Ασφάλεια Συστημάτων

Τζέριες Μπεσαράτ, PhD

8 Απριλίου 2022, Άρτα

Στο προηγούμενο μάθημα αναπτύξαμε

- Προφίλ επιτιθεμένων
  - Hackers
  - Script Kiddies
  - Κυβερνο-κατάσκοποι
  - Κυβερνο-εγκληματίες
  - Insiders
- Μεθοδολογία επίθεσης
- Αμυντικά μοντέλα
  - Lollipop model
  - Onion model
- Ασφάλεια περιμέτρου
  - Τείχος προστασίας
  - Είδη Firewall
  - Bastion host

Προγραμματισμός στο Διαδίκτυο

- 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.

- 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.

Σε αυτό το μάθημα θα δούμε

- Αρχές Ασφαλούς Προγραμματισμού
- Κατηγορίες Ευπαθειών
  - 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)
- Μελέτη Περίπτωσης: Java

Εισαγωγή

Βασική αρχή: δεν υπάρχει ασφαλές πληροφοριακό σύστημα

Υπάρχουν οδηγίες και κατευθυντήριες γραμμές, τις οποίες οφείλουμε να ακολουθήσουμε προκειμένου να επιτύχουμε τον υψηλότερο βαθμό ασφάλειας, ανάλογα με τα μέσα που διαθέτουμε.

Web application

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.

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).

Οι κυριότεροι λόγοι για τους οποίους οι προγραμματιστές καταλήγουν στο να γράψουν κώδικα που περιέχει ή προκαλεί ευπάθειες στη διαδικτυακή εφαρμογή, είναι:

- 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

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:

Αρχή της Ελάχιστης Επιφάνειας Επίθεσης (Minimum Attack Surface Principle)

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.

Ρυθμίσεις ασφάλειας εξ ορισμού

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.

Αρχή του Ελάχιστου Προνομίου (Principle of Least Privilege)

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.

Μηχανισμοί ασφάλειας

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.

Παράδειγμα

```
/*
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());
    }
```

Προσοχή στην εμπιστοσύνη

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.

Αρχή του Διαχωρισμού Καθηκόντων (Separation of Duties)

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.

Security by obscurity

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».

Αρχής της Απλότητας

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)

Κατηγορίες Ευπαθειών

- 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)

Μελέτη περίπτωσης: Java

Γλώσσα Προγραμματισμού Java

Τι γνωρίζετε;

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)

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
- Ασφάλεια από την αρχή
- Περιορισμός Προνομίων
- Καθορισμός ορίων εμπιστοσύνης
- Αντοχή σε επιθέσεις άρνησης εξυπηρέτησης

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).

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 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));
     }
}
```

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();
     }
}
```

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();
    }
}
```

Μελέτη Περίπτωσης Java

- Ευαίσθητα δεδομένα
- Ψεκασμός εντολών
- Προσβασιμότητα και επεκτασιμότητα
- Επαλήθευση εισόδου
- Κατασκευή αντικειμένων
- Serialization και Deserialization
- Έλεγχος πρόσβασης

Ψεκασμός εντρολών - αποφυγή

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

Προσβασιμότητα κ Επεκτασιμότητα (1/2)

```
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.");
  }
```

Προσβασιμότητα κ Επεκτασιμότητα (2/2)

```
//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);
    }
}
```

Επαλήθευση εισόδου

```
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();

        /*
```

Επαλήθευση εισόδου

```
            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);
      }
}
```

Κατασκευή αντικειμένων

```
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;
     }
}
```

Έλεγχος πρόσβασης

```
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();
     }
```