Κεφάλαιο 4. Προγραμματισμός στο Διαδίκτυο

Σύνοψη
Η ανάπτυξη αξιόπιστων διαδικτυακών εφαρμογών που θα λειτουργούν διασφαλίζοντας την ικανοποίηση των βασικών ιδιοτήτων ασφάλειας, είναι ένα ζήτημα που απασχολεί τους ειδικούς του χώρου της Τεχνολογίας Λογισμικού και της Ασφάλειας Πληροφοριών. Ο κώδικας, που γράφεται σήμερα, δίνει στις εφαρμογές αυτές ικανότητες δικτυακής σύνδεσης, έτσι ώστε να εκμεταλλεύονται οι χρήστες τους τις δυνατότητες του Διαδικτύου και να χρησιμοποιούν απομακρυσμένες υπηρεσίες. Η αρχιτεκτονική που ακολουθείται, συνήθως, είναι αυτή του πελάτη/εξυπηρετητή, αν και τα τελευταία χρόνια εμφανίζει ιδιαίτερη άνοδο αυτή των ομότιμων κόμβων (peep-to-peer). Όμως, τα περιστατικά ασφάλειας που σχετίζονται με αδυναμίες των διαδικτυακών εφαρμογών είναι δυστυχώς αρκετά συχνά. Για αυτό, καταβάλλεται μια συντονισμένη προσπάθεια ώστε τα περιστατικά αυτά να καταγράφονται συστηματικά για να συσσωρεύεται γνώση και να λαμβάνονται μέτρα που θα βοηθούν τις επόμενες γενιές προγραμματιστών ώστε να αποφεύγουν τα ίδια προγραμματιστικά λάθη και παραλείψεις κατά την ανάπτυξη των διαδικτυακών εφαρμογών.

Προαπαιτούμενη γνώση
Για την κατανόηση του παρόντος κεφαλαίου, απαιτείται γνώση των βασικών εννοιών και ζητημάτων ασφάλειας (Κεφ. 1).

4.1 Εισαγωγή

Στο χώρο της Ασφάλειας Πληροφοριών, η παρατήρηση ότι δεν υπάρχει το ασφαλές πληροφοριακό σύστημα αποτελεί βασική αρχή, καθώς τα πληροφοριακά συστήματα αλλάζουν διαρκώς. Αυτό δε σημαίνει ότι πρέπει να δεχτούμε αυτή τη διαπίστωση μοιρολατρικά και να παραιτηθούμε από κάθε προσπάθεια προστασίας του πληροφοριακού μας συστήματος. Αντιθέτως, υπάρχουν οδηγίες και κατευθυντήριες γραμμές, τις οποίες οφείλουμε να ακολουθήσουμε προκειμένου να επιτύχουμε τον υψηλότερο βαθμό ασφάλειας, ανάλογα με τα μέσα που διαθέτουμε.

Μια διαδικτυακή εφαρμογή (Web application) είναι εκτεθειμένη σε περισσότερους κινδύνους από ότι μια εφαρμογή που είναι εγκατεστημένη και λειτουργεί τοπικά στο δικό μας υπολογιστή ή εντός ενός τοπικού δικτύου. Οι τεχνολογίες διασύνδεσης τις οποίες χρησιμοποιούμε ώστε να καταστήσουμε την εφαρμογή μας περισσότερο λειτουργική, δίνουν ταυτόχρονα τη δυνατότητα σε κακόβουλους χρήστες να έρθουν σε άμεση επικοινωνία μαζί της και έτσι μπορούν να εκμεταλλευτούν τις τυχόν ευπάθειες, οι οποίες προέκυψαν από προγραμματιστικά σφάλματα και παραλείψεις.

Προφανώς, οι προγραμματιστές δεν έχουν σκοπό να γράψουν κώδικα που να περιέχει ευπάθειες. Ωστόσο, αυτό συμβαίνει για διάφορους λόγους. Οι κυριότεροι λόγοι για τους οποίους οι προγραμματιστές καταλήγουν στο να γράψουν κώδικα που περιέχει ή προκαλεί ευπάθειες στη διαδικτυακή εφαρμογή, είναι:

- Η ασφάλεια δεν αποτελεί συνήθως σημαντική προτεραιότητα των προγραμματιστών και μόλις τα τελευταία χρόνια έχει καταστεί σημαντικό χαρακτηριστικό των εφαρμογών. Επιπλέον, οι παλαιότεροι προγραμματιστές είναι αρκετά πιθανό να μην έχουν διδαχθεί τεχνικές ασφαλούς προγραμματισμού.

- Μερικές γλώσσες προγραμματισμού (όπως για παράδειγμα η C) διαθέτουν εγγενείς ιδιότητες που οδηγούν στην ανάπτυξη ευάλωτου κώδικα, όπως για παράδειγμα οι άμεσες αναφορές και εγγραφές στη μνήμη σε δομές δεδομένων, όπως η στοίβα (stack), ή ο σωρός (heap).

- Η ασφάλεια, όντας κατά κανόνα μη λειτουργική απαίτηση, απαιτεί επιπρόσθετο φόρτο εργασίας, με αποτέλεσμα αρκετοί προγραμματιστές να ικανοποιούν μόνο τα απαραίτητα λειτουργικά χαρακτηριστικά της εφαρμογής που αναπτύσσουν.

- Μόλις τα τελευταία χρόνια ξεκίνησε η ευαισθητοποίηση των τελικών χρηστών σε θέματα ασφάλειας, ώστε να επιζητούν την ύπαρξη χαρακτηριστικών ασφάλειας προκειμένου να χρησιμοποιήσουν μια διαδικτυακή εφαρμογή.

- Απαιτείται μεγαλύτερος χρόνος και κόστος για την ανάπτυξη της εφαρμογής, ειδικότερα αν ο κύκλος ανάπτυξης ακολουθεί ένα μοντέλο (όπως για παράδειγμα το σπειροειδές) που περιλαμβάνει στάδια ελέγχου και αποτίμησης της επικινδυνότητας.

4.2 Αρχές Ασφαλούς Προγραμματισμού

Έχουν προταθεί από ανεξάρτητους οργανισμούς (όπως ο οργανισμός OWASP) ή ομάδες ασφάλειας (όπως η ομάδα CERT του πανεπιστημίου Carnegie Mellon), ορισμένες αρχές ασφαλούς προγραμματισμού οι οποίες συνοψίζονται ως εξής:

- Κάθε επιπλέον ιδιότητα που προστίθεται σε μια εφαρμογή, προσθέτει στη συνολική επικινδυνότητα της εφαρμογής αυτής. Πρέπει να επιδιώκεται η μείωση της λεγόμενης «επιφάνειας επίθεσης» (attack surface), προσθέτοντας εκείνα τα ελάχιστα χαρακτηριστικά, τα οποία είναι απαραίτητα προκειμένου να ικανοποιηθούν οι προδιαγραφές λειτουργικότητάς της. Αυτή η αρχή είναι γνωστή ως **Αρχή της Ελάχιστης Επιφάνειας Επίθεσης (Minimum Attack Surface Principle)**.

- Φροντίζουμε ώστε η εφαρμογή να εγκαθίσταται με τις ρυθμίσεις ασφάλειας εξ' ορισμού (by default) ενεργοποιημένες σε μέγιστο βαθμό. Στη συνέχεια, δίνεται στο χρήστη η δυνατότητα μείωσης του επιπέδου ασφάλειας της εφαρμογής, εφόσον το επιθυμεί και με δική του ευθύνη.

- Εφαρμόζουμε, γενικά, την **Αρχή του Ελάχιστου Προνομίου (Principle of Least Privilege)**, που ορίζει ότι στον κάθε χρήστη θα πρέπει να αποδίδεται το ελάχιστο σύνολο προνομίων το οποίο απαιτείται για να μπορεί να εκτελέσει μια εργασία του.

- Είναι καλό να προσθέτουμε περισσότερους από ένα μηχανισμούς ασφάλειας αν κάτι τέτοιο δεν επηρεάζει σημαντικά την απόδοση ή τη λειτουργικότητα της εφαρμογής μας. Αν μια ευπάθεια μειώνεται με την εφαρμογή ενός μηχανισμού ασφάλειας, τότε σίγουρα θα είναι δυσκολότερο να την εκμεταλλευτεί κάποιος επίδοξος εισβολέας έχοντας να υπερνικήσει επιπρόσθετους (δυο ή περισσοτέρους) μηχανισμούς ασφάλειας. Σε εφαρμογές διαδικτύου κάτι τέτοιο θα σήμαινε, για παράδειγμα, το να σχεδιάζονται φόρμες εισαγωγής στοιχείων με πολλαπλά επίπεδα επαλήθευσης. Με αυτό τον τρόπο ο χρήστης αυθεντικοποιείται για να εκτελέσει μια εργασία της εφαρμογής, αλλά όχι για το σύνολο των διαθέσιμων εργασιών. Η εκτέλεση μιας επόμενης εργασίας απαιτεί την εκ νέου αυθεντικοποίηση του χρήστη. Είναι ιδιαίτερα σημαντικός ο τρόπος διαχείρισης από τον ίδιο τον κώδικα (resilience) μιας πιθανής αστοχίας της εφαρμογής, καθώς πρέπει να προληφθεί οποιαδήποτε παρενέργεια που θα μπορούσε να οδηγήσει σε δημιουργία ευπάθειας. Για παράδειγμα, στο παρακάτω τμήμα κώδικα Java το οποίο συναντάμε στον ιστότοπο https://www.owasp.org, ο κακός χειρισμός εξαίρεσης (exception handling) οδηγεί σε απόδοση του ρόλου του διαχειριστή (administrator) στον τελικό χρήστη που εκτελεί την εφαρμογή, αν το τμήμα του κώδικα badCode() αποτύχει:

```
/*
Αν το τμήμα κώδικα badCode ή το τμήμα κώδικα isUserInRole
αποτύχει, τότε ο χρήστης παραμένει σε ρόλο administrator,
όπως αρχικά του είχε αποδοθεί στην πρώτη γραμμή κώδικα.
```

```
*/
isAdmin = true;
try {
     badCode();
     isAdmin = isUserInRole( "Administrator" );
}
catch (Exception ex) {
log.write(ex.toString());
}
```

- Δεν πρέπει να εμπιστευόμαστε χωρίς έλεγχο υπηρεσίες οι οποίες προσφέρονται από πληροφοριακά συστήματα τρίτων οργανισμών. Δεν είναι βέβαιο ότι κάθε οργανισμός διαθέτει μια πολιτική ασφάλειας η οποία καθορίζει τα πλαίσια της ασφαλούς λειτουργίας των πληροφοριακών του συστημάτων. Ακόμη και αν υπάρχει πολιτική ασφάλειας, τότε αυτή πιθανώς να απέχει από τα δικά μας πρότυπα και επιθυμητά επίπεδα ασφάλειας. Όλα τα εξωτερικά πληροφοριακά συστήματα θα πρέπει να αντιμετωπίζονται και να ελέγχονται με την ίδια αυστηρότητα.

- Η πιστή εφαρμογή της **Αρχής του Διαχωρισμού Καθηκόντων (Separation of Duties)** είναι σημαντική, γιατί με τον κατάλληλο διαχωρισμό (π.χ. των διεργασιών και των ρόλων) είναι ευκολότερο να ορίζουμε ρητά τα δικαιώματα της κάθε διεργασίας ή ρόλου που εμπλέκεται στην επιτέλεση μιας εργασίας, χωρίς να δημιουργούμε γκρίζες περιοχές αμφισβήτησης δικαιωμάτων.

- Θα πρέπει να αποφεύγεται η απόκρυψης του τρόπου λειτουργίας των μηχανισμών ασφάλειας, η οποία είναι γνωστότερη ως «**security by obscurity**». Η μυστικοπάθεια σε ότι αφορά τις υπάρχουσες ευπάθειες ενός πληροφοριακού συστήματος ή τους μηχανισμούς ασφάλειας (π.χ. κρυπτογραφικοί αλγόριθμοι), καθιστά το σύστημα «ασφαλές» μέχρι του σημείου γνωστοποίησης ενός τέτοιου «μυστικού». Παράδειγμα προς μίμηση αποτελεί το λειτουργικό σύστημα Linux, του οποίου ο κώδικας διατίθεται ελεύθερα, ώστε πολλές χιλιάδες ερευνητών να έχουν την ευκαιρία να τον ελέγξουν για τυχόν σφάλματα, τα οποία θα προκαλούσαν τη δημιουργία ευπαθειών. Ένας έλεγχος του κώδικα και του τρόπου λειτουργίας των μηχανισμών ασφάλειας από μεγάλο αριθμό ερευνητών σίγουρα υπόσχεται καλύτερα αποτελέσματα από τον αντίστοιχο έλεγχο που θα διεξήγαγε μια μικρή ομάδα μερικών δεκάδων ανθρώπων (π.χ. κρυπταναλυτών).

- Σε συνδυασμό με την Αρχή της Ελάχιστης Επιφάνειας Επίθεσης, προτείνεται η εφαρμογή της **Αρχής της Απλότητας**, που ορίζει ότι μεταξύ δύο λύσεων ο μηχανικός λογισμικού (software engineer) θα πρέπει να επιλέξει την απλούστερη λύση. Η αρχή αυτή είναι γνωστή και ως το «Ξυράφι του Όκαμ» (Occam's Razor).

4.3 Κατηγορίες Ευπαθειών

Οι περισσότερες ευπάθειες που παρουσιάζονται στις διαδικτυακές εφαρμογές ανήκουν σε μία από τις παρακάτω κατηγορίες:

- Υπερχείλιση ενταμιευτήρα (buffer overflow)

- Μη επικυρωμένη είσοδος (invalidated input) χρήστη

- Συνθήκες ανταγωνισμού (race conditions)

- Προβλήματα ελέγχου πρόσβασης (access control)

- Αποθήκευση σε σύστημα διαχείρισης βάσεων δεδομένων (ΣΔΒΔ)

4.3.1 Υπερχείλιση ενταμιευτήρα

Μια τέτοια υπερχείλιση (overflow) συμβαίνει όταν μια διεργασία της εφαρμογής προσπαθεί να εγγράψει δεδομένα πέρα από το τέλος (ή, περιστασιακά, πριν από την αρχή) ενός ενταμιευτήρα (buffer). Ο ενταμιευτήρας είναι ένα τμήμα της μνήμης RAM που χρησιμοποιείται από τη διεργασία μια δεδομένη χρονική στιγμή.

Γενικότερα, μια υπερχείλιση μνήμης μπορεί να προκαλέσει την κατάρρευση (break) μιας διεργασίας, μπορεί να θέσει σε κίνδυνο τα δεδομένα που αυτή χειρίζεται, ή ακόμη μπορεί να βοηθήσει σε μια προσπάθεια περαιτέρω κλιμάκωσης των προνομίων πρόσβασης που απέκτησε ένας επιτιθέμενος σε ένα υπολογιστικό σύστημα. Συνολικά, το 20% των επιθέσεων που έχουν αναφερθεί στις Ηνωμένες Πολιτείες της Αμερικής από την ομάδα ετοιμότητας US-CERT αφορούν περιπτώσεις υπερχείλισης ενταμιευτήρα.

Πιο αναλυτικά, κάθε διεργασία αποθηκεύει στη μνήμη RAM την είσοδο που καταχωρεί ο χρήστης χρησιμοποιώντας μια από τις ακόλουθες δομές δεδομένων:

- **Στοίβα (Stack)**, που υλοποιείται σε ένα τμήμα του χώρου διευθύνσεων μνήμης (memory address space) που χρησιμοποιεί η διεργασία και το οποίο αφορά μια μεμονωμένη κλήση συνάρτησης, μεθόδου, ή άλλης ισοδύναμης λειτουργίας.

- **Σωρός (Heap)**, που αποτελεί έναν γενικής χρήσης αποθηκευτικό χώρο για τη διεργασία. Τα δεδομένα, που αποθηκεύονται στο σωρό παραμένουν διαθέσιμα για το χρονικό διάστημα εκτέλεσης της διεργασίας ή έως ότου το λειτουργικό σύστημα αποφασίσει ότι η διεργασία δεν τα χρειάζεται πλέον.

Γενικότερα, επιθέσεις τύπου υπερχείλισης ενταμιευτήρα συμβαίνουν όταν κανείς εκμεταλλευτεί επιτυχώς τους περιορισμούς ή/και την ελλιπή διαχείριση των παραπάνω δομών δεδομένων.

4.3.2 Μη επικυρωμένη είσοδος από χρήστη

Κατά γενικό κανόνα, ο κώδικας της εφαρμογής θα πρέπει να ελέγχει όλα τα στοιχεία εισόδου που εισάγονται (input data) από τον χρήστη προκειμένου να επικυρώνεται ότι καταχωρήθηκαν οι κατάλληλες και επιτρεπτές τιμές δεδομένων στο πλαίσιο της επιθυμητής λειτουργικότητας της εφαρμογής.

Μια εφαρμογή η οποία δεν εκτελεί έλεγχο κατά την καταχώρηση δεδομένων εισόδου από μη αξιόπιστη πηγή προέλευσης αποτελεί ένα πιθανό στόχο επίθεσης από κακόβουλους χρήστες. Στο πλαίσιο αυτής της θεώρησης, κάθε χρήστης μιας εφαρμογής είναι μια μη αξιόπιστη πηγή προέλευσης και ο έλεγχος των εισηγμένων δεδομένων είναι επιβεβλημένος.

Τα κυριότερα σημεία ελέγχου από τον κώδικα της εφαρμογής, κατά τη διαδικασία επικύρωσης, είναι τα παρακάτω:

- Όρια τιμών: Για κάθε πεδίο (field) θα πρέπει να ορίζεται ένα κάτω και ένα άνω όριο επιτρεπτών τιμών εισόδου. Π.χ. ένα πεδίο το οποίο δέχεται τιμή χρηματικού ποσού θα πρέπει να ελέγχεται ώστε η τιμή αυτή να μην είναι αρνητική ή να μην ξεπερνάει ένα άνω όριο.

- Μήκος εισόδου: Για κάθε πεδίο θα πρέπει να έχει τεθεί ένα όριο στο μήκος της σειράς αλφαριθμητικών χαρακτήρων που μπορεί να δεχθεί. Π.χ. ένα πεδίο στο οποίο καταχωρείται η ηλικία του χρήστη, δεν θα πρέπει να δέχεται τιμή που το μήκος της υπερβαίνει τους 2 χαρακτήρες.

- Ανυπαρξία τιμής: Η γλώσσα προγραμματισμού C χρησιμοποιεί τη δεκαεξαδική τιμή 0x00 (null byte) ως διακριτικό τερματισμού μιας σειράς αλφαριθμητικών χαρακτήρων. Επομένως, το null byte δε θα πρέπει να περιέχεται στην τιμή εισόδου, αφού οι χαρακτήρες που το ακολουθούν θα

μπορούσαν να χρησιμοποιηθούν εσφαλμένα σε μια επόμενη δραστηριότητα της εφαρμογής και να προκαλέσουν πιθανώς μια απρόβλεπτη συμπεριφορά.

Για την επικύρωση των εισαγόμενων δεδομένων συνήθως χρησιμοποιείται μια από τις ακόλουθες δύο τεχνικές:

- Μαύρη Λίστα: Αναγνώριση μη έγκυρων δεδομένων και αφαίρεσή τους. Για το σκοπό αυτό διαθέτουμε μια μαύρη λίστα (black list) με πιθανά δεδομένα εισόδου, τα οποία δεν θεωρούνται έγκυρα. Στη συνέχεια, κάθε τιμή εισόδου ελέγχεται έναντι των περιεχομένων της μαύρης λίστας. Όσα δεδομένα συμπίπτουν, απορρίπτονται.

- Λευκή Λίστα: Αναγνώριση έγκυρων δεδομένων και αφαίρεση των υπόλοιπων. Για το σκοπό αυτό, διαθέτουμε μια λευκή λίστα (white list) με πιθανά δεδομένα εισόδου τα οποία θεωρούνται έγκυρα. Στη συνέχεια, κάθε τιμή εισόδου ελέγχεται έναντι των περιεχομένων της λευκής λίστας και όσα δεδομένα συμπίπτουν γίνονται δεκτά, ενώ όλα τα υπόλοιπα απορρίπτονται.

4.3.3 Συνθήκες ανταγωνισμού

Συνθήκη ανταγωνισμού (race condition) έχουμε όταν δύο νήματα (threads) προσπαθούν να προσπελάσουν το ίδιο αντικείμενο ταυτόχρονα και η συμπεριφορά του προγράμματος εξαρτάται από το ποιο νήμα προηγείται στη σειρά προσπέλασης. Η σειρά αυτή δεν εμπίπτει σε μια ντετερμινιστική ακολουθία, άρα όταν δεν ελέγχεται (π.χ. synchronized) είναι μη προβλέψιμη και εξαρτάται από το λειτουργικό σύστημα.

Πέρα από τα προβλήματα στη λειτουργία της εφαρμογής, οι επιτιθέμενοι μπορούν μερικές φορές να επωφεληθούν από μικρά χρονικά κενά κατά την επεξεργασία του κώδικα για να παρέμβουν στην εξέλιξη των εργασιών, τις οποίες στη συνέχεια εκμεταλλεύονται. Ένας κακόβουλος χρήστης μπορεί να εκμεταλλευτεί μια τέτοια κατάσταση για να εισάγει κατάλληλο κώδικα. Για παράδειγμα, να αλλάξει το όνομα ενός αρχείου και να επηρεάσει την ομαλή λειτουργία της διεργασίας.

4.3.4 Προβλήματα ελέγχου πρόσβασης

Με τη διαδικασία του ελέγχου πρόσβασης (access control) ελέγχουμε συνήθως στη βάση της επιβεβαιωμένης ταυτότητάς του (authentication) αν ο νόμιμος χρήστης της εφαρμογής είναι εξουσιοδοτημένος (authorization) και επομένως του επιτρέπεται να εκτελέσει μια εργασία σε συγκεκριμένους πόρους του πληροφοριακού συστήματος μετά από αίτημά του. Οι μηχανισμοί ελέγχου πρόσβασης είναι δυνατό να επιβάλλονται και να ελέγχονται από το λειτουργικό σύστημα, το σύστημα διαχείρισης βάσεων δεδομένων (ΣΔΒΔ), την ίδια την εφαρμογή, μια υπηρεσία, ένα πρωτόκολλο επικοινωνίας, κ.ά.

Οι εγγενείς ευπάθειες των μηχανισμών αυθεντικοποίησης, εξουσιοδότησης ή/και των κρυπτογραφικών μηχανισμών, που πιθανώς χρησιμοποιούνται για τον έλεγχο πρόσβασης, οφείλονται κυρίως σε σχεδιαστικά σφάλματα ή σε αστοχίες της συγγραφής κώδικα της εφαρμογής. Κατά την ανάπτυξη των διαδικτυακών εφαρμογών, σημαντικό ρόλο κατέχει η αρχιτεκτονική της εφαρμογής που εφαρμόζεται (π.χ. μοντέλο πελάτη/εξυπηρετητή), καθώς και η γλώσσα προγραμματισμού.

4.3.5 Αποθήκευση σε Σύστημα Διαχείρισης Βάσεων Δεδομένων

Κατά τη λειτουργία μιας διαδικτυακής εφαρμογής, συνήθως αποθηκεύονται και ανακτώνται δεδομένα σε /από ένα σύστημα διαχείρισης βάσεων δεδομένων (ΣΔΒΔ). Η επικοινωνία με το ΣΔΒΔ πραγματοποιείται με ανταλλαγή συμβολοσειρών οι οποίες είναι είτε δεδομένα της εφαρμογής είτε εντολές επεξεργασίας τους ή εντολές ελέγχου του ΣΔΒΔ. Στη συνέχεια, το ΣΔΒΔ χρησιμοποιεί ένα διερμηνευτή ο οποίος αποκωδικοποιεί τις εισερχόμενες συμβολοσειρές διαβάζοντας ένα-προς-ένα τους χαρακτήρες και αποφασίζοντας για το είδος της πληροφορίας που περιέχεται στη συμβολοσειρά (δεδομένα εφαρμογής ή εντολή προς εκτέλεση).

Πρόβλημα ασφάλειας παρουσιάζεται όταν ο προγραμματιστής προσπαθεί να αποθηκεύσει δεδομένα στο ΣΔΒΔ και ο διερμηνευτής μεταφράζει εσφαλμένα ένα τμήμα των δεδομένων ως εντολή προς εκτέλεση. Σε

μια τέτοια περίπτωση υπάρχει ο κίνδυνος ένας κακόβουλος χρήστης να κατορθώσει να εκμεταλλευτεί μια τέτοια ευπάθεια και να εισάγει εντολές προς εκτέλεση, δημιουργώντας ρήγμα στην ασφάλεια της εφαρμογής. Αυτό το ιδιαίτερα διαδεδομένο είδος επίθεσης ονομάζεται **ψεκασμός κώδικα SQL (SQL Injection)**.

Κατά την επίθεση ψεκασμού κώδικα SQL ο επιτιθέμενος εισάγει κατάλληλα διαμορφωμένες συμβολοσειρές ώστε να προκαλέσει την εκτέλεση εντολών SQL από το ΣΔΒΔ που πλήττουν την εμπιστευτικότητα ή την ακεραιότητα των δεδομένων της εφαρμογής. Η επιτυχία της επίθεσης εξαρτάται από την ύπαρξη αποτελεσματικού κώδικα λογισμικού εφαρμογής για την κατάλληλη επεξεργασία και έλεγχο των εισαγόμενων από τον τελικό χρήστη χαρακτήρων (input validation).

4.4. Μελέτη Περίπτωσης: Java

Η γλώσσα προγραμματισμού Java αποτελεί μια αξιόλογη περίπτωση για την εφαρμογή των παραπάνω αρχών ασφαλούς διαδικτυακού προγραμματισμού. Οι διαδικτυακές εφαρμογές που αναπτύσσονται με τη γλώσσα αυτή μπορούν να εκτελεστούν από πολλούς χρήστες με διαφορετικά δικαιώματα πρόσβασης, χωρίς να δημιουργούνται παρενέργειες, με ή χωρίς πρόθεση από τον τελικό χρήστη. Αυτό επιτυγχάνεται με τη βοήθεια ενός ειδικού υποσυστήματος της γλώσσας το οποίο ονομάζεται επιβεβαιωτής ενδιάμεσου κώδικα (bytecode verifier). Ο επιβεβαιωτής ελέγχει τον ενδιάμεσο κώδικα που παράγεται κατά την εκτέλεση μιας εφαρμογής εντοπίζοντας τα «ύποπτα» σημεία, των οποίων η εκτέλεση θα μπορούσε να προκαλέσει ευπάθειες στο υπολογιστικό σύστημα.

Ωστόσο, ενώ η αρχιτεκτονική της Java μπορεί να προστατέψει το υπολογιστικό σύστημα από κακόβουλα προγράμματα που εκτελούνται μέσω του διαδικτύου, είναι ανυπεράσπιστη σε υλοποιήσεις εφαρμογών με προγραμματιστικά σφάλματα. Τέτοια σφάλματα μπορεί να οδηγήσουν σε ανεπιθύμητη χρήση αρχείων του συστήματος, εκτυπωτών, κάμερας, μικροφώνου και άλλων περιφερειακών. Είναι δυνατό, επίσης, τέτοια σφάλματα να καταστήσουν το υπολογιστικό σύστημα υποχείριο («ζόμπι») ενός επιτιθέμενου, δηλαδή να το μετατρέψουν σε έναν ενδιάμεσο σταθμό γενικότερων επιθέσεων, όπως επιθέσεων υποκλοπής προσωπικών δεδομένων από μια συντονισμένη επίθεση που εξαπολύουν πολλά υπολογιστικά συστήματα μέσω του Διαδικτύου.

Η γλώσσα προγραμματισμού Java διαθέτει εγγενώς μηχανισμούς ασφάλειας, οι οποίοι υπερνικούν το συνηθισμένο πρόβλημα της υπερχείλισης ενταμιευτήρα. Ωστόσο, υπάρχουν αρκετά σημεία στα οποία θα πρέπει να δώσει ιδιαίτερη προσοχή ο προγραμματιστής, προκειμένου να αναπτύξει προγράμματα χωρίς ευπάθειες. Τα κυριότερα σημεία στα οποία θα πρέπει να δώσει έμφαση είναι τα παρακάτω:

- **Απλός σχεδιασμός**: Κατά την ανάπτυξη του λογισμικού εφαρμογής θα πρέπει να επιλέγεται πάντα ο απλούστερος σχεδιασμός, ώστε για τα λάθη που πιθανώς να προκύψουν να είναι εύκολος ο εντοπισμός των αιτίων τους.

- **Ασφάλεια από την αρχή**: Η ασφάλεια θα πρέπει να απασχολεί τον προγραμματιστή από το αρχικό στάδιο της σχεδίασης της εφαρμογής. Η προσπάθεια εισαγωγής μηχανισμών ασφάλειας σε μια εφαρμογή λογισμικού κατά τη διάρκεια ενός επόμενου σταδίου δεν είναι πάντα αποτελεσματική και είναι πιθανό να οδηγήσει σε περαιτέρω σφάλματα. Για παράδειγμα, χαρακτηρίζοντας μια κλάση ως final, προστατεύεται το λογισμικό από πιθανούς κακόβουλους χρήστες οι οποίοι θα επιδίωκαν να δημιουργήσουν νέες κλάσεις-κληρονόμους, ή να κάνουν override μεθόδους αυτής της κλάσης. Επίσης, η χρήση του SecurityManager σε ένα τμήμα κώδικα, υποδηλώνει πως αυτή η περιοχή κώδικα θα ελεγχθεί σχολαστικά.

- **Περιορισμός Προνομίων**: Παρά τις προσπάθειες για συγγραφή ασφαλούς κώδικα, είναι σχεδόν βέβαιο ότι θα συνεχίσουν να υπάρχουν ατέλειες/σφάλματα στα διάφορα προϊόντα λογισμικού διαδικτυακών εφαρμογών. Γι' αυτό το λόγο, είναι προτιμότερο ο κώδικας να εκτελείται με μειωμένα προνόμια, ώστε ακόμη και αν υπάρχουν σφάλματα τα οποία θα μπορούσε να εκμεταλλευτεί μια πιθανή απειλή για να εξαπολύσει μια επιτυχημένη επίθεση, να μη μπορέσει σε μια τέτοια περίπτωση η επίθεση να «εξαπλωθεί» στο υπόλοιπο υπολογιστικό σύστημα. Τα προνόμια εκτέλεσης του κώδικα Java μπορούν να δηλωθούν είτε στατικά, με το μηχανισμό ασφάλειας της Java ο οποίος χρησιμοποιεί αρχεία πολιτικής (policy files), ή

δυναμικά, με τη χρήση του μηχανισμού java.security.AccessController.doPrivileged. Οι εφαρμογές Rich Internet (RIA) μπορούν να προσδιορίζουν τα απαιτούμενα προνόμια εκτέλεσης μέσω της χρήσης ενός εφαρμογιδίου (applet) ή μέσω του JNLP. Επίσης, ένα υπογεγραμμένο αρχείο jar μπορεί να δηλώνει στο αρχείο Manifest ένα χαρακτηριστικό, το οποίο θα προσδιορίζει αν απαιτείται να εκτελεστεί με πλήρη δικαιώματα ή με περιορισμένα δικαιώματα στο πλαίσιο ενός προστατευόμενου περιβάλλοντος (sandbox). Και στις δύο αυτές περιπτώσεις, κάθε παραβίαση της ασφάλειας θα ενεργοποιήσει την παρέμβαση του περιβάλλοντος JRE (Java Runtime Environment), με ταυτόχρονη κλήση του χειριστή εξαιρέσεων.

- **Καθορισμός ορίων εμπιστοσύνης**: Για να είμαστε βέβαιοι ότι ένα σύστημα προστατεύεται επαρκώς θα πρέπει να γνωρίζουμε τα όρια που το καθορίζουν, σε σχέση με κάθε άλλο εξωτερικό σύστημα, το οποίο ανήκει στο εξωτερικό περιβάλλον λειτουργίας του συστήματος μας (όπως για παράδειγμα το διαδίκτυο). Ως όρια του συστήματος, ορίζουμε τα σημεία εκείνα από τα οποία τα δεδομένα εξέρχονται στο εξωτερικό περιβάλλον ή εισέρχονται στο σύστημα μας προερχόμενα από το εξωτερικό περιβάλλον. Ειδικότερα, τα δεδομένα τα οποία διαπερνούν αυτά τα όρια, κατευθυνόμενα προς το εσωτερικό του συστήματος, θα πρέπει να «αποστειρώνονται» (sanitize) και να επικυρώνονται (validate) πριν χρησιμοποιηθούν από τον κώδικα μιας διαδικτυακής εφαρμογής.

- **Αντοχή σε επιθέσεις άρνησης εξυπηρέτησης**: Οι επιθέσεις άρνησης εξυπηρέτησης (Denial of Service) στοχεύουν στην άσκοπη κατανάλωση πόρων του υπολογιστικού συστήματος μέχρι του σημείου κατάρρευσης από την πλευρά του εξυπηρετητή, λόγω εξάντλησης των διαθέσιμων πόρων του. Οι συνηθισμένοι πόροι εξυπηρετητή, που αποτελούν στόχους εκμετάλλευσης, είναι η επεξεργαστική ισχύς, η μνήμη RAM, ο αποθηκευτικός χώρος δίσκου και το εύρος ζώνης δικτύου (bandwidth) που του διατίθεται.

Απαιτείται ιδιαίτερη προσοχή κυρίως στις ακόλουθες περιπτώσεις:

- Δημιουργία διανυσματικών αρχείων γραφικών (vector graphics) πολύ μεγάλου μεγέθους (αρχεία svg ή font).

- Σφάλματα υπερχείλισης σε περιπτώσεις ακεραίων τιμών.

- Ένα γραφικό αντικείμενο, το οποίο έχει δημιουργηθεί από τη σάρωση ενός κειμένου, μπορεί να έχει απαιτήσεις σε μνήμη και χωρητικότητα αποθηκευτικού χώρου. πολλές φορές. μεγαλύτερη από ότι το αρχικό κείμενο.

- Αποσυμπίεση υπερσυμπιεσμένων αρχείων («Βόμβες zip»), όπως για παράδειγμα οι εικόνες GIF. Όταν αποσυμπιέζονται τέτοια αρχεία, είναι ασφαλέστερο να τίθεται όριο στο μέγεθος των δεδομένων που τελικά παράγονται.

- Αυθαίρετη κατανάλωση χρόνου από επεξεργασία εκφράσεων XPath.

- Απεριόριστη χρήση μνήμης ή επεξεργαστικής ισχύος από τη διαδικασία σειριοποίησης ή αποσειριοποίησης (java serialization/deserialization).

Η αποδέσμευση των πόρων συστήματος μετά τη χρήση τους (όταν πλέον δεν είναι απαραίτητοι), είναι σημαντική ιδιαίτερα στην περίπτωση ανοικτών αρχείων, κλειδωμάτων (locks) και μνήμης η οποία δεσμεύτηκε ρητά. Η αποδέσμευση των πόρων ενός υπολογιστικού συστήματος είναι απαραίτητη για να μπορούμε να εξυπηρετήσουμε καλύτερα το σκοπό της εφαρμογής μας, χωρίς να επιβαρύνουμε το υπολογιστικό σύστημα.

Το μοτίβο (pattern) **Execute Around Method** παρέχει έναν εξαιρετικό τρόπο διαχείρισης του ζεύγους ενεργειών απόκτησης - απελευθέρωσης πόρων. Στην έκδοση 8 της γλώσσας Java, το μοτίβο αυτό χρησιμοποιείται με τη χρήση της ιδιότητας lambda. Για παράδειγμα:

```
/*
το μοτίβο Execute around method χρησιμοποιείται για να
εκτελέσουμε, μετά από κλήση, διεργασίες οι οποίες είναι
επαναλαμβανόμενες κατά τη διάρκεια λειτουργίας της εφαρμογής
μας.
*/

long sum = readFileBuffered(InputStream in -> {
     long current = 0;
    for (;;) {
          int b = in.read();
        if (b == -1) {
                return current;
        }
        current += b;
    }
});
```

Η σύνταξη try-with-resource, που παρουσιάστηκε στην έκδοση 7 της γλώσσας Java, χειρίζεται αυτόματα την απελευθέρωση πολλών τύπων πόρων. Για παράδειγμα:

```
/*
Η σύνταξη try-with-resource επιβεβαιώνει ότι μετά την κλήση
της μεθόδου, οι πόροι που χρησιμοποιήθηκαν θα αποδεσμευτούν
αυτόματα.
*/

public R readFileBuffered(InputStreamHandler handler) throws
IOException {
     try (final InputStream in = Files.newInputStream(path))
{
          handler.handle(new BufferedInputStream(in));
    }
}
```

Πόροι οι οποίοι δεν μπορούν να χρησιμοποιήσουν αυτές τις νέες δομές θα πρέπει να χρησιμοποιούν το συνηθισμένο τρόπο κτήσης και απελευθέρωσης πόρων. Για παράδειγμα:

```
/*
αυτόματα αποδεσμεύονται πόροι οι οποίοι εφαρμόζουν το
java.lang.AutoCloseable. Οι υπόλοιποι πόροι πρέπει ρητά να
αποδεσμευτούν με τη κλήση unlock()
*/

public R locked (Action action) {
     lock.lock();
     try {
          return action.run();
     } finally {
          lock.unlock();
    }
}
```

Επίσης, είναι σημαντικό να αδειάζουν όλοι οι ενταμιευτήρες εξόδου (output buffers). Αν αποτύχει το άδειασμα των ενταμιευτήρων, θα πρέπει να γίνεται ρίψη εξαίρεσης.

```
/*
Η εντολή out.flush() στο πλαίσιο της εντολής try επιβεβαιώνει
ότι αν το άδειασμα του ενταμιευτήρα δεν επιτύχει, τότε θα
γίνει ρίψη εξαίρεσης
*/
public void writeFile(OutputStreamHandler handler) throws
IOException {
     try (final OutputStream rawOut =
Files.newOutputStream(path)) {
          final BufferedOutputStream out = new
BufferedOutputStream(rawOut);
        handler.handle(out);
        out.flush();
    }
}
```

4.4.1 Ευαίσθητα δεδομένα

Αρκετές φορές, οι μηχανισμοί εξαίρεσης μπορεί να εκθέσουν δεδομένα τα οποία επιθυμούμε να προστατεύσουμε. Για παράδειγμα, αν μια μέθοδος καλέσει τον κατασκευαστή (constructor) java.io.FileInputStream για να αναγνώσει ένα αρχείο ρυθμίσεων και το αρχείο δεν υπάρχει, τότε επιστρέφεται μια εξαίρεση java.io.FileNotFoundException, η οποία περιέχει τη διαδρομή του αρχείου. Η διάδοση αυτής τη εξαίρεσης προς τα πίσω στην καλούσα μέθοδο, μπορεί να αποκαλύψει τη δομή του συστήματος αρχείων. Αρκετές επιθέσεις στηρίζονται στη γνώση αυτών των διαδρομών του συστήματος αρχείων.

Ακόμη, είναι πιθανό σε μια τέτοια διαδρομή να περιέχονται στοιχεία, όπως το όνομα χρήστη ή ο οικείος φάκελός του (home directory). Ο μηχανισμός SecurityManager μπορεί να προστατεύει αυτή την πληροφορία όταν δηλώνεται σε ιδιότητες συστήματος, όπως είναι το user.home. Ωστόσο, η πληροφορία αυτή θα μπορούσε να ξεφύγει από τον έλεγχο του SecurityManager σε περίπτωση ενεργοποίησης μιας εξαίρεσης και να παρουσιαστεί στον τελικό χρήστη. Χρειάζεται, επομένως, ιδιαίτερη προσοχή στη διαχείριση των εξαιρέσεων, ακόμη και για τις περιπτώσεις όπου γίνεται χρήση βιβλιοθηκών οι οποίες μια δεδομένη στιγμή δεν χρησιμοποιούν ευαίσθητα δεδομένα, αλλά είναι πιθανό μα επόμενη έκδοσή τους να τα χρησιμοποιεί.

Οι εξαιρέσεις μπορούν επίσης να αφορούν ευαίσθητα δεδομένα σχετικά με τις ρυθμίσεις και τα εσωτερικά χαρακτηριστικά του υπολογιστικού συστήματος. Γι' αυτό, δεν πρέπει να επιστρέφονται στους τελικούς χρήστες πληροφορίες εξαίρεσης, εκτός και αν το επιβάλλει σε ειδικές περιπτώσεις το περιεχόμενο των σχετικών μηνυμάτων. Για παράδειγμα, δεν πρέπει να επιστρέφονται μηνύματα εξαιρέσεως σχετικά με ίχνη στοίβας (stack traces) μέσα σε σχόλια κώδικα HTML. Ακόμη υπάρχουν άλλου είδους πληροφορίες, όπως είναι τα δεδομένα προσωπικού χαρακτήρα (π.χ. αριθμός κοινωνικής ασφάλισης, αριθμός αστυνομικής ταυτότητας κ.λπ.), τα οποία δεν πρέπει να διατηρούνται στο σύστημα περισσότερο από όσο απαιτείται για την ορθή λειτουργία των διεργασιών του συστήματος. Για παράδειγμα, η αποθήκευση και διατήρηση τέτοιων πληροφοριών σε αρχεία καταγραφής μητρώου (log files) θα πρέπει να συνοδεύεται από βοηθητικές ενέργειες ασφάλειας. Τα αρχεία καταγραφής θα πρέπει να εξαιρούνται από πιθανές ενέργειες αναζήτησης ή να διαγράφεται η σχετική πληροφορία όταν περατωθεί το χρονικό διάστημα χρήσης της.

Οι ευαίσθητες πληροφορίες απαιτούν ιδιαίτερο χειρισμό και στην περίπτωση όπου κρατούνται μόνο στη μνήμη, καθώς και εκεί αποτελούν στόχο κακόβουλων ενεργειών. Θα πρέπει να φροντίζουμε να πραγματοποιούμε εκκαθάριση της μνήμης όταν καταχωρούνται ευαίσθητες πληροφορίες.

4.4.2 Ψεκασμός εντολών

Είναι πλέον σύνηθες το φαινόμενο κατασκευής αλφαριθμητικών τα οποία προορίζονται για χρήση απλού κειμένου, αλλά χρησιμοποιούνται με τέτοιο τρόπο ώστε να καταλήγουν να αποτελέσουν εκτελέσιμες εντολές.

Για αυτό, θα πρέπει να αποφεύγεται η χρήση δυναμικής δημιουργίας εντολών SQL, καθώς ένας κακόβουλος χρήστης μπορεί να εκμεταλλευτεί το γεγονός ότι χρησιμοποιώντας το χαρακτήρα «΄» (quote) μπορεί να προσθέσει στη συμβολοσειρά εισόδου μια εντολή SQL, η οποία στη συνέχεια θα εκτελεστεί από τη διεργασία.

Για παραμετροποιημένες εντολές SQL μέσω του Java Database Connectivity (JDBC), συστήνεται να γίνεται ορθή χρήση του αντικειμένου java.sql.PreparedStatement ή του αντικειμένου java.sql.CallableStatement και όχι του αντικειμένου java.sql.Statement. Ακόμη, προτιμότερη είναι η χρήση μιας έτοιμης βιβλιοθήκης διαχείρισης βάσης δεδομένων, έτσι ώστε να πραγματοποιείται ορθή χρήση των απαραίτητων εντολών SQL.

Ένα παράδειγμα ορθής χρήσης του αντικειμένου java.sql.PreparedStatement παρουσιάζεται στο παρακάτω τμήμα κώδικα:

```
String sql = "SELECT * FROM User WHERE userId = ?";
PreparedStatement stmt = con.prepareStatement(sql);
stmt.setString(1, userId);
ResultSet rs = prepStmt.executeQuery();
```

Δεδομένα τα οποία προέρχονται από μη αξιόπιστη πηγή προέλευσης (όπως ο τελικός χρήστης) θα πρέπει να εξετάζονται προσεκτικά, πριν εισαχθούν σε αρχεία HTML ή XML. Μια αποτυχία ελέγχου τέτοιων δεδομένων θα μπορούσε να οδηγήσει σε επιθέσεις τύπου Cross-Site Scripting (XSS) ή XML Injection. Προσοχή απαιτείται κυρίως στην περίπτωση των Java Server Pages (JSP). Οι κυριότεροι τρόποι χειρισμού τέτοιων δεδομένων είναι το φιλτράρισμα (έλεγχος και επικύρωση), η αποφυγή αποστολής τους και η κωδικοποίηση επικίνδυνων χαρακτήρων, οι οποίοι μπορεί να μεταφραστούν ως χαρακτήρες ελέγχου. Και σε αυτή την περίπτωση, η χρήση έτοιμων βιβλιοθηκών αποτελεί εφαρμογή μιας ορθής πρακτικής ασφάλειας.

Σε περιβάλλον Unix χρειάζεται ιδιαίτερη προσοχή όταν η διαδικτυακή εφαρμογή παράγει ως έξοδο αλφαριθμητικά σε γραμμή κελύφους (shell), όπου υπάρχει το ενδεχόμενο ένας ή περισσότεροι χαρακτήρες να θεωρηθούν ως διακόπτες (options) σε μια εντολή προς εκτέλεση (π.χ. προς το λειτουργικό σύστημα). Σε αυτές τις περιπτώσεις, συνηθίζεται η κωδικοποίηση τέτοιων χαρακτήρων σε μια μορφή όπως η Base64.

Τα αρχεία εικόνων τύπου BMP μπορεί να περιέχουν αναφορές σε τοπικά αρχεία ICC (International Color Consortium). Ενώ το περιεχόμενο των αρχείων ICC είναι απίθανο να παρουσιάζει κάποιο ενδιαφέρον, η προσπάθεια να διαβαστούν τα αρχεία μπορεί να είναι ένα ζήτημα, καθώς τα αρχεία αυτά συνήθως βρίσκονται αποθηκευμένα σε περιοχές κρίσιμες για τη λειτουργία του συστήματος. Γι' αυτό, είτε αποφεύγουμε τα αρχεία τύπου BMP, ή μειώνουμε τα προνόμιά, έτσι ώστε να μην επιτρέπεται η πρόσβαση σε αρχεία ICC.

Μερικά τμήματα Swing μεταφράζουν τμήματα κώδικα τα οποία ξεκινούν με <html> ως κώδικα HTML. Για να αποφύγουμε τον πιθανό κίνδυνο από μη έμπιστο κώδικα, ορίζουμε την ιδιότητα html.disable σε κάθε τέτοιο τμήμα θέτοντας την τιμή Boolean.TRUE. Για παράδειγμα:

```
label.putClientProperty("html.disable", true);
```

4.4.3 Προσβασιμότητα και επεκτασιμότητα

Τα Containers μπορεί να κρύβουν κώδικα υλοποίησης, τροποποιώντας την ιδιότητα ασφάλειας package.access. Αυτή η ιδιότητα εμποδίζει μη έμπιστες κλάσεις να συνδεθούν και να κληθούν από φορτωτές κλάσεων (class loaders) στην καθορισμένη ιεραρχία πακέτου. Πρέπει να λαμβάνεται μέριμνα για να διασφαλιστεί ότι τα πακέτα δεν είναι προσβάσιμα από μη έμπιστα περιβάλλοντα πριν οριστεί αυτή η ιδιότητα. Το παρακάτω παράδειγμα υποδεικνύει την ορθή χρήση της ιδιότητας package.access:

```
private static final String PACKAGE_ACCESS_KEY =
"package.access";

static {

     //Ανάγνωση ιδιότητας package.access
     String packageAccess =
java.security.Security.getProperty(PACKAGE_ACCESS_KEY);
```

```
        //Ορθή χρήση ιδιότητας package.access
        java.security.Security.setProperty(PACKAGE_ACCESS_KEY,((
    packageAccess == null || packageAccess.trim().isEmpty()) ? ""
    :(packageAccess + ","))
    +"xx.example.product.implementation.");
    }
```

Ένας ακόμη σημαντικός παράγοντας ασφάλειας, που πρέπει να εξετάζεται, είναι η επεκτασιμότητα των κλάσεων και των μεθόδων. Πρέπει ρητά να δηλώνονται με χαρακτηρισμό final οι κλάσεις που δεν πρέπει να επεκτείνονται. Όπως, για παράδειγμα, στο τμήμα κώδικα που ακολουθεί:

```
    //Η κλάση SensitiveClass δεν μπορεί να επεκταθεί
    public final class SensitiveClass {

        //Η μέθοδος Behavior δεν μπορεί να επεκταθεί
        private final Behavior;

        // Απόκρυψη κατασκευαστή
        private SensitiveClass(Behavior behavior) {
            this.behavior = behavior;
        }

        //Guarded construction.
        public static SensitiveClass newSensitiveClass(Behavior
    behavior) {
            // ... validate any arguments ...

            // ... perform security checks ...

            return new SensitiveClass(behavior);
        }
    }
```

4.4.4 Επαλήθευση εισόδου

Ενώ ο κώδικας Java υπόκειται σε έλεγχο πραγματικού χρόνου για τους τύπους, τα όρια πινάκων, τη χρήση βιβλιοθηκών κ.ά., ο εκτελέσιμος κώδικας (native code), δηλαδή ο κώδικας που έχει συμβολομεταφραστεί για μια συγκεκριμένη οικογένεια επεξεργαστών δεν υπόκειται σε κανένα έλεγχο, με αποτέλεσμα να υπάρχει ο κίνδυνος υπερχείλισης ενταμιευτήρα κατά την εκτέλεσή του. Γι' αυτό το λόγο, δε θα πρέπει να δηλώνονται ως δημόσιες (public) οι μέθοδοι του εκτελέσιμου κώδικα. Τέτοιες μέθοδοι θα πρέπει να δηλώνονται ως private και να χρησιμοποιούνται μέσω μιας public μεθόδου wrapper, όπως παρουσιάζεται στο παράδειγμα κώδικα που ακολουθεί:

```
    public final class NativeMethodWrapper {

        // Η μέθοδος nativeOperation δηλώνεται ιδιωτική
        private native void nativeOperation(byte[] data, int
    offset, int len);

        // Η μέθοδος doOperation δηλώνεται δημόσια για να
    χρησιμοποιηθεί ως διεπαφή της μεθόδου nativeOperation
        public void doOperation(byte[] data, int offset, int
    len) {
            data = data.clone();
```

```
        /*
          Επικύρωση δεδομένων εισόδου. Το άθροισμα offset+len
μπορεί να προκαλέσει υπερχείλιση ακεραίου. Για παράδειγμα αν
offset=1 and len=Integer.MAX_VALUE, τότε offset+len ==
Integer.MIN_VALUE το οποίο είναι μικρότερο από το
data.length. Επίσης, βρόγχοι της μορφής for (int i=offset;
i<offset+len; ++i) { ... } δεν θα προκαλούν πλέον εξαίρεση
          */

          if (offset < 0 || len < 0 || offset > data.length -
len) {
               throw new IllegalArgumentException();
          }
          nativeOperation(data, offset, len);
     }
}
```

4.4.5 Κατασκευή αντικειμένων

Κατά τη διάρκεια κατασκευής τους τα αντικείμενα βρίσκονται σε μια ιδιαίτερη κατάσταση όπου υπάρχουν αλλά δεν μπορούν να χρησιμοποιηθούν. Σε περίπτωση που ένας κατασκευαστής (constructor) σε μια non-final κλάση επιστρέψει μια εξαίρεση (exception), παρέχεται η δυνατότητα σε ένα κακόβουλο χρήστη για να προσπαθήσει να αποκτήσει πρόσβαση σε μια μερικώς αρχικοποιημένη έκδοση της κλάσης. Σε μια τέτοια κατάσταση, δεν έχουν αποδοθεί τιμές σε όλες τις μεταβλητές της μεθόδου και δεν έχουν αρχικοποιηθεί ολοκληρωμένα όλες οι μέθοδοι. Γι' αυτό, πρέπει να είμαστε βέβαιοι ότι μια non-final κλάση παραμένει εντελώς αχρησιμοποίητη, έως ότου ο κατασκευαστής της ολοκληρώσει την εκτέλεσή του επιτυχώς.

Ο έλεγχος ορθής δημιουργίας μιας non-final κλάσης μπορεί να γίνει κατά την κλήση this() ή super(), όπως στο παράδειγμα που ακολουθεί:

```
public abstract class ClassLoader {
     protected ClassLoader() {

     //έλεγχος ορθής δημιουργίας αντικειμένου
     this(securityManagerCheck());
     }

     private ClassLoader(Void ignored) {
          // ... συνέχεια αρχικοποίησης ...
     }

     private static Void securityManagerCheck() {
          SecurityManager security =
System.getSecurityManager();
          if (security != null) {
               security.checkCreateClassLoader();
          }
          return null;
     }
}
```

Για συμβατότητα με παλαιότερες εκδόσεις της γλώσσας Java, είναι θεμιτή η χρήση μιας αρχικοποιημένης σημαίας (initialized flag). Ο ορισμός της σημαίας θα πρέπει να είναι η τελευταία ενέργεια του κατασκευαστή πριν την επιτυχή ολοκλήρωση της εκτέλεσής του. Για παράδειγμα:

```
public abstract class ClassLoader {

     //ορισμός σημαίας
     private volatile boolean initialized;

     protected ClassLoader() {

          // χρειάζεται άδεια για τη δημιουργία ClassLoader
          securityManagerCheck();
          init();

          // Τελευταία ενέργεια του κατασκευαστή
          this.initialized = true;
     }

     protected final Class defineClass(...) {
          //έλεγχος κατάστασης σημαίας
          checkInitialized();
               //...
          }

     private void checkInitialized() {
          if (!initialized) {
               throw new SecurityException("NonFinal not
initialized");
          }
     }
}
```

4.4.6 Serialization και Deserialization

Η διαδικασία serialization (σειριοποίησης ή αποτύπωσης σε σειριακή μορφή) παρέχει μια διεπαφή στις κλάσεις, η οποία παρακάμπτει τους μηχανισμούς ελέγχου πρόσβασης της γλώσσας Java. Κάνοντας μια κλάση serializable, ουσιαστικά δημιουργούμε μια δημόσια διεπαφή για όλα τα πεδία αυτής της κλάσης. Το serialization μπορεί, επίσης, να προσθέσει ένα κρυφό δημόσιο κατασκευαστή της κλάσης. Αυτό το ενδεχόμενο θα πρέπει να εξεταστεί κατά την προσπάθεια περιορισμού της αυτόματης κατασκευής αντικειμένων.

Όταν ένα αντικείμενο γίνει serialized, τότε οι μηχανισμοί ελέγχου πρόσβασης της Java παύουν να επιβάλλονται και οι επιτιθέμενοι μπορούν να προσπελάσουν ιδιωτικά πεδία ενός αντικειμένου, αναλύοντας τη serialized ροή του. Για το λόγο αυτό, δεν πρέπει να σειριοποιούνται κλάσεις οι οποίες περιέχουν ευαίσθητα δεδομένα.

Τρόποι αντιμετώπισης περιπτώσεων με ευαίσθητα δεδομένα σε serialized κλάσεις είναι οι παρακάτω:

- Ορισμός των ευαίσθητων πεδίων ως transient.
- Κατάλληλος ορισμός του πεδίου serialPersistentFields.
- Υλοποίηση writeObject και χρήση ObjectOutputStream.putField.
- Υλοποίηση writeReplace.
- Υλοποίηση της διεπαφής Externalizable.

Η διαδικασία deserialization (αποσειριοποίηση ή αναδόμηση από σειριακή μορφή) θα πρέπει να αντιμετωπίζεται ως διαδικασία κατασκευής αντικειμένου. Η διαδικασία deserialization δημιουργεί μια νέα εκδοχή της κλάσης, χωρίς την κλήση του κατασκευαστή αυτής της κλάσης. Θα πρέπει να γίνεται χρήση της ObjectInputStream.readFields για να προστατέψουμε τα περιεχόμενα της κλάσης, όπως στο παράδειγμα που ακολουθεί:

```
public final class ByteString implements java.io.Serializable
{
     private static final long serialVersionUID = 1L;
     private byte[] data;
     public ByteString(byte[] data) {

          // Δημιουργία αντιγράφου πριν την εκχώρηση
          this.data = data.clone();
     }

     private void readObject(java.io.ObjectInputStream in)
throws java.io.IOException, ClassNotFoundException {

          // Προστασία των περιεχομένων της κλάσης με χρήση
της readFields
          java.io.ObjectInputStreadm.GetField fields =
in.readFields();
          this.data = ((byte[])fields.get("data")).clone();
     }
     //...
}
```

Σε μια μέθοδο readObject, θα πρέπει να διενεργούμε τους ίδιους ελέγχους εισόδου, όπως και σε έναν κατασκευαστή. Επίσης, είναι προτιμότερο να δημιουργούμε αντίγραφα αντικειμένων που έχουν προκύψει από deserialization, πριν αυτά οριστούν ως εσωτερικά πεδία σε μια υλοποίηση της readObject, Για παράδειγμα:

```
public final class Nonnegative implements
java.io.Serializable {
     private static final long serialVersionUID = 1L;
     private int value;

     public Nonnegative (int value) {
          // έλεγχος πριν την εκχώρηση
          this.data = nonnegative(value);
     }

     private static int nonnegative(int value) {
          if (value < 0) {
               throw new IllegalArgumentException (value + "
is negative");
          }
     return value;
     }

     // έλεγχος εισόδου
     private void readObject(java.io.ObjectInputStream in)
throws java.io.IOException, ClassNotFoundException {
          java.io.ObjectInputStreadm.GetField fields =
          in.readFields();
          this.value = nonnegative(field.get(value, 0));
     }

     //...
}
```

Αν μια serializable κλάση επιβάλει τον έλεγχο του SecurityManager κατά τη χρήση του κατασκευαστή της, τότε θα πρέπει να επιβάλλουμε τον ίδιο έλεγχο σε μια μέθοδο readObject ή readObjectData. Για παράδειγμα:

```
public final class SensitiveClass implements
java.io.Serializable {
     public SensitiveClass() {

          /*
          Χρειάζεται άδεια για να γίνει instantiation του
SensitiveClass. Επιβολή ελέγχου από το securityManager
          */

          securityManagerCheck();
          // ...
     }

     /*
     Υλοποίηση readObject για επιβολή ελέγχων κατά τη
διάρκεια της αποσειριοποίησης. Ο ίδιος έλεγχος επιβάλλεται
και για την readObject
     /*

    private void readObject(java.io.ObjectInputStream in) {

        // διπλός έλεγχος από κατασκευαστή
        securityManagerCheck();
          //...
     }
}
```

Αν μία serializable κλάση επιτρέπει, μέσω μιας public μεθόδου, αλλαγή της εσωτερικής της κατάστασης και η αλλαγή αυτή ελέγχεται από τον SecurityManager, τότε θα πρέπει επίσης να επιβληθεί ο αντίστοιχος έλεγχος σε μια υλοποίηση της μεθόδου readObject. Για παράδειγμα:

```
public final class SecureName implements java.io.Serializable
{

     // Ιδιωτική εσωτερική κατάσταση
     Private String name;
     private static final String DEFAULT = "DEFAULT";

     public SecureName() {

     // Απόδοση αρχικής τιμής στο πεδίο name
     name = DEFAULT;

/*
Επιτρέπεται στους καλούντες να αλλάζουν την ιδιωτική
εσωτερική κατάσταση. Η δημόσια μέθοδος SetName μπορεί να
επιφέρει αλλαγή στην εσωτερική κατάσταση της κλάσης
SecureName
*/

     public void setName(String name) {
```

```
            if (name!=null ? name.equals(this.name): (this.name
== null)) {

            // Καμία αλλαγή
            return;
            } else {
                  /*
                  Χρειάζεται άδεια για αλλαγή ονόματος. Επιβολή
ελέγχου από το securityManager
                  */
                  securityManagerCheck();
                  inputValidation(name);
                  this.name = name;
            }
      }


      /* Υλοποίηση readObject για επιβολή ελέγχων κατά τη
διάρκεια της αποσειριοποίησης
      private void readObject(java.io.ObjectInputStream in) {
            java.io.ObjectInputStream.GetField fields =
            in.readFields();
            String name = (String) fields.get("name", DEFAULT);

            /*
            Αν το αποσειριοποιημένο όνομα δεν ταιριάζει με την
εξ ορισμού τιμή που κανονικά δημιουργήθηκε κατά το χρόνο
κατασκευής, επανάληψη ελέγχων
            */
            if (!DEFAULT.equals(name)) {
                  securityManagerCheck();
                  inputValidation(name);
            }
            this.name = name;
      }
}
```

4.4.7 Έλεγχος πρόσβασης

Ένα ενδιάμεσο (cached) αποτέλεσμα δεν πρέπει να αποστέλλεται σε μια κλάση ή μέθοδο, η οποία δεν έχει τα απαραίτητα προνόμια να το δημιουργεί. Επειδή ο υπολογισμός των προνομίων μπορεί να περιέχει σφάλματα, είναι θεμιτή η χρήση του API AccessController για να επιβάλλουμε τον περιορισμό, όπως στο παράδειγμα που ακολουθεί:

```
private static final Map cache;
public static Thing getThing(String key) {
      // Try cache.
      CacheEntry entry = cache.get(key);
      if (entry != null) {

      /*
      Επιβεβαίωση ότι υπάρχουν οι απαιτούμενες άδειες πριν την
επιστροφή του cached αποτελέσματος.
      */
```

```
        AccessController.checkPermission(entry.getPermission());
        return entry.getValue();
        }

        // Επιβεβαίωση ότι δεν γίνεται αναβάθμιση αδειών

        Permission perm = getPermission(key);
        AccessController.checkPermission(perm);

        // Δημιουργία νέας τιμής με ακριβείς άδειες

        PermissionCollection perms =
    perm.newPermissionCollection();
        perms.add(perm);
        Thing value = AccessController.doPrivileged(
              new PrivilegedAction<Thing>() {
                    public Thing run() {
                          return createThing(key);
                    }
              },
              new AccessControlContext(
                    new ProtectionDomain[] {
                          new ProtectionDomain(null, perms)
                    }
              )
        );
        cache.put(key, new CacheEntry(value, perm));
        return value;
    }
```

Βιβλιογραφία

Checklist: Security Review for Managed Code. (n.d.). Retrieved 30 September 2015, from https://msdn.microsoft.com/en-us/library/ff648189.aspx

Gong, L., Ellison, G., & Dageforde, M. (2003). Inside Java 2 platform security: architecture, API design, and implementation (2nd ed). Boston: Addison-Wesley.

Long, F. (2014). Java coding guidelines: 75 recommendations for reliable and secure programs. Upper Saddle River, NJ: Addison-Wesley.

Long, F., & Carnegie-Mellon University (Eds.). (2012). The CERT Oracle secure coding standard for Java. Upper Saddle River, NJ: Addison-Wesley.

Secure Coding Guidelines for Java SE. (n.d.). Retrieved 30 September 2015, from http://www.oracle.com/technetwork/java/seccodeguide-139067.html

Secure Coding Principles - OWASP. (n.d.). Retrieved 30 September 2015, from https://www.owasp.org/index.php/Secure_Coding_Principles

The Java® Language Specification. (n.d.). Retrieved 30 September 2015, from https://docs.oracle.com/javase/specs/jls/se8/html/

The Java® Virtual Machine Specification. (n.d.). Retrieved 30 September 2015, from https://docs.oracle.com/javase/specs/jvms/se8/html/

Κριτήρια αξιολόγησης

Απαντήστε στις ακόλουθες ερωτήσεις. Η κάθε ερώτηση μπορεί να έχει μοναδική ή περισσότερες απαντήσεις.

1. Ποια γλώσσα διαθέτει εγγενείς αδυναμίες που προκαλούν υπερχείλιση ενταμιευτήρα;
α) Java
β) C
γ) Python
δ) HTML

2. Ποιοι οργανισμοί εκδίδουν αρχές ασφαλούς διαδικτυακού προγραμματισμού;
α) CERT, Carnegie Mellon
β) NIST
γ) OWASP
δ) Όλοι οι παραπάνω

3. Η αντιμετώπιση "security by obscurity'
α) επιλύει όλα τα προβλήματα.
β) επιλύει τα προβλήματα, μέχρι να «ανακαλυφθούν» τα υπάρχοντα σφάλματα.
γ) είναι αρχή ασφαλούς προγραμματισμού.
δ) Δεν ισχύει κάποιο από τα παραπάνω.

4. Ποιο από τα παρακάτω δεν είναι κατηγορία ευπάθειας;
α) Υπερχείλιση ενταμιευτήρα.
β) Χρήση κρυπτογραφικών μηχανισμών.
γ) Συνθήκες ανταγωνισμού.
δ) Μη επαληθευμένη είσοδος χρήστη.

5. Μια επίθεση υπερχείλισης ενταμιευτήρα εκμεταλλεύεται την αποθήκευση σε:
α) stack.
β) heap.
γ) buffer.
δ) όλα τα παραπάνω.

6. Κατά τον έλεγχο εισόδου, ελέγχουμε:
α) τα όρια τιμών.
β) το μήκος της εισόδου.
γ) την ύπαρξη null bytes.
δ) κανένα από τα παραπάνω.

7. Ο περιορισμός προνομίων στη Java επιτελείται από το:
α) java.securityManager.AccessController.doPrivileged
β) java.security.AccessManager.doPrivileged
γ) java.security.AccessController.doPrivileged
δ) java.security.AccessController.restrictPrivileged

8. Ποια από τα παρακάτω απαιτούν ιδιαίτερη προσοχή:
α) Δημιουργία BMP αρχείων.

β) Χρήση υπερσυμπιεσμένων αρχείων.
γ) Εκφράσεις XPath.
δ) Όλα τα παραπάνω.

9. Κατά τη δυναμική δημιουργία SQL εντολών είναι θεμιτή η χρήση:
α) java.sql.StealthStatement
β) java.sql.PreparedStatement
γ) java.sSecureSQL.PreparedStatement
δ) java.sql.Statement

10. Μηχανισμός ασφάλειας της Java είναι το:
α) SecurityManager
β) SecurityGuardManager
γ) SecurityGuard
δ) TotalSecurityManager