γιατί ο προγραμματιστής πρέπει να ελέγξει έναν μεγάλο κατάλογο σφαλμάτων, αλλά μπορεί επίσης να παραβλέψει ένα πραγματικό πρόβλημα, θαμμένο σε μια ατελείωτη λίστα.

Τα ψευδώς αρνητικά, είναι σίγουρα ανεπιθύμητα, αλλά από τη σκοπιά της ασφάλειας είναι πολύ χειρότερα. Με ένα ψευδώς αρνητικό, υπάρχει ένα πρόβλημα στο πρόγραμμα, αλλά το εργαλείο δεν το αναφέρει. Το κόστος του ψευδώς θετικού, είναι ο χρόνος που δαπανήθηκε στον έλεγχό του. Το κόστος όμως του ψευδώς αρνητικού είναι κατά πολύ μεγαλύτερος. Όχι μόνο παραμένει μια ευπάθεια στο λογισμικό, αλλά δημιουργείται επίσης μια ψευδής αίσθηση ασφάλειας, από το γεγονός ότι το εργαλείο στατικής ανάλυσης αναφέρει πως δεν υπάρχει πρόβλημα.

3.1.3 Θεωρητικοί περιορισμοί

Η στατική ανάλυση είναι εγγενώς μη αποκρίσιμο πρόβλημα. Ο Alan Turing απέδειξε τη δεκαετία του 1940 ότι δεν μπορεί να υπάρξει ένας αλγόριθμος ο οποίος θα αποφασίζει αν ένα πρόγραμμα τερματίζει, έχοντας ως είσοδο μόνο την περιγραφή του [17]. Το θεώρημα του Rice [18] επεκτείνει το πρόβλημα τερματισμού του Turing και ορίζει ότι δεν υπάρχει γενική και αποτελεσματική μέθοδος, η οποία θα αποφασίζει αν ένα πρόγραμμα παράγει σφάλματα στην εκτέλεση ή αν παραβιάζει συγκεκριμένες προδιαγραφές. Το αποτέλεσμα αυτής της μη αποκρισιμότητας, είναι ότι όλα τα εργαλεία στατικής ανάλυσης θα παράγουν κάποια ψευδώς θετικά ή ψευδώς αρνητικά.

Το ακόλουθο κομμάτι ψευδοκώδικα, μπορεί εύκολα να δείξει αυτό το πρόβλημα:

```
bother(function f) {
      if (is_safe(f))
           call unsafe();
}

b = bother;
bother(b);
```

Έστω ότι υπάρχει μια συνάρτηση is_safe(), η οποία αποφασίζει ορθώς αν μια συνάρτηση είναι ασφαλής και η ίδια είναι επίσης ασφαλής. Η συνάρτηση bother(), δέχεται ως όρισμα μια συνάρτηση (f) και αν αυτή είναι ανασφαλής, καλεί την ανασφαλή συνάρτηση unsafe(). Ο παραπάνω κώδικας, καλεί την bother() με όρισμα τον εαυτό της. Αν η is_safe() αποφανθεί ότι η bother() είναι ασφαλής, η bother() θα καλέσει την ανασφαλή unsafe(). Αντίστοιχα αν η is_safe() αποφανθεί ότι η bother() είναι ανασφαλής, αυτή δεν θα καλέσει την unsafe(), οπότε δεν θα υπάρχει πρόβλημα. Και οι δύο περιπτώσεις οδηγούν σε αντίφαση, οπότε η unsafe() δεν μπορεί να κάνει αυτό που ορίστηκε.

3.1.4 Ανάλυση πηγαίου/μεταγλωττισμένου κώδικα

Τα περισσότερα εργαλεία στατικής ανάλυσης εξετάζουν ένα πρόγραμμα κοιτάζοντας τον πηγαίο κώδικα, αλλά υπάρχουν και κάποια που ελέγχουν τον εκτελέσιμο κώδικα ή τον bytecode. Η περίπτωση του μεταγλωττισμένου κώδικα έχει δύο πλεονεκτήματα:

- Το εργαλείο δεν χρειάζεται να μαντέψει πως ο μεταγλωττιστής θα μεταφράσει τον κώδικα, γιατί αυτό έχει ήδη γίνει. Αφαιρώντας τον μεταγλωττιστή, αφαιρείται και η ασάφεια.

- Ο πηγαίος κώδικας μπορεί να είναι δύσκολο να ελεγχθεί. Σε κάποιες περιπτώσεις, είναι ευκολότερη η ανάλυση του εκτελέσιμου κώδικα, γιατί απλά είναι ευκολότερα διαθέσιμος.

Αυτή η μέθοδος όμως έχει και μειονεκτήματα:

- Η κατανόηση του εκτελέσιμου κώδικα μπορεί να είναι δύσκολη. Αυτό ισχύει περισσότερο σε κώδικες που επιτρέπουν μεταβλητού μήκους εντολές, όπως στην περίπτωση του Intel x86, επειδή η σημασία του προγράμματος αλλάζει ανάλογα με το σημείο έναρξης της αποκωδικοποίησης. Μερικά εργαλεία στατικής ανάλυσης χρησιμοποιούν πληροφορίες από εργαλεία δυναμικής ανάλυσης, για να αντιμετωπίσουν αυτό το πρόβλημα. Ακόμα κι έτσι όμως, ο δυαδικός κώδικας δεν περιέχει πληροφορίες για τους τύπους. Η έλλειψη πληροφοριών για τους τύπους δυσκολεύει την ανάλυση, όπως επίσης και οι βελτιστοποιήσεις που εφαρμόζει ο μεταγλωττιστής. Γλώσσες όπως η Java, οι οποίες μεταγλωττίζονται σε bytecode, δεν έχουν αυτά τα προβλήματα και περιέχονται πληροφορίες τύπων στο εκτελέσιμο. Και σε αυτήν την περίπτωση όμως, οι μετασχηματισμοί που κάνει ο μεταγλωττιστής μπορεί να αφαιρέσει ή να αποκρύψει πληροφορίες σχετικά με το σκοπό του προγραμματιστή.

- Η ανάλυση εκτελέσιμου κώδικα, δυσχεραίνει τη δημιουργία χρήσιμων αναφορών σφαλμάτων. Οι περισσότεροι

προγραμματιστές θα ήθελαν τα ευρήματα εκφρασμένα σε πηγαίο κώδικα. Αυτό απαιτεί από το εργαλείο στατικής ανάλυσης να συνδέσει την ανάλυσή του, από τον εκτελέσιμο, πίσω στον πηγαίο κώδικα. Αν ο εκτελέσιμος κώδικας περιέχει πληροφορίες αποσφαλμάτωσης, αυτό μπορεί να μην είναι πολύ δύσκολο. Αν δεν τις περιέχει, ή ο μεταγλωττιστής έχει βελτιστοποιήσει το εκτελέσιμο σε σημείο που δυσκολεύει την αντιστοίχιση στον πηγαίο κώδικα, θα είναι πολύ δύσκολο να παραχθούν χρήσιμες αναφορές.

Σε γενικές γραμμές, για μορφές εκτελέσιμου κώδικα όπως τον bytecode της Java, δεν υπάρχει ξεκάθαρη ορθή απάντηση. Ο πηγαίος κώδικας περιέχει περισσότερες πληροφορίες, αλλά ο bytecode είναι ευκολότερος στην ανάλυση. Στην περίπτωση του δυαδικού εκτελέσιμου, τα μειονεκτήματα εύκολα κυριαρχούν. Η ανάλυση του πηγαίου κώδικα είναι και πιο εύκολη και πιο αποτελεσματική.

3.2 Κατασκευή μοντέλου

Το πρώτο πράγμα που χρειάζεται να κάνει ένα εργαλείο στατικής ανάλυσης, είναι να μετατρέψει τον κώδικα σε ένα μοντέλο προγράμματος, δηλαδή σε ένα σύνολο δομών δεδομένων που θα αναπαριστούν τον κώδικα. Το μοντέλο αυτό εξαρτάται από το είδος της στατικής ανάλυσης, αλλά γενικά οι σχετικές μέθοδοι δανείζονται πολλά από τους μεταγλωττιστές. Μάλιστα πολλές τεχνικές στατικής ανάλυσης, αναπτύχθηκαν κατά την έρευνα σχετικά με προβλήματα μεταγλωττιστών. Ακολουθεί μια σύντομη παρουσίαση των σημαντικότερων αυτών τεχνικών.

3.2.1 Λεκτική ανάλυση

Τα εργαλεία που χειρίζονται πηγαίο κώδικα, ξεκινούν μετατρέποντάς τον σε μια σειρά λεκτικών μονάδων (tokens), απορρίπτοντας ασήμαντους χαρακτήρες, όπως τα κενά και τα σχόλια. Η δημιουργία αυτής της σειράς ονομάζεται λεκτική ανάλυση. Για παράδειγμα, ο παρακάτω κώδικας C:

```
if (ret) // probably true
mat[x][y] = END_VAL;
```

Παράγει την ακόλουθη σειρά λεκτικών μονάδων:

```
IF LPAREN ID(ret) RPAREN ID(mat) LBRACKET ID(x)
RBRACKET LBRACKET ID(y) RBRACKET EQUAL ID(END_VAL)
SEMI
```

Οι περισσότερες λεκτικές μονάδες περιγράφονται πλήρως από τον τύπο τους, εκτός από τη μονάδα ID, η οποία απαιτεί και το όνομα του αναγνωριστικού. Για την αναφορά των σφαλμάτων, απαιτείται και η θέση τους στον πηγαίο κώδικα.

Για τα πιο απλά εργαλεία στατικής ανάλυσης, η δουλειά έχει σχεδόν τελειώσει. Αν το μόνο που θα κάνει το εργαλείο είναι να ψάξει για ονόματα επικίνδυνων συναρτήσεων, αρκεί μια αναζήτηση στη σειρά των λεκτικών μονάδων. Αυτό έκαναν τα ITS4, RATS και το FlawFinder.

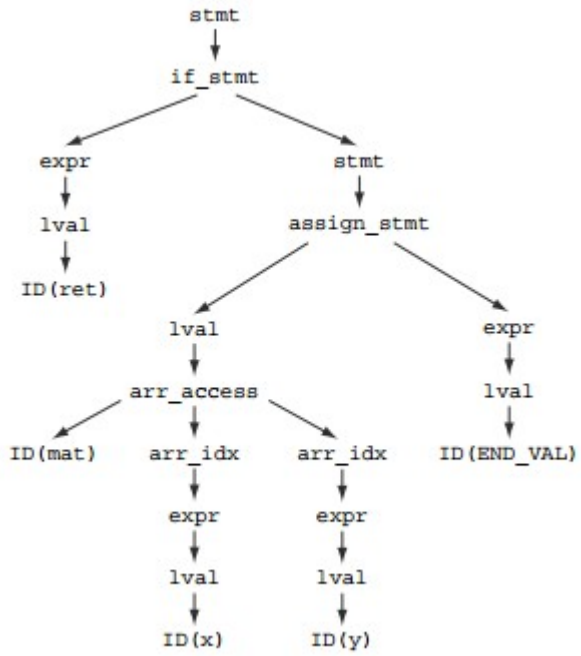
3.2.2 Συντακτική ανάλυση

Ένας συντακτικός αναλυτής, χρησιμοποιεί μια γραμματική χωρίς συμφραζόμενα (context-free grammar) για να ταιριάξει τη σειρά των λεκτικών μονάδων. Η γραμματική αποτελείται από μια σειρά κανόνων παραγωγής, οι οποίοι περιγράφουν τα σύμβολα της γλώσσας. Αυτό είναι ένα ενδεικτικό σύνολο κανόνων για το προηγούμενο παράδειγμα λεκτικής ανάλυσης:

```
stmt := if_stmt | assign_stmt
if_stmt := IF LPAREN expr RPAREN stmt
expr := lval
assign_stmt := lval EQUAL expr SEMI
lval = ID | arr_access
arr_access := ID arr_index+
arr_idx := LBRACKET expr RBRACKET
```

Ο συντακτικός αναλυτής ταιριάζει τις λεκτικές μονάδες με τους κανόνες παραγωγής. Αν κάθε σύμβολο συνδεθεί με το σύμβολο από

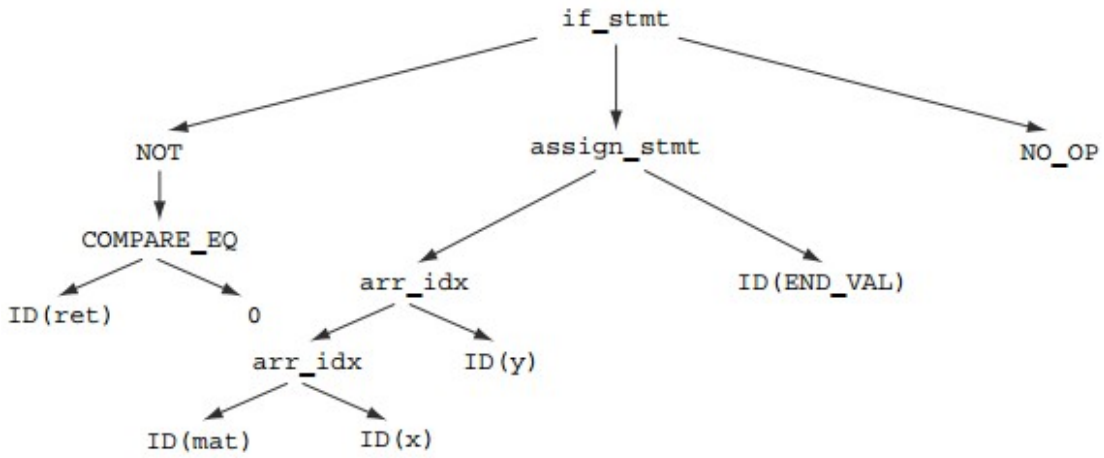
το οποίο παράχθηκε, σχηματίζεται ένα συντακτικό δέντρο. Ακολουθεί το (μερικό) συντακτικό δέντρο που θα προέκυπτε από τα προηγούμενα παραδείγματα:



Εικόνα 4: Δέντρο συντακτικής ανάλυσης

Είναι δυνατό να γίνει σημαντική ανάλυση πάνω σε ένα δέντρο συντακτικής ανάλυσης και κάποια είδη ελέγχου στυλ, γίνονται καλύτερα σε αυτό, επειδή αναπαριστά άμεσα τον κώδικα, όπως ακριβώς τον έγραψε ο προγραμματιστής. Ωστόσο, η εφαρμογή σύνθετης ανάλυσης απευθείας πάνω στο δέντρο συντακτικής ανάλυσης, παρουσιάζει διάφορες δυσκολίες. Οι κόμβοι του δέντρου προέρχονται απευθείας από τους κανόνες παραγωγής της γραμματικής και αυτοί οι κανόνες μπορούν να εισάγουν μη τερματικά σύμβολα, ο σκοπός των οποίων είναι να κάνουν τη συντακτική ανάλυση εύκολη και σαφή και δεν βοηθούν στην δημιουργία ενός εύκολα κατανοητού δέντρου. Γενικά είναι προτιμότερη η αφαίρεση από τις λεπτομέρειες της γραμματικής και από τις συντακτικές λεπτομέρειες που υπάρχουν στον κώδικα. Μια δομή δεδομένων που τα εφαρμόζει αυτά, ονομάζεται αφαιρετικό δέντρο συντακτικής ανάλυσης. Ο σκοπός του είναι να παρέχει μια τυποποιημένη έκδοση του προγράμματος, κατάλληλη για περαιτέρω ανάλυση. Αυτό το δέντρο συνήθως φτιάχνεται συσχετίζοντας το δέντρο παραγωγής κώδικα με τους κανόνες παραγωγής της γραμματικής. Για το παράδειγμα κώδικα που παρουσιάστηκε στην

αρχή, της ενότητας, ένα αφαιρετικό δέντρο συντακτικής ανάλυσης θα ήταν:



Εικόνα 5: Αφαιρετικό δέντρο συντακτικής ανάλυσης

Ανάλογα με τις ανάγκες του συστήματος, το αφαιρετικό δέντρο συντακτικής ανάλυσης μπορεί να περιέχει ένα μικρότερο σύνολο κατασκευασμάτων, από τη γλώσσα του κώδικα. Για παράδειγμα, οι κλήσεις των μεθόδων θα μπορούσαν να μετατραπούν σε κλήσεις συναρτήσεων και οι βρόχοι for και do θα μπορούσαν να μετατραπούν σε βρόχους while. Η σημαντική απλοποίηση του προγράμματος με αυτόν τον τρόπο, ονομάζεται ελάττωση. Γλώσσες που σχετίζονται στενά, όπως οι C και C++, μπορούν να ελαττωθούν αφαιρετικό δέντρο ίδιας μορφής, αν και μια τέτοια ελάττωση έχει το ρίσκο της απόκρυψης των προθέσεων του προγραμματιστή. Γλώσσες συντακτικά παρόμοιες, όπως οι C++ και Java, μπορούν να παράγουν αφαιρετικά δέντρα με πολλά κοινά, αλλά σίγουρα θα υπάρχουν ειδικές κατηγορίες κόμβων για τη λειτουργικότητα που απαντάται μόνο σε μία από τις δύο.

3.2.3 Σημασιολογική ανάλυση

Καθώς το αφαιρετικό δέντρο κατασκευάζεται, το εργαλείο δημιουργεί παράλληλα έναν πίνακα συμβόλων. ο πίνακας συμβόλων συσχετίζει κάθε αναγνωριστικό του προγράμματος, με έναν δείκτη στη δήλωση ή τον ορισμό του.

Με το αφαιρετικό δέντρο συντακτικής ανάλυσης και τον πίνακα συμβόλων, το εργαλείο είναι έτοιμο να κάνει έλεγχο τύπου. Ένα

εργαλείο στατικής ανάλυσης μπορεί να μην απαιτείται να αναφέρει σφάλματα τύπου με τον ίδιο τρόπο που το κάνει ο μεταγλωττιστής, αλλά οι πληροφορίες σχετικά με τους τύπους είναι κρίσιμα σημαντική στην ανάλυση μιας αντικειμενουστρεφούς γλώσσας, επειδή ο τύπος ενός αντικειμένου, ορίζει το σύνολο των μεθόδων που μπορεί να καλέσει. Επί πλέον είναι συνήθως επιθυμητό στο αφαιρετικό δέντρο, να μετατραπούν οι υπονοούμενες (implict) μετατροπές τύπων σε κατηγοτηματικές (explicit). Γι' αυτούς τους λόγους, ένα εξελιγμένο εργαλείο στατικής ανάλυσης, πρέπει να κάνει την ίδια δουλειά με τον μεταγλωττιστή, σχετικά με τους τύπους.

Στον τομέα των μεταγλωττιστών, η ανάλυση των συμβόλων και ο έλεγχος τύπου ονομάζονται σημασιολογική ανάλυση, γιατί ο μεταγλωττιστής δίνει σημασία στα σύμβολα που παρουσιάζονται στο πρόγραμμα. Τα εργαλεία στατικής ανάλυσης που χρησιμοποιούν αυτές τις δομές δεδομένων, έχουν το διακριτό πλεονέκτημα σε σχέση με αυτά που δεν τις χρησιμοποιούν. Για παράδειγμα, μπορούν ορθά να μεταφράσουν τη σημασία των υπερφορτωμένων τελεστών στη C++ ή να καθορίσουν ότι μια μέθοδος στη Java με το όνομα doPost(), είναι στην ουσία η υλοποίηση του HttpServlet. Αυτές οι ικανότητες επιτρέπουν σε ένα εργαλείο να εφαρμόσει χρήσιμους ελέγχους στη δομή του προγράμματος. Χρησιμοποιούμε τον όρο δομική ανάλυση, για τέτοιου είδους ελέγχους.

Μετά τη σημασιολογική ανάλυση, οι μεταγλωττιστές και τα περισσότερο εξελιγμένα εργαλεία στατικής ανάλυσης παίρνουν διαφορετικούς δρόμους. Ένας σύγχρονος μεταγλωττιστής χρησιμοποιεί το αφαιρετικό δέντρο συντακτικής ανάλυσης και τον πίνακα συμβόλων για να παράγει μια ενδιάμεση αναπαράσταση, μια γενική έκδοση κώδικα μηχανής, η οποία είναι χρήσιμη στη βελτιστοποίηση και στη μετέπειτα μετατροπή σε αντικειμενικό κώδικα για μια ειδική πλατφόρμα. Στην περίπτωση της στατικής ανάλυσης, δεν υπάρχει ξεκάθαρη οδός. Ανάλογα με τον τύπο της στατικής ανάλυσης που εφαρμόζεται, ένα εργαλείο μπορεί να μετατρέψει και άλλο το αφαιρετικό δέντρο ή μπορεί να παραγάγει τη δική του έκδοση της ενδιάμεσης αναπαράστασης, η οποία θα ταιριάζει στις ανάγκες του.

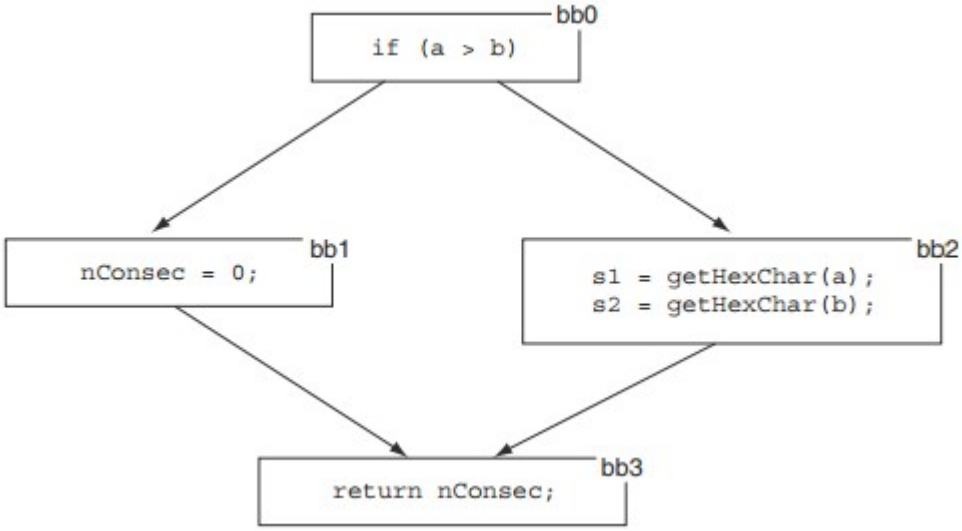
Αν ένα εργαλείο στατικής ανάλυσης χρησιμοποιεί την δικιά του ενδιάμεση αναπαράσταση, γενικά επιτρέπει τουλάχιστον ανάθεση, διακλάδωση, βρόχους και κλήσεις συναρτήσεων. Η ενδιάμεση αναπαράσταση ενός εργαλείου στατικής ανάλυσης, είναι συνήθως υψηλότερου επιπέδου από την αντίστοιχη ενός μεταγλωττιστή. Για παράδειγμα ένας μεταγλωττιστής της γλώσσας C, πιθανά θα μετατρέψει όλες τις αναφορές στα πεδία μιας δομής (struct) σε σχετικές διευθύνσεις, ενώ ένα εργαλείο στατικής ανάλυσης είναι πιθανότερο ότι θα συνεχίσει να αναφέρεται στα πεδία με το όνομά τους.

3.2.4 Ιχνηλάτηση ροής ελέγχου

Πολλοί αλγόριθμοι στατικής ανάλυσης (και τεχνικές βελτιστοποίησης μεταγλωττιστών) εξερευνούν τα διαφορετικά μονοπάτια εκτέλεσης που μπορούν να συμβούν, όταν εκτελεστεί μια συνάρτηση. Για να είναι αυτοί οι αλγόριθμοι αποδοτικοί, τα περισσότερα εργαλεία κατασκευάζουν ένα γράφημα ροής ελέγχου πάνω από το αφαιρετικό δέντρο συντακτικής ανάλυσης ή την ενδιάμεση αναπαράσταση. Οι κόμβοι σε αυτό το γράφημα είναι βασικά μπλοκ, δηλαδή ακολουθίες εντολών που θα εκτελεστούν πάντα αρχίζοντας από την πρώτη και καταλήγοντας στην τελευταία, χωρίς την πιθανότητα να παρακαμφθούν κάποιες από αυτές. Οι ακμές στο γράφημα ροής ελέγχου είναι κατευθυνόμενες και αναπαριστούν πιθανά μονοπάτια ροών ελέγχου μεταξύ βασικών μπλοκ. Ακμές προς τα πίσω, αναπαριστούν πιθανούς βρόχους. Για παράδειγμα, από τον ακόλουθο κώδικα C:

```
if (a > b) {
      nConsec = 0;
} else {
      s1 = getHexChar(1);
      s2 = getHexChar(2);
}
return nConsec;
```

Μπορεί να σχηματιστεί το ακόλουθο γράφημα ροής ελέγχου:



Εικόνα 6: Γράφημα ροής ελέγχου

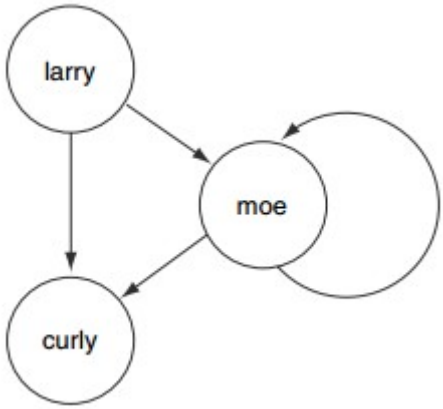
Τα τέσσερα βασικά μπλοκ έχουν τις ετικέτες bb0 έως bb3. Στο παραπάνω παράδειγμα οι εντολές κάθε μπλοκ είναι σε μορφή πηγαίου κώδικα, αλλά μια δομή δεδομένων βασικού μπλοκ ενός εργαλείου στατικής ανάλυσης, πιθανότερα θα περιέχει δείκτες στο αφαιρετικό δέντρο συντακτικής ανάλυσης ή στην ενδιάμεση αναπαράσταση.

Όταν ένα πρόγραμμα τρέχει, η ροή ελέγχου του μπορεί να περιγραφεί από την ακολουθία των βασικών μπλοκ που εκτελεί. Ένα ίχνος είναι μια ακολουθία βασικών μπλοκ, τα οποία ορίζουν ένα μονοπάτι δια μέσω του κώδικα. Υπάρχουν μόνο δύο πιθανά μονοπάτια εκτέλεσης του κώδικα του παραπάνω παραδείγματος. Αυτά τα δύο μονοπάτια αναπαρίστανται από δύο μοναδικά ίχνη στο γράφημα ροής ελέγχου: [bb0, bb1, bb3] και [bb0, bb2, bb3].

Ένα γράφημα κλήσεων αναπαριστά πιθανές ροές ελέγχου μεταξύ συναρτήσεων ή μεθόδων. Αν δεν υποστηρίζονται δείκτες συναρτήσεων ή εικονικές μέθοδοι, η κατασκευή του γραφήματος κλήσεων απαιτεί απλά την εύρεση των αναγνωριστικών των συναρτήσεων, μέσα σε κάθε συνάρτηση. Οι κόμβοι του γραφήματος αυτού αναπαριστούν συναρτήσεις και οι κατευθυνόμενες ακμές αναπαριστούν την πιθανή κλήση μιας συνάρτησης από μια άλλη. Για παράδειγμα από το παρακάτω πρόγραμμα, το οποίο περιλαμβάνεις τρεις συναρτήσεις:

```
int larry(int fish) {
      if (fish) {
           moe(1);
      } else {
           curly();
      }
}
int moe(int scissors) {
      if (scissors) {
           curly();
           moe(0);
      } else {
           curly();
      }
}
int curly() {
      /* empty */
}
```

Προκύπτει το ακόλουθο γράφημα κλήσεων:



Εικόνα 6: Γράφημα κλήσεων

Όταν καλούνται δείκτες σε συναρτήσεις ή εικονικές μέθοδοι, το εργαλείο μπορεί να χρησιμοποιήσει ένα συνδυασμό από ανάλυση ροής δεδομένων και ανάλυση τύπων δεδομένων, για να περιορίσει το σύνολο των πιθανών συναρτήσεων που μπορούν να κληθούν σε ένα τμήμα του κώδικα. Αν το πρόγραμμα φορτώνει τμήματα κώδικα δυναμικά κατά την εκτέλεση, δεν υπάρχει τρόπος να διασφαλιστεί

ότι το γράφημα κλήσεων είναι πλήρες, γιατί προφανώς μπορεί να κληθεί κώδικας που δεν είναι ορατός στη φάση της ανάλυσης.

Για τα συστήματα λογισμικού τα οποία περιλαμβάνουν περισσότερες από μία γλώσσες προγραμματισμού, ή τα οποία αποτελούνται από πολλαπλές συνεργαζόμενες διεργασίες, ένα εργαλείο στατικής ανάλυσης ιδανικά θα κατασκευάσει ένα γράφημα ροής ελέγχου, το οποίο θα περιέχει τις συνδέσεις μεταξύ των διαφορετικών τμημάτων. Σε πολλά συστήματα, τα αρχεία παραμέτρων περιέχουν τα δεδομένα τα οποία χρειάζονται στην κατασκευή τέτοιων γραφημάτων κλήσεων.

3.2.5 Ιχνηλάτηση ροής δεδομένων

Οι αλγόριθμοι ανάλυσης ροής δεδομένων, εξετάζουν τον τρόπο με τον οποίο τα δεδομένα μετακινούνται δια μέσω του προγράμματος. Οι μεταγλωττιστές εφαρμόζουν αυτήν την ανάλυση για να δεσμεύσουν καταχωρητές, να αφαιρέσουν νεκρό κώδικα και για πολλές άλλες βελτιστοποιήσεις.

Η ανάλυση ροής δεδομένων συνήθως περιλαμβάνει τη διαπέραση του γραφήματος ροής ελέγχου της συνάρτησης και την καταγραφή των σημείων όπου παράγονται και χρησιμοποιούνται οι τιμές των δεδομένων. Η μετατροπή μιας συνάρτησης σε μορφή μοναδικής στατικής ανάθεσης (static single assignment), είναι χρήσιμη σε πολλά προβλήματα ροής δεδομένων. Μια συνάρτηση σε αυτήν τη μορφή, επιτρέπεται να θέσει τιμή σε μια μεταβλητή, μόνο μία φορά. Για την τήρηση αυτού του κανόνα, νέες μεταβλητές πρέπει να εισαχθούν στο πρόγραμμα. Στους μεταγλωττιστές, οι νέες μεταβλητές συνήθως ονομάζονται προσθέτοντας έναν αριθμό δείκτη στο αρχικό όνομα μιας μεταβλητής. Αν για παράδειγμα γίνεται ανάθεση σε μια μεταβλητή x , τρεις φορές, η μετατροπή του προγράμματος θα αναφέρεται στις μεταβλητές x_1, x_2 και x_3.

Η μορφή μοναδικής στατικής ανάθεσης είναι χρήσιμη γιατί για οποιαδήποτε μεταβλητή του προγράμματος, είναι πολύ εύκολο να βρεθεί το σημείο όπου αυτή παίρνει τιμή. Αυτή η ιδιότητα έχει πολλές χρήσεις. Για παράδειγμα, αν μια τέτοια μεταβλητή παίρνει την τιμή μιας σταθεράς, η σταθερή τιμή μπορεί να αντικαταστήσει

όλα τα σημεία χρήσης της μεταβλητής. Αυτή η μέθοδος ονομάζεται διάδοση σταθεράς. Η μέθοδος διάδοσης σταθεράς είναι χρήσιμη και στην εύρεση προβλημάτων ασφάλειας, όπως τη χρήση ενσωματωμένων στον κώδικα διαπιστευτηρίων ή κλειδιών κρυπτογράφησης.

3.2.6 Διάδοση μόλυνσης

Τα εργαλεία ασφάλειας χρειάζεται να γνωρίζουν ποιες τιμές σε ένα πρόγραμμα θα μπορούσε να ελέγξει ο επιτιθέμενος. Η χρήση της ροής δεδομένων στον καθορισμό των τιμών που μπορεί να ελέγξει ένας επιτιθέμενος, ονομάζεται διάδοση μόλυνσης. Απαιτεί γνώση των σημείων από τα οποία εισέρχεται η πληροφορία στο πρόγραμμα και πως αυτή μετακινείται εσωτερικά στο πρόγραμμα. Η διάδοση μόλυνσης είναι το κλειδί στην αναγνώριση πολλών προβλημάτων αναπαράστασης και ελέγχου της εισόδου. Για παράδειγμα, αν ένα πρόγραμμα περιέχει μια εκμεταλλεύσιμη ευπάθεια υπερχείλισης στοίβας, σχεδόν πάντα περιέχει μια ροή δεδομένων από συνάρτηση εισόδου, σε ευπαθή λειτουργία.

Η ιδέα της διάδοσης μόλυνσης, δεν περιορίζεται στα εργαλεία στατικής ανάλυσης. Πιθανά η πιο γνωστή υλοποίηση δυναμικής διάδοσης μόλυνσης είναι στη γλώσσα Perl, όπου όταν ενεργοποιηθεί ένας τρόπος λειτουργίας, χρησιμοποιείται ένας μηχανισμός εκτέλεσης ο οποίος επιβάλει ότι κάθε είσοδος από το χρήστη, πρέπει να ελέγχεται με βάση μια κανονική έκφραση, πριν χρησιμοποιηθεί.

3.2.7 Ψευδώνυμα δεικτών

Τα ψευδώνυμα δεικτών είναι ένα άλλο πρόβλημα ροής δεδομένων. Ο σκοπός της σχετικής ανάλυσης, είναι να κατανοήσει ποιοι δείκτες θα μπορούσαν να δείχνουν στην ίδια θέση μνήμης. Οι αλγόριθμοι που χρησιμοποιούνται περιγράφουν τις σχέσεις των δεικτών με όρους όπως “πρέπει να είναι ψευδώνυμο”, “μπορεί να είναι ψευδώνυμο” και “δεν μπορεί να είναι ψευδώνυμο”. Πολλές βελτιστοποιήσεις που κάνουν οι μεταγλωττιστές απαιτούν μια μορφή αυτής της ανάλυσης για τον έλεγχο ορθότητας. Για

παράδειγμα, ένας μεταγλωττιστής θα μπορούσε να αναδιατάξει τις ακόλουθες δύο εντολές, μόνο αν οι δείκτες p1 και p2 δεν δείχνουν στην ίδια διεύθυνση μνήμης:

```
*p1 = 1;
*p2 = 2;
```

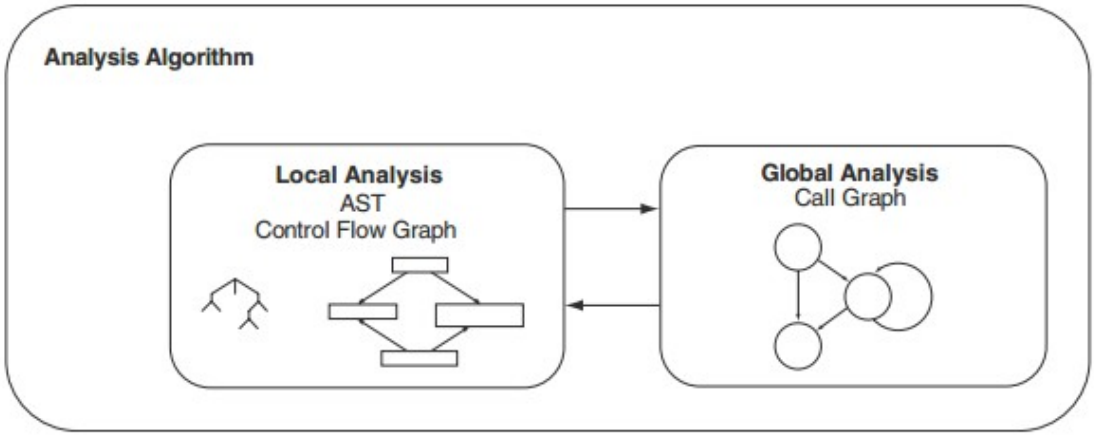
Για τα εργαλεία ασφάλειας, αυτή η ανάλυση είναι σημαντική στην διάδοση μόλυνσης. Ένας ευαίσθητος στη ροή, ιχνηλατών τη μόλυνση αλγόριθμος, χρειάζεται να κάνει ανάλυση ψευδωνύμων για να καταλάβει ότι τα δεδομένα ρέουν από τη getUserInput() στη processInput() στον ακόλουθο κώδικα:

```
p1 = p2;
*p1 = getUserInput();
processInput(*p2);
```

3.3 Αλγόριθμοι ανάλυσης

Το κίνητρο πίσω από τη χρήση εξελιγμένων αλγορίθμων στατικής ανάλυσης, είναι η βελτίωση της ευαισθησίας στα συμφραζόμενα. Δηλαδή, ο καθορισμός των καταστάσεων και των προϋποθέσεων, κάτω από τις οποίες τρέχει ένα κομμάτι κώδικα. Καλύτερη ευαισθησία στα συμφραζόμενα, σημαίνει καλύτερη εκτίμηση του κινδύνου που αντιπροσωπεύει ο κώδικας. Είναι εύκολο να βρεθούν όλα τα σημεία που καλείται η strcpy() και η αναφορά ότι πρέπει να αντικατασταθούν, αλλά είναι πολύ δυσκολότερο να αναφερθούν μόνο οι κλήσεις της strcpy(), οι μπορεί να επιτρέψουν σε έναν επιτιθέμενο να προκαλέσει υπερχείλισης τη στοίβας.

Όλες οι εξελιγμένες στρατηγικές ανάλυσης, περιέχουν τουλάχιστον δύο βασικά τμήματα. Ένα τμήμα ενδο-συναρτησιακής ανάλυσης, για την ανάλυση μιας συγκεκριμένης συνάρτησης και ένα τμήμα δια-συναρτησιακής ανάλυσης για την ανάλυση των αλληλεπιδράσεων μεταξύ συναρτήσεων. Δύο καλύτεροι όροι γι' αυτά τα είδη αναλύσεων, είναι τοπική ανάλυση και καθολική ανάλυση αντίστοιχα. Σχηματικά, οι δύο μέθοδοι με τις αντίστοιχες δομές δεδομένων είναι:



Εικόνα 6: Τοπική και καθολική ανάλυση

3.3.1 Έλεγχος ισχυρισμών

Πολλές ιδιότητες ασφάλειας, μπορούν να εκφραστούν ως ισχυρισμοί, οι οποίοι πρέπει να είναι αληθείς, για να είναι ασφαλές το πρόγραμμα. Για τον έλεγχο της υπερχείλισης στοίβας στον κώδικα:

```
strcpy(dest, src);
```

αρκεί η εισαγωγή του παρακάτω ισχυρισμού, πριν την κλήση της strcpy():

```
assert(alloc_size(dest) > strlen(src));
```

Αν η προγραμματιστική λογική εγγυάται ότι αυτός ο ισχυρισμός θα είναι πάντα αληθής, δεν μπορεί να υπάρξει υπερχείλιση. Αν υπάρχει ένα σύνολο προϋποθέσεων, κάτω από τις οποίες ο ισχυρισμός μπορεί να αποτύχει, ο αναλυτής πρέπει να αναφέρει πιθανή υπερχείλιση στοίβας. Η ίδια προσέγγιση λειτουργεί αντίστοιχα καλά στις περιπτώσεις των SQL injection, XSS και άλλων κατηγοριών ευπαθειών.

Κυρίως χρησιμοποιούνται τριών ειδών ισχυρισμοί για τον έλεγχο ιδιοτήτων ασφάλειας:

- Οι επικρατέστερες μορφές προβλημάτων ασφάλειας, προκύπτουν από την εμπιστοσύνη των προγραμματιστών στην

είσοδο. Έτσι ένα εργαλείο χρειάζεται να ελέγξει ισχυρισμούς σχετικά με το επίπεδο εμπιστοσύνης στα δεδομένα, όπως αυτά μετακινούνται μέσα στο πρόγραμμα. Αυτά είναι προβλήματα διάδοσης μόλυνσης. Τα SQL injection και XSS είναι δύο τύποι ευπαθειών που θα οδηγήσουν ένα εργαλείο στη δημιουργία ισχυρισμών, σχετικά με τη διάδοση μόλυνσης. Στο απλούστερο σενάριο, μια τιμή είναι μολυσμένη (πιθανά ελεγχόμενη από έναν επιτιθέμενο), ή όχι. Ένας επιτιθέμενος μπορεί για παράδειγμα να ελέγχει τα περιεχόμενα μιας προσωρινής μνήμης, αλλά όχι το μέγεθός της.

- Η αναζήτηση για εκμεταλλεύσιμες υπερχειλίσεις στοίβας, οδηγεί στους ισχυρισμούς οι οποίοι είναι παρόμοιοι με αυτούς που προκύπτουν από τη διάδοση μόλυνσης, αλλά η απόφαση για την πιθανότητα πρόκλησης υπερχείλισης σε μια προσωρινή μνήμη, απαιτεί περισσότερα. Το εργαλείο πρέπει επίσης να ξέρει το μέγεθος της προσωρινής μνήμης και τον τρόπο δείκτη προσπέλασής της. Αυτά ονομάζονται προβλήματα ανάλυσης εύρους γιατί απαιτούν τη γνώση του εύρους των πιθανών τιμών μιας μεταβλητής.

- Σε κάποιες περιπτώσεις, τα εργαλεία ενδιαφέρονται λιγότερο για τις συγκεκριμένες τιμές δεδομένων, παρά για την κατάσταση ενός αντικειμένου κατά την εκτέλεση του προγράμματος. Αυτό ονομάζεται κατάσταση τύπου. Οι μεταβλητές μπορεί να έχουν διαφορετικούς τύπους σε κάθε σημείο του κώδικα. Για παράδειγμα, μια περιοχή μνήμης μπορεί να είναι σε δύο καταστάσεις. Δεσμευμένη, καλώντας τη malloc() ή την αποδεσμευμένη, μετά την κλήση της free(). Αν ένα πρόγραμμα διαγράψει όλες τις αναφορές σε μια θέση μνήμης, όσο είναι στην κατάσταση δεσμευμένη, η μνήμη έχει διαρρεύσει. Αν ένας δείκτης στη μνήμη περαστεί στη free() όταν αυτή η μνήμη είναι σε κατάσταση αποδεσμευμένη, μια ευπάθεια διπλής free είναι παρούσα. Πολλές τέτοιες ιδιότητες χρονικής ασφάλειας, μπορούν να εκφραστούν ως μικρά πεπερασμένα αυτόματα.

3.3.2 Τοπική ανάλυση

Η τοπική παρουσιάζει πολλά προβλήματα τα οποία σχετίζονται κυρίως με τη διακλάδωση του προγράμματος και τους βρόχους. Γι' αυτό το λόγο τα εργαλεία στατικής ανάλυσης χρησιμοποιούν λιγότερο ακριβείς, αλλά περισσότερο αξιόπιστες μεθόδους. Ενδεικτικά αναφέρονται οι:

Αφαιρετική ερμηνεία
Είναι μια γενική τεχνική που αναπτύχθηκε από τους Cousot & Cousot [19], για την αφαίρεση όψεων του προγράμματος που δεν είναι σχετικές με τις ιδιότητες που ελέγχονται και έπειτα εφαρμόζοντας μια ερμηνεία βάση της επιλεγμένης αφαίρεσης.

Τα προβλήματα που εισάγονται από τους βρόχους μπορούν να λυθούν εφαρμόζοντας μια μη ευαίσθητη στη ροή ανάλυση, σύμφωνα με την οποία δεν λαμβάνεται υπόψιν η σειρά των εντολών. Από τη σκοπιά του προγραμματιστή, αυτό μπορεί να φανεί υπερβολικό. Η σειρά με την οποία εκτελούνται οι εντολές είναι σημαντική. Αλλά η μη ευαίσθητη στη ροή ανάλυση, εγγυάται ότι ελέγχονται όλες οι ταξινομήσεις των εντολών. Αυτό εξαλείφει την ανάγκη για σύνθετη ανάλυση ροής, αλλά το κόστος είναι ότι πολλές αδύνατες ταξινομήσεις εντολών μπορεί να αναλυθούν.

Μετατροπείς κατηγορημάτων
Μια εναλλακτική της προσομοίωσης ή της ερμηνείας, είναι η εξαγωγή των απαιτήσεων που ορίζει μια συνάρτηση, σε αυτούς που την καλούν. Ο Dijkstra [20], εισήγαγε την έννοια της πιο αδύναμης προϋπόθεσης (weakest precondition) που εξάγεται από ένα πρόγραμμα, χρησιμοποιώντας μετατροπείς κατηγορημάτων. Μια πιο αδύναμη προϋπόθεση για ένα πρόγραμμα, είναι το ελάχιστο σύνολο απαιτήσεων στους καλούντες ενός προγράμματος, οι οποίες είναι απαραίτητες για την κατάληξη σε μια επιθυμητή τελική κατάσταση. Για παράδειγμα, για να ολοκληρωθεί επιτυχημένα η εντολή:

```
assert (x < y);
```

απαιτείται ο κώδικας που οδηγεί σε αυτήν, να ικανοποιεί τουλάχιστον την απαίτηση (x < y). Είναι πιθανό ο κώδικας που οδηγεί στην εντολή, να ικανοποιεί πάντα μια πιο ισχυρή απαίτηση, όπως την:

$$(x < 0 \land y > 0)$$

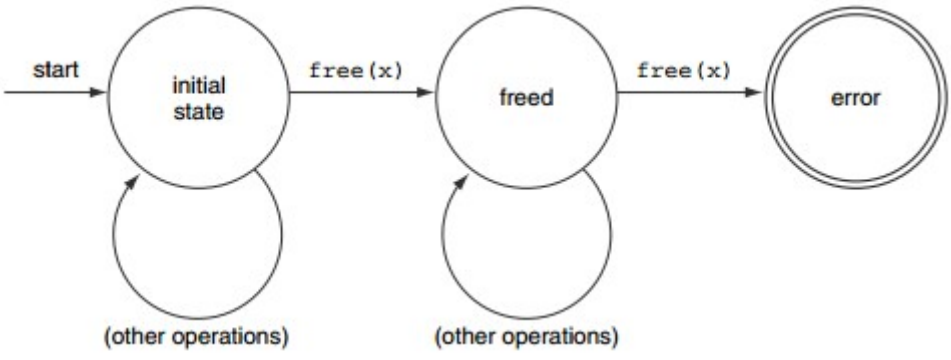
Ωστόσο, η παραπάνω εντολή assert, δεν το απαιτεί αυτό. Αν θέλουμε ένα πρόγραμμα να φτάσει στην τελική κατάσταση r, μπορούμε να γράψουμε έναν μετατροπέα κατηγορημάτων για την εξαγωγή της πιο αδύναμης απαίτησης:

$$WP(assert(p), r) = p \land r$$

Οι μετατροπείς κατηγορημάτων είναι ελκυστικοί, γιατί παράγοντας μια προαπαίτηση από ένα σώμα κώδικα, αφαιρούν τις λεπτομέρειες του προγράμματος και δημιουργούν μια σύνοψη των απαιτήσεων που αυτό επιβάλει στον καλούντα.

Έλεγχος μοντέλου

Οι ιδιότητες χρονικής ασφάλειας, όπως η “η μνήμη πρέπει να αποδεσμευτεί μόνο μία φορά” και η “γίνεται αναφορά μόνο σε δείκτες που δεν είναι null”, είναι εύκολο να εκφραστούν ως μικρά πεπερασμένα αυτόματα. Ακολουθεί ένα αυτόματο για την πρώτη ιδιότητα:



Εικόνα 7: Πεπερασμένο αυτόματο για την ιδιότητα “η μνήμη πρέπει να αποδεσμευτεί μόνο μία φορά”

Μια προσέγγιση ελέγχου μοντέλου δέχεται τέτοιες ιδιότητες ως απαιτήσεις, μετατρέπει το πρόγραμμα σε ένα αυτόματο (μοντέλο) και συγκρίνει τις απαιτήσεις με το μοντέλο.

3.3.3 Καθολική ανάλυση

Η απλούστερη προσέγγιση καθολικής ανάλυσης, είναι η υπόθεση ότι όλα τα προβλήματα θα παρουσιαστούν αν εξεταστεί μια συνάρτηση τη φορά. Αυτή η υπόθεση είναι ιδιαίτερα κακή για πολλά προβλήματα ασφάλειας, ειδικά γι' αυτά που σχετίζονται με την επικύρωση εισόδου, γιατί συχνά απαιτούν έλεγχο πέρα από τα όρια των συναρτήσεων.

Η πιο φιλόδοξη προσέγγιση στη καθολική ανάλυση, έχει στόχο να αναλύσει κάθε συνάρτηση, έχοντας μια πλήρη κατανόηση του περιβάλλοντος των συναρτήσεων που την καλούν. Μια απλή γενική ιδέα για να εφαρμοστεί μια τέτοια ανάλυση, είναι να αντικατασταθεί κάθε κλήση μιας συνάρτησης με τον ορισμό της, οπότε το αποτέλεσμα να είναι ένα ενιαίο πρόγραμμα.

Μια πιο ευέλικτη μέθοδος καθολικής ανάλυσης, είναι η χρήση ενός αλγορίθμου τοπικής ανάλυσης, ο οποίος θα χρησιμοποιηθεί για τη δημιουργία μιας σύνοψης για κάθε συνάρτηση. Όταν ένας αλγόριθμος τοπικής ανάλυσης συναντήσει μια κλήση συνάρτησης, μπορεί να ανατρέξει στη σύνοψή της. Μια σύνοψη συνάρτησης μπορεί να είναι πολύ ακριβής, οπότε και σύνθετη, ή πολύ ανακριβής, άρα και απλή.

3.4 Κανόνες

Οι κανόνες που ορίζουν τι ένα εργαλείο ασφάλειας πρέπει να αναφέρεις είναι το ίδιο σημαντικό, αν όχι περισσότερο, από τους αλγορίθμους ανάλυσης. Τα καλά εργαλεία στατικής ανάλυσης επιτρέπουν την προσθαφαίρεση και τη μεταβολή των κανόνων από το χρήστη. Έτσι είναι δυνατή η επέκταση των κανόνων και η εισαγωγή ελέγχων για νέες ευπάθειες.

Πολλά προβλήματα ασφάλειας μπορούν εκφραστούν ως προβλήματα διάδοσης μόλυνσης και η λύση αυτών των προβλημάτων απαιτούν διαφορετικών ειδών κανόνες. Ενδεικτικά οι τύποι μπορεί να είναι:

- **Κανόνες κώδικα.** Ορίζουν σημεία του προγράμματος στα οποία εισέρχονται τα μολυσμένα δεδομένα. Συναρτήσεις που ονομάζονται read() συχνά σχετίζονται, αλλά και άλλες όπως οι getenv(), getpass() ή ακόμα και η gets().

- **Κανόνες σημείων υποδοχής (sink).** Ορίζουν τα σημεία του προγράμματος, στα οποία δεν πρέπει να φτάσει η μόλυνση. Για παράδειγμα στην περίπτωση του SQL injection στη Java, η κλήση της Statement.executeQuery() είναι ένα σημείο υποδοχής.

- **Κανόνες διαπέρασης (pass-through).** Ορίζουν τον τρόπο με τον οποίο μια συνάρτηση χειρίζεται τα μολυσμένα δεδομένα. Για παράδειγμα στη Java, αν η είσοδος της συνάρτησης String.trim() είναι μολυσμένη, τότε θα είναι και η έξοδός της.

- **Κανόνες καθαρισμού (cleanse).** Είναι μια μορφή των κανόνων διαπέρασης, που αφαιρούν τη μόλυνση. Χρησιμοποιούνται για να περιγράψουν την επικύρωση της εισόδου.

- **Κανόνες σημείων εισόδου.** Είναι παρόμοιοι με τους κανόνες κώδικα, με την έννοια ότι ορίζουν σημεία εισαγωγής μόλυνσης, αλλά αντί των σημείων κλήσης μιας συνάρτησης, περιγράφουν τα σημεία τα οποία μπορεί να καλέσει ένας επιτιθέμενος. Για παράδειγμα η συνάρτηση main() στη C και η HttpServlet.doPost() στη Java, είναι σημεία εισόδου.

ΚΕΦΑΛΑΙΟ 4

4 ΠΕΡΙΠΤΩΣΕΙΣ ΧΡΗΣΗΣ

Στα πλαίσια αυτής της εργασίας, έγινε δοκιμαστική εκτέλεση τριών ελεύθερων εργαλείων στατικής ανάλυσης, σε τρεις εφαρμογές λογισμικού Java. Σε αυτήν την ενότητα παρουσιάζονται αυτά τα εργαλεία, οι εφαρμογές στις οποίες εκτελέστηκαν και τα αποτελέσματα της ανάλυσης.

4.1 Παρουσίαση των εργαλείων στατικής ανάλυσης

4.1.1 Findbugs

Η Findbugs [21] είναι μια ελεύθερη εφαρμογή στατικής ανάλυσης κώδικα γραμμένη στη γλώσσα Java. Έχει αναπτυχθεί από το πανεπιστήμιο του Maryland και αναλύει μεταγλωττισμένο κώδικα Java (bytecode), χωρίς να τον εκτελέσει. Μπορεί να εκτελεστεί ως ανεξάρτητη εφαρμογή αλλά και ως πρόσθετη σε όλα τα διαδεδομένα ολοκληρωμένα περιβάλλοντα ανάπτυξης (IDE) για εφαρμογές Java. Αναζητεί πολλών ειδών σφάλματα, με έμφαση στα λογικά προβλήματα (null pointer dereferences, bad casts, infinite recursive loops κ.α.). Η τρέχουσα έκδοση είναι η 2.0.2 με ημερομηνία μεταβολής την 8/4/2013. Συνολικά η έκδοση 2.0.2. ψάχνει για 400 πρότυπα σφαλμάτων στις εξής κατηγορίες:

- Κακή πρακτική
- Λογικά σφάλματα
- Πειραματικοί έλεγχοι
- Διεθνοποίηση
- Προσβολή από κακόβουλο κώδικα
- Προβλήματα παράλληλης επεξεργασίας
- Προβλήματα απόδοσης

- Ασφάλεια
- Αναξιόπιστος κώδικας

Στην παρούσα εργασία ασχοληθήκαμε με τα προβλήματα ασφάλειας τα οποία μπορεί να ανιχνεύσει το Findbugs. Αναλυτικά αυτά είναι:

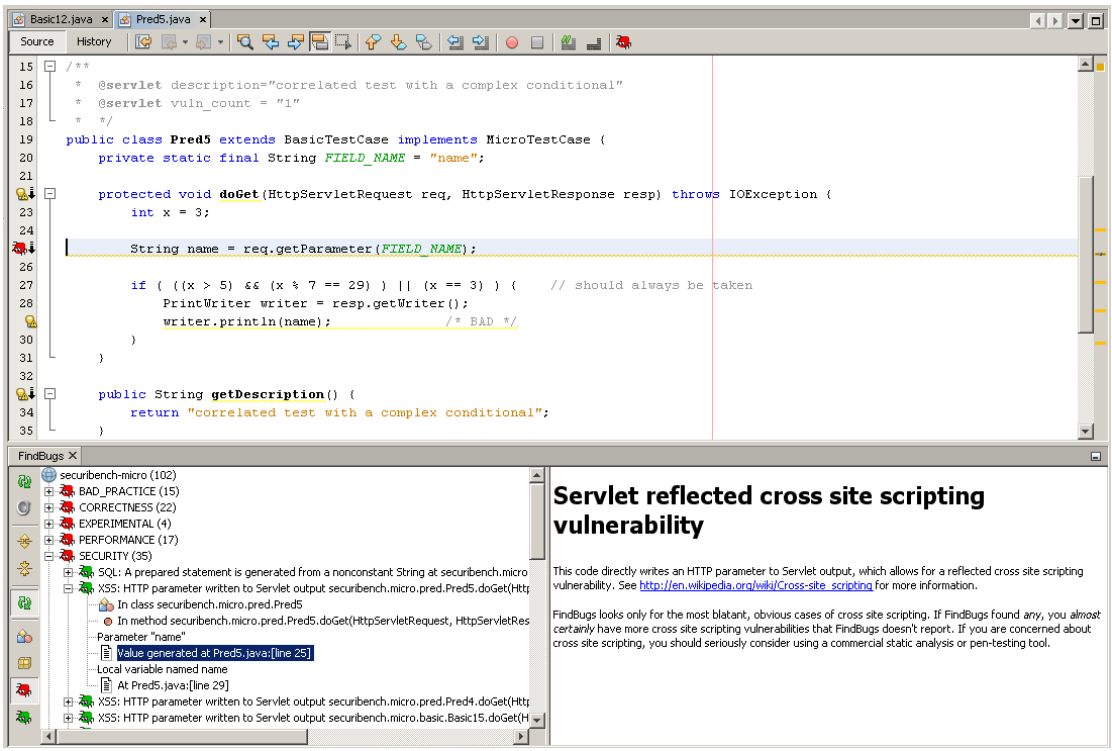
- Dm: Hardcoded constant database password
- Dm: Empty database password
- HRS: HTTP cookie formed from untrusted input
- HRS: HTTP Response splitting vulnerability
- PT: Absolute path traversal in servlet
- PT: Relative path traversal in servlet
- SQL: Nonconstant string passed to execute method on an SQL statement
- SQL: A prepared statement is generated from a nonconstant String
- XSS: JSP reflected cross site scripting vulnerability
- XSS: Servlet reflected cross site scripting vulnerability in error page
- XSS: Servlet reflected cross site scripting vulnerability

Για να βοηθήσει στην επεξεργασία των αποτελεσμάτων, κατατάσσει τα ευρήματά του σε μια κλίμακα σοβαρότητας 1-20:

- 1-4 Τρομακτικότερα
- 5-9 Τρομακτικά
- 10-14 Προβληματικά
- 15-20 Ανησυχητικά

Διεπαφή χρήστη:
Το Findbugs μπορεί να κληθεί ως αυτόνομο πρόγραμμα, αλλά παρέχει τη δυνατότητα να εγκατασταθεί ως πρόσθετο (plug-in) στα πιο διάσημα ολοκληρωμένα περιβάλλοντα ανάπτυξης (IDE) για Java. Η διεπαφή του χρήστη αποτελείται από δύο παράθυρα, όπως φαίνεται στην επόμενη εικόνα, ένα με τη λίστα των προειδοποιήσεων και ένα με μια σύντομη περιγραφή για την επιλεγμένη προειδοποίηση. Η λίστα των προειδοποιήσεων μπορεί να ταξινομηθεί για ευκολία, ανάλογα με το είδος του σφάλματος ή το

αρχείο στο οποίο περιέχεται. Επίσης δίνεται η δυνατότητα μετάβασης στην προβληματική γραμμή κώδικα, με ένα κλικ πάνω στην προειδοποίηση.



Εικόνα 8: Διεπαφή χρήστη του εργαλείου Findbugs

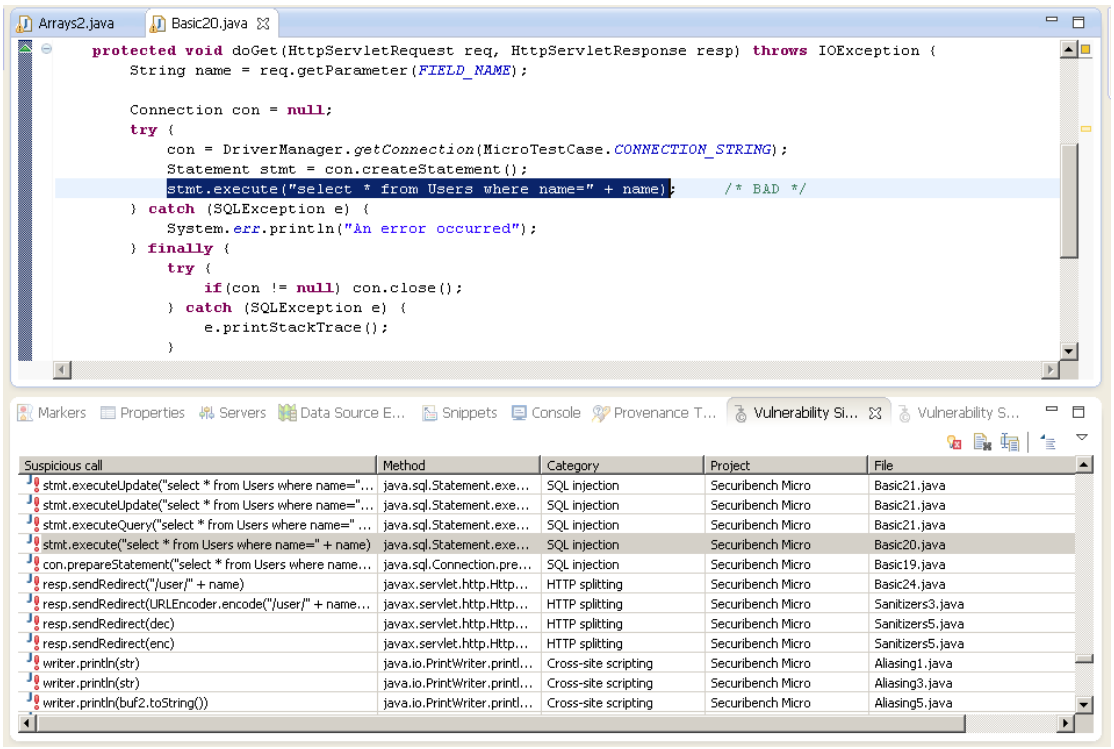
4.1.2 Lightweight Analysis for Program Security in Eclipse (LAPSE)

Η εφαρμογή LAPSE [22] είναι έργο της κοινότητας του Open Web Application Security Project (OWASP) για την εύρεση ευπαθειών σε εφαρμογές Java. Έχει αναπτυχθεί από τον Benjamin Livshits και είναι μέρος του Griffin Software Security Project [23]. Η τελευταία έκδοση είναι η 2.5.6 με ημερομηνία 14/9/2006 αλλά το έργο επανεκκινήθηκε ως Lapse+ από τον Pablo Martin Perez. Οι κατηγορίες ευπαθειών που μπορεί να ανιχνεύσει το LAPSE είναι οι:

- Parameter manipulation
- Header manipulation
- Cookie poisoning
- Command-line parameters
- SQL injections
- Cross-site scripting

- HTTP splitting
- Path traversal

Διεπαφή χρήστη:
Το LAPSE λειτουργεί ως πρόσθετο στο Eclipse, το οποίο είναι το διασημότερο ολοκληρωμένο περιβάλλον ανάπτυξης Java και η διεπαφή του περιλαμβάνει τρία παράθυρα, τα οποία παρουσιάζουν τις σημεία εισόδου της μόλυνσης, τα σημεία υποδοχής της μόλυνσης και στο τρίτο προσπαθεί να γίνει ένας έλεγχος, αν υπάρχει σύνδεση μεταξύ των σημείων εισόδου με τα σημεία υποδοχής. Στην έκδοση που δοκιμάσαμε, αυτή η λειτουργία παρουσίαζε σφάλμα.



Εικόνα 9: Διεπαφή χρήστη του εργαλείου LAPSE

4.1.3 Yet Another Source Code Analyzer (YASCA)

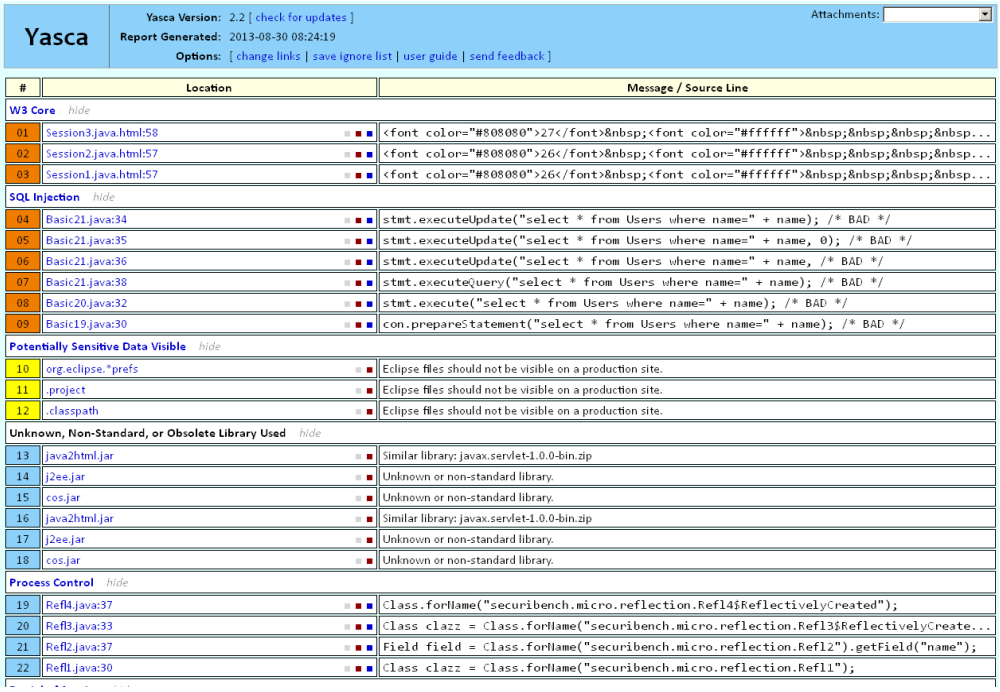
Η εφαρμογή YASCA [24] αναπτύχθηκε από τον Michael V. Scovetta για την εύρεση ευπαθειών ασφάλειας, προβλήματα ποιότητας, απόδοσης και κακών πρακτικών στον πηγαίο κώδικα. Εκτός από τα πρότυπα σφαλμάτων που περιέχει η ίδια η εφαρμογή, μπορεί επίσης να ενσωματώσει αποτελέσματα από άλλες, όπως οι: FindBugs, PMD,

JLint, JavaScript Lint, PHPLint, Cppcheck, ClamAV, Pixy, και RATS. Η τρέχουσα έκδοση είναι η 2.21 με ημερομηνία μεταβολής την 1/11/2010. Η YASCA περιέχει τουλάχιστον έναν έλεγχο για τις παρακάτω γλώσσες προγραμματισμού:

- .NET (VB.NET, C#, ASP.NET)
- C/C++
- COBOL
- ColdFusion
- Raw HTTP Traffic, HTML, CSS, JavaScript
- Java
- Perl
- PHP
- Python

Διεπαφή χρήστη:

Το YASCA εκτελείται από τη γραμμή εντολών και όταν ολοκληρωθεί η εκτέλεση, παράγεται μια αναφορά σε μορφή αρχείου HTML. Τα ευρήματα είναι ταξινομημένα με την κρισιμότητά τους και ο χρήστης ενημερώνεται για κάθε προβληματική γραμμή και μια σύντομη περιγραφή του προβλήματος.



Yasca

Yasca Version: 2.2 [check for updates]
Report Generated: 2013-08-30 08:24:19
Options: [change links | save ignore list | user guide | send feedback]
Attachments:

#	Location	Message / Source Line
W3 Core hide		
01	Session3.java.html:58	27 ...
02	Session2.java.html:57	26 ...
03	Session1.java.html:57	26 <font color="#ffffff"> ...
SQL Injection hide		
04	Basic21.java:34	stmt.executeUpdate("select * from Users where name=" + name); /* BAD */
05	Basic21.java:35	stmt.executeUpdate("select * from Users where name=" + name, 0); /* BAD */
06	Basic21.java:36	stmt.executeUpdate("select * from Users where name=" + name, /* BAD */
07	Basic21.java:38	stmt.executeQuery("select * from Users where name=" + name); /* BAD */
08	Basic20.java:32	stmt.execute("select * from Users where name=" + name); /* BAD */
09	Basic19.java:30	con.prepareStatement("select * from Users where name=" + name); /* BAD */
Potentially Sensitive Data Visible hide		
10	org.eclipse.*prefs	Eclipse files should not be visible on a production site.
11	.project	Eclipse files should not be visible on a production site.
12	.classpath	Eclipse files should not be visible on a production site.
Unknown, Non-Standard, or Obsolete Library Used hide		
13	java2html.jar	Similar library: javax.servlet-1.0.0-bin.zip
14	j2ee.jar	Unknown or non-standard library.
15	cos.jar	Unknown or non-standard library.
16	java2html.jar	Similar library: javax.servlet-1.0.0-bin.zip
17	j2ee.jar	Unknown or non-standard library.
18	cos.jar	Unknown or non-standard library.
Process Control hide		
19	Refl4.java:37	Class.forName("securibench.micro.reflection.Refl4$ReflectivelyCreated");
20	Refl3.java:33	Class clazz = Class.forName("securibench.micro.reflection.Refl3$ReflectivelyCreate...
21	Refl2.java:37	Field field = Class.forName("securibench.micro.reflection.Refl2").getField("name");
22	Refl1.java:30	Class clazz = Class.forName("securibench.micro.reflection.Refl1");

Εικόνα 10: Αναφορά αποτελεσμάτων ανάλυσης του εργαλείου YASCA

4.2 Αποτελέσματα εφαρμογής των εργαλείων στατικής ανάλυσης

4.2.1 Securibench micro

Η εφαρμογή Securibench micro [25] έχει αναπτυχθεί από το πανεπιστήμιο του Stanford, ως μέρος του Griffin Software Security Project, για την αξιολόγηση των εργαλείων στατικής ανάλυσης. Περιέχει ένα πλήθος από τεχνητά προβλήματα ασφάλειας όπως τα:

- SQL injection
- Cross-site scripting
- HTTP splitting
- Path traversal
- Unsafe redirect

Η έκδοση της εφαρμογής που αναλύθηκε είναι η 1.06 με ημερομηνία 21/4/2006.

Αποτελέσματα ανάλυσης

Και τα τρία εργαλεία βρήκαν τα σημαντικότερα προβλήματα SQL injection και δύο από τα τρία βρήκαν αρκετά σφάλματα XSS. Συγκεντρωτικά τα αποτελέσματα είναι:

Σφάλματα	**Findbugs**		**LAPSE**		**YASCA**	
	ΑΘ	**ΨΘ**	**ΑΘ**	**ΨΘ**	**ΑΘ**	**ΨΘ**
SQL injection	3	0	5	5	6	6
XSS	27	5	105	43	0	0
HTTP Response splitting	0	0	0	2	0	0
Unsafe redirect	0	0	2	0	0	0
Potentially Sensitive Data Visible	0	0	0	0	0	3
Denial of service	0	0	0	0	0	1
ΣΥΝΟΛΟ:	**30**	**5**	**112**	**50**	**6**	**10**

Όπως φαίνεται παραπάνω η αναλογία αληθώς θετικών προς ψευδώς θετικών είναι πολύ καλή και μόνο το YASCA ανέφερε λιγότερα αληθώς από ψευδώς θετικά.

Αληθώς θετικά

Ενδεικτικά, δύο απλές περιπτώσεις SQL injection που βρέθηκαν είναι, στη doGet() του Basic19.java:

```
String name = req.getParameter(FIELD_NAME);
Connection con = null;
try {
 con =
DriverManager.getConnection(MicroTestCase.CONNECTI
ON_STRING);
 con.prepareStatement("select * from Users where
name=" + name); /* BAD */
```

και στη doGet() του Basic20.java:

```
String name = req.getParameter(FIELD_NAME);
Connection con = null;
try {
      con =
DriverManager.getConnection(MicroTestCase.CONNECTI
ON_STRING);
      Statement stmt = con.createStatement();
      stmt.execute("select * from Users where
name=" + name);     /* BAD */
```

Μια περίπτωση XSS στη doGet() του Basic12.java:

```
String s1 = req.getParameter("name");
```

```
PrintWriter writer = resp.getWriter();
boolean choice = new Random().nextBoolean();
if(choice) {
      writer.println(s1 + ":");   /* BAD */
} else{
      writer.println(s1 + ";");   /* BAD */
}
```

Και μια περίπτωση unsafe redirect στη doGet() του Basic24.java:

```
String s = req.getParameter(FIELD_NAME);
String name = s.toLowerCase(Locale.UK);
resp.sendRedirect("/user/" + name);          /*
BAD */
```

Είναι ενδιαφέρον ότι το προηγούμενο πρόβλημα το βρήκε μόνο το LAPSE και αυτό όμως το ανέφερε ως πρόβλημα Http response splitting.

Ψευδώς θετικά

Δύο ενδιαφέρουσες περιπτώσεις ψευδώς θετικών αναφορών για προβλήματα XSS είναι στη doGet() του StriongUpdates.java:

```
String name = req.getParameter(FIELD_NAME);
Widget w = new Widget();
w.value = name;
w.value = "abc";
PrintWriter writer = resp.getWriter();
writer.println(w.value);      /* OK */
```

Και στη doGet() του Pred3.java:

```
boolean choice = new Random().nextBoolean();
String name = "abc";
if(choice) {
    name = req.getParameter(FIELD_NAME);
}
if(!choice) {
    PrintWriter writer = resp.getWriter();
    writer.println(name);      /* OK */
}
```

Τα παραπάνω παραδείγματα, αναφέρθηκαν και από το Findbugs και από το LAPSE. Το YASCA, δεν ανέφερε κανένα πρόβλημα (ορθώς ή ψευδώς) της κατηγορίας XSS. Από τον κώδικα αυτών των παραδειγμάτων, φαίνεται ότι και τα δύο εργαλεία δεν είναι σε θέση να κάνουν ανάλυση σε βάθος. Στην πρώτη περίπτωση, το πεδίο w.value παίρνει αρχική τιμή, η οποία μπορεί να μολυνθεί από το χρήστη, αλλά πριν γραφτεί στην έξοδο, η τιμή της αντικαθίσταται από μία σταθερά (“abc”). Τα εργαλεία στατικής ανάλυσης όμως, ανέφεραν το πρόβλημα. Αυτό δείχνει ότι δεν είναι σε θέση να ακολουθήσουν τη ροή δεδομένων αυτού του απλού προγράμματος. Στο δεύτερο παράδειγμα ψευδούς θετικού, υπάρχουν δύο περιπτώσεις κατά την εκτέλεση. Στην πρώτη περίπτωση (choice = true), η μεταβλητή name μπορεί να μολυνθεί από το χρήστη, αλλά δεν θα γραφτεί στην έξοδο. Άρα δεν υπάρχει ευπάθεια. Στη δεύτερη περίπτωση (choice = false), η μεταβλητή name θα γραφτεί στην έξοδο, αλλά τότε δεν θα μολυνθεί, γιατί θα περιέχει τη σταθερά “abc”. Αυτό το παράδειγμα δείχνει ότι και τα δύο εργαλεία, δεν μπορούν να αναλύσουν τη ροή ελέγχου αυτή της μεθόδου.

Ψευδώς αρνητικά

Αν και τουλάχιστον τα Findbugs και LAPSE διαφημίζουν ότι ανακαλύπτουν τα σοβαρά προβλήματα της κατηγορίας path traversal, κανένα δεν ανέφερε ούτε την παρακάτω απλή περίπτωση, από τη doGet() του Basic22.java:

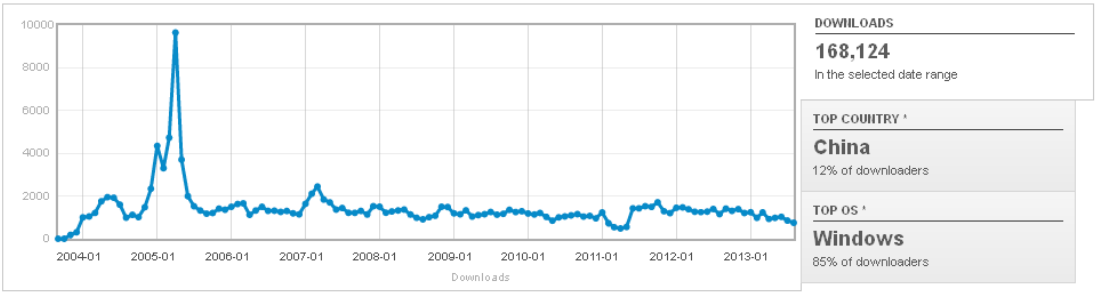
```
String s = req.getParameter(FIELD_NAME);
String name = s.toLowerCase(Locale.UK);
File f = new File(name);
f.createNewFile();      /* BAD */
```

4.2.2 Hipergate

Η Hipergate [26] είναι μια ελεύθερη σουίτα διαχείρισης πελατειακών σχέσεων (CRM) και συνεργασίας (groupware) γραμμένη στη γλώσσα Java. Έχει μια μεγάλη βάση κώδικα και αποτελείται από τις παρακάτω μονάδες:

- Διαχείριση επαφών
- Εταιρική επικοινωνία
- Εργαλεία συνεργασίας
- Διαχείριση έργου
- Παραγγελίες και τιμολόγηση
- Διαχείριση εκπαιδεύσεων

Το sourceforge.net αναφέρει ότι έχουν κατεβάσει την εφαρμογή σχεδόν 170.000 φορές τα τελευταία χρόνια:



Η έκδοση που αναλύθηκε είναι η 7.0 με ημερομηνία μεταβολής την 28/5/2013.

Αποτελέσματα ανάλυσης

Ο ακόλουθος πίνακας περιέχει τα προβλήματα που ανέφεραν τα τρία εργαλεία στατικής ανάλυσης:

Σφάλματα	**Findbugs**		**LAPSE**		**YASCA**	
	ΑΘ	**ΨΘ**	**ΑΘ**	**ΨΘ**	**ΑΘ**	**ΨΘ**
SQL injection	1	128	1	973	1	1024
XSS	7	0	2	134	0	13
HTTP Response splitting	2	0	1	5	0	0
Command injection	0	0	0	7	0	0
Authentication: Weak Credentials	0	0	0	0	0	4
Authentication: Simple/Guessable Password	0	0	0	0	0	4
Cryptography	0	0	0	0	2	11
Potentially Sensitive Data Visible	0	0	0	0	0	4
Potentially Logging Sensitive Information	0	0	0	0	0	5
Authentication: Password Stored in Cleartext	0	0	0	0	0	7
Weak Cryptographic Algorithm: Math.random()	0	0	0	0	0	2
Denial of service	0	0	0	0	0	34
ΣΥΝΟΛΟ:	**10**	**128**	**4**	**1119**	**3**	**1108**

Τα συνολικά αποτελέσματα, είναι απογοητευτικά. Το πλήθος των ψευδώς θετικών στις περιπτώσεις των LAPSE και YASCA, είναι μια τάξη μεγαλύτερο από το πλήθος των αληθώς θετικών. Το Findbugs ανέφερε 10 φορές περισσότερα ψευδώς θετικά, από αληθώς θετικά. Το μεγάλο πρόβλημα παρουσιάζεται στην αναφορά σφαλμάτων SQL injection. Αυτό συνέβη γιατί το Hipergate έχει μεγάλο σώμα κώδικα και οι προγραμματιστές χρησιμοποιούν κατά κόρον τη συνένωση συμβολοσειρών για τη δημιουργία ερωτήματων SQL. Αυτή δεν είναι μια γενικά καλή πρακτική, αλλά αυτό δεν σημαίνει ότι κάθε τέτοια παραγωγή ερωτήματος, θα οδηγήσει σε ευπάθεια. Αντίθετα, από την επισκόπηση του κώδικα, μόνο μία περίπτωση βρέθηκε εκμεταλλεύσιμη. Επίσης το YASCA ανέφερε ένα σύνολο από διαφορετικού είδους προβλήματα, τα οποία όμως δεν επιβεβαιώθηκαν από την επισκόπηση. Ενδιαφέρουσα εξαίρεση, είναι η αναφορά του για ένα πρόβλημα αλγορίθμου, η οποία θα παρουσιαστεί παρακάτω.

Αληθώς θετικά

Το μοναδικό εκμεταλλεύσιμο SQL injection που ανακάλυψαν και τα τρία εργαλεία, είναι στη doGet() του HttpBLOBServlet.java (γραμμές 179-193):

```
sSQL = "SELECT " +
request.getParameter("nm_field") + "," +
request.getParameter("bin_field") + " FROM " +
request.getParameter("nm_table") + sSQL;
...
oStmt = oConn.prepareStatement (sSQL);
...
oRSet = oStmt.executeQuery();
```

Στον παραπάνω κώδικα γίνεται απευθείας χρήση των παραμέτρων της αίτησης Http στην κατασκευή του ερωτήματος SQL. Είναι η πιο κλασική περίπτωση αυτής της ευπάθειας και η παρουσία της δείχνει ότι ο προγραμματιστής αγνοεί πλήρως αυτού του είδους τα προβλήματα ασφάλειας.

Ένα χαρακτηριστικό πρόβλημα XSS, βρέθηκε στη μέθοδο doGet() του HttpCalendarServlet.java:

```
String sCmd = request.getParameter("command");
...
sMeet = request.getParameter("meeting");
...
oBuf.append("<id>"+sMeet+"</id>");
...
response.getWriter().print(oBuf.toString());
...
} else {
```

```
response.sendError(HttpServletResponse.SC_BAD_REQU
EST, "Invalid command "+sCmd);
```

Καταρχήν υπάρχει XSS γιατί γράφεται στην Http απάντηση, μέσω της sMeet, η τιμή της Http παραμέτρου “meeting”. Μια άλλη κλασική περίπτωση όμως είναι η τελευταία γραμμή. Αν κάτι δεν πάει καλά, το servlet απαντάει με ένα λάθος, το οποίο όμως περιέχει την Http παράμετρο “command”. Από τη σκοπιά του προγραμματιστή αυτό είναι καλό, γιατί διευκολύνει την αποσφαλμάτωση, αλλά έτσι δίνεται ένα εργαλείο στον επιτιθέμενο.

Στην doGet() του HttpVCardServlet.java, ο κώδικας διαβάζει ένα τμήμα του Http header (μεταβλητή sNick) της απάντησης, από την παράμετρο “pk_value” της αίτησης. Αυτή η ροή είναι μια ευπάθεια Http response splitting:

```
sPKVal = (request.getParameter("pk_value")!=null ?
request.getParameter("pk_value") : "null");
...
if (!oFlw.isNull("tx_nickname"))
          sNick = oFlw.getString("tx_nickname");
        else
          sNick = sPKVal;
...
response.setHeader("Content-Disposition",
"attachment; filename=\"" + sNick + ".vcf\"");
```

Ένα άλλο αληθώς θετικό, το οποίο ανακάλυψε μόνο το εργαλείο YASCA, είναι η χρήση σε κάποια σημεία του κώδικα, επικίνδυνων αλγορίθμων ασφάλειας (MD4, MD5, RC4).

Ψευδώς θετικά

Από τα πάρα πολλά ψευδώς θετικά SQL injection, θα παρουσιάσουμε μόνο ένα χαρακτηριστικό παράδειγμα. Στη getIdFromName() του EdcationDegree.java υπάρχει η εντολή:

```
oStmt = oConn.prepareStatement("SELECT
"+DB.gu_degree+" FROM "+DB.k_education_degree+"
WHERE "+DB.gu_workarea+"=? AND
"+DB.nm_degree+"=?");
```

Εδώ γίνεται συνένωση συμβολοσειρών για την παραγωγή του ερωτήματος, αλλά όλες οι συμβολοσειρές είναι σταθερές. Αυτό το αναγνώρισε μόνο το Findbugs, ενώ τα LAPSE και YASCA ανέφεραν πρόβλημα.

4.2.3 Πραγματική εμπορική εφαρμογή

Σε αυτήν την ενότητα, θα παρουσιάσουμε τα αποτελέσματα της εκτέλεσης των εργαλείων σε μια ιδιωτική εμπορική εφαρμογή λογισμικού. Είναι μια μικρού μεγέθους web εφαρμογή φτιαγμένη σε Java. Αναπτύχθηκε από το τμήμα IT μιας εταιρείας, για να χρησιμοποιηθεί από τους πελάτες της. Για λόγους εμπιστευτικότητας, δεν είναι δυνατό να αναφερθούν περισσότερα για την εφαρμογή, αλλά οι σχετικές πληροφορίες είναι διαθέσιμες στον επιβλέποντα. Επίσης στα παραδείγματα που θα παρουσιαστούν, έχουν γίνει κάποιες αλλαγές στα ονόματα των αναγνωριστικών.

Αποτελέσματα ανάλυσης

Ο συγκεντρωτικός πίνακας των αποτελεσμάτων της ανάλυσης είναι:

Σφάλματα	**Findbugs**		**LAPSE**		**YASCA**	
	ΑΘ	**ΨΘ**	**ΑΘ**	**ΨΘ**	**ΑΘ**	**ΨΘ**
SQL injection	2	22	2	26	2	31
XSS	2	0	2	17	16	16
Authentication: Hardcoded credentials	0	0	0	0	1	0
Use of a Hidden Form Field	0	0	0	0	0	30
ΣΥΝΟΛΟ:	**4**	**22**	**4**	**43**	**19**	**77**

Και σε αυτήν την περίπτωση, τα προβλήματα που βρέθηκαν είναι τύπου SQL injection και XSS. Επίσης βρέθηκε ένα άλλο σημαντικό πρόβλημα, τα ενσωματωμένα διαπιστευτήρια, μόνο όμως από το εργαλείο YASCA.

Αληθώς θετικά

Ο ακόλουθος κώδικας είναι από μια μέθοδο με παραμέτρους from και to:

```
String str = "SELECT * FROM table1 WHERE c01>='" +
from + "' AND c01<='" + to + "'  ORDER BY c01
DESC";
...
ResultSet results = stmt.executeQuery(str);
```

Η επισκόπηση του κώδικα έδειξε ότι αυτή η μέθοδος καλείται από ένα servlet και οι παράμετροι της μεθόδου (from και to), παίρνουν τιμές από αντίστοιχες Http παραμέτρους. Οπότε υπάρχει μια SQL injection ευπάθεια.

Σε μια JSP σελίδα, βρέθηκε ο παρακάτω κώδικας:

```
<%
...
 String D1=request.getParameter("D1");
```

```
...
%>
...
<%  if (D1!=null){%><option value="<%=D1%>"><
%=D1%></option><%}%>
```

Ο στόχος του προγραμματιστή εδώ, είναι να εμφανίσει μια λίστα επιλογών, με προεπιλεγμένη μια τιμή που λαμβάνεται ως παράμετρος. Το αποτέλεσμα είναι μια κλασική ευπάθεια XSS.

Τέλος, το πολύ σημαντικό πρόβλημα των ενσωματωμένων διαπιστευτηρίων, το οποίο αναφέρθηκε μόνο από το YASCA, υπήρχε σε μια γραμμή της παρακάτω μορφής:

```
super("com.microsoft.jdbc.sqlserver.SQLServerDrive
r",
"jdbc:microsoft:sqlserver://192.168.1.1:1433;User=
user1;Password=kodikos;DatabaseName=db1);
```

Το πρόγραμμα έκανε τη σύνδεση στη βάση δεδομένων, με ενσωματωμένα διαπιστευτήρια.

Ψευδώς θετικά

Όπως φαίνεται στο συγκεντρωτικό πίνακα, αναφέρθηκαν αρκετά σφάλματα SQL injection. Σε αρκετές μεθόδους υπήρχε η δημιουργία ερωτημάτων SQL, συνενώνοντας σταθερές συμβολοσειρές με παραμέτρους. Από την επισκόπηση κώδικα όμως προέκυψε ότι δεν υπήρχε ροή δεδομένων από το χρήστη, σε αυτές τις μεθόδους. Και αυτή η περίπτωση έδειξε ότι εργαλεία που δοκιμάστηκαν, αδυνατούν να εφαρμόσουν τέτοια ανάλυση.

Το YASCA ανέφερε επίσης ότι η εφαρμογή χρησιμοποιούσε κρυφά πεδία σε web φόρμες. Αυτό δεν είναι καλή πρακτική, αλλά η

επισκόπηση κώδικα έδειξε ότι δεν υπάρχει κανένα πραγματικό πρόβλημα ασφάλειας.

4.3 Αξιολόγηση αποτελεσμάτων και γενικά συμπεράσματα

Από τα αποτελέσματα της χρήσης των εργαλείων στατικής ανάλυσης στη δοκιμαστική και τις πραγματικές εφαρμογές, προκύπτει ότι τα συγκεκριμένα ελεύθερα εργαλεία στατικής ανάλυσης, δεν έχουν τη δυνατότητα να αναλύσουν τον κώδικα σε βάθος. Δεν είναι (ακόμα) ικανά να ιχνηλατήσουν τις ροές ελέγχου ή δεδομένων.

Αυτό έχει δύο σοβαρές επιπτώσεις. Καταρχήν υπάρχει ο κίνδυνος να μη βρουν κάποια σοβαρή ευπάθεια. Στα παραδείγματα παραπάνω κανένα εργαλείο μια πολύ απλή περίπτωση path traversal, ενώ διαφημίζουν ότι ψάχνουν τέτοια προβλήματα και μια απλή περίπτωση ενσωματωμένων διαπιστευτηρίων ανακαλύφθηκε μόνο από το YASCA. Αυτό μπορεί επίσης να δημιουργήσει μια ψευδή αίσθηση ασφάλειας στον χρήστη των εργαλείων. Η άλλη επίπτωση της μη εις βάθος ανάλυσης, είναι το μεγάλο πλήθος των ψευδώς θετικών. Τα οποία μπορούν μεν να προσφέρουν μια ιδέα των κακών πρακτικών που χρησιμοποιούνται μέσα στον κώδικα, αλλά δυσχεραίνουν πολύ την αναζήτηση πραγματικών ευπαθειών. Συνεπώς, υπάρχει περιθώριο η κοινότητα να βελτιώσει τα εργαλεία αυτά ή να δημιουργήσει νέα και ένας οργανισμός που ρίχνει μεγάλο βάρος στην ασφάλεια λογισμικού και θέλει να εφαρμόσει τη στατική ανάλυση, πρέπει να εξετάσει και τις επιλογές των εμπορικών εργαλείων.

Παρά τις περιορισμένες δυνατότητες των εργαλείων που δοκιμάστηκαν, κατάφεραν να ανακαλύψουν πολύ σοβαρά προβλήματα. Η ευπάθεια SQL injection που βρέθηκε στην εμπορική εφαρμογή, απειλούσε ευαίσθητες πληροφορίες στη βάση δεδομένων. Επίσης τα εργαλεία δείχνουν ότι επιβεβαιώνεται η γενική ιδέα ότι τα κλασικά σφάλματα ασφάλειας (SQL injection, XSS κ.λπ.), είναι κοινά. Οι προγραμματιστές που δεν κάποια ενημέρωση για τα σφάλματα ασφάλειας, επιλέγουν τους ίδιους προβληματικούς

τρόπους να λύσουν κλασικά προγραμματιστικά προβλήματα. Γι' αυτό το λόγο κι επειδή τα εργαλεία που δοκιμάστηκαν εκτός από το ότι βρήκαν κάποια από αυτά τα προβλήματα, παρέχουν και μια ενημέρωση γι' αυτά, κρίνονται πολύ χρήσιμα. Ειδικά ένα εργαλείο σαν το Findbugs που εκτελεί και ελέγχους για άλλα σφάλματα (λογικά, κακές πρακτικές κ.λπ.) μπορεί να βοηθήσει οποιονδήποτε προγραμματιστή.

ΒΙΒΛΙΟΓΡΑΦΙΚΕΣ ΑΝΑΦΟΡΕΣ

[1] KarvoRice, David. Geekonomics The Real Cost of Insecure Software. New York: Addison-Wesley Professional, 2007. Print.

[2] Viega, John, and Gary McGraw. Building Secure Software How to Avoid Security Problems the Right Way. New York: Addison-Wesley Professional, 2001. Print.

[3] Computer Security Training, Network Research & Resources. <http://www.sans.org>

[4] McGraw, Gary. Software security : building security in. Upper Saddle River, NJ: Addison-Wesley, 2006. Print.

[5] M. Bishop. Computer Security: Art and Science. Addison-Wesley, December 2002. Print.

[6] Tsipenyuk, Katrina, Brian Chess, Gary McGraw. “Seven Pernicious Kingdoms: A Taxonomy of Software Secu-rity Errors.” Proceedings of the NIST Workshop on Software Security Assurance Tools, Techniques, and Metrics (SSATTM) (Long Beach, CA, 2005), 36–43.

[7] 2011 CWE/SANS Top 25 Most Dangerous Software Errors <http://cwe.mitre.org/top25/>

[8] SANS Institute, Twenty Critical Security Controls for Effective Cyber Defense <http://www.sans.org/top20>

[9] MITRE Corporation, Common Weakness Enumeration (CWE) <http://cwe.mitre.org/>

[10] OWASP, Top 10 2013, <https://www.owasp.org/index.php/Top_10_2013-Top_10>

[11] Howard, Michael. 24 deadly sins of software security programming flaws and how to fix them. San Francisco: McGraw-Hill Professional, 2009. Print.

[12] 2011 CWE/SANS Top 25: Monster Mitigations <http://cwe.mitre.org/top25/mitigations.html>

[13] Chess, Brian, and Jacob West. Secure Programming with Static Analysis. Upper Saddle River, NJ: Addison-Wesley, 2007. Print.

[14] Viega, J., J. Bloch, T. Kohno, G. McGraw. “ITS4: A Sta-tic Vulnerability Scanner for C and C++ Code.” The 16th Annual Com-puter Security Applications Conference (ACSAC’00)(New Orleans, LA, 11–15 December 2000), 257–267.

[15] Secure Software, Inc. “RATS—Rough Auditing Tool for Secu-rity.” 2001. <http://www.securesoftware.com/>.

[16] Wheeler, D. A. “FlawFinder.” 2001. <http://www.dwheeler.com/flawfinder/>.

[17] Alan Turing. "On computable numbers, with an application to the Entscheidungsproblem". In Proceedings of the London Mathematical Society, Series 2, Volume 42, pages 230-265, 1936.

[18] Rice, H. G. "Classes of Recursively Enumerable Sets and Their Decision Problems". In Transactions of the American Mathematical Society, 74, pages 358-366, 1953.

[19] Cousot, Patrick and Cousot, Radhia. “Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints”. In Conference Record of the Sixth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, pages 238—252, Los Angeles, California, 1977. ACM Press, New York.

[20] Dijkstra, E. W. A Discipline of Programming. Englewood Cliffs, NJ: Prentice Hall, 1976.

[21] FindBugs™ - Find Bugs in Java Programs. <http://findbugs.sourceforge.net/>

[22] LAPSE: Web Application Security Scanner for Java. <http://suif.stanford.edu/~livshits/work/lapse/>

[23] Griffin Software Security Project. <http://suif.stanford.edu/~livshits/work/griffin/>

[24] Yasca source code analysis tool. <http://www.scovetta.com/yasca.html>

[25] Stanford SecuriBench Micro. <http://suif.stanford.edu/~livshits/work/securibench-micro/>

[26] Hipergate open source CRM. http://www.hipergate.org/portal/en/index.html