WebGoat

Το WebGoat αποτελεί μία τοπική πλατφόρμα testing και εκμάθησης για θέματα ασφαλείας διαδικτύου. Εντός του project υπάρχουν (κατηγοριοποιημένα) τεράστια κενά ασφαλείας τα οποία ο χρήστης καλείται αρχικά να ανακαλύψει και έπειτα να καλύψει. Για τις ανάγκες της παρούσας εργασίας χρησιμοποιήσαμε την έκδοση 'WebGoat 5.2 Developer Version' την οποία και τρέξαμε μέσω του περιβάλλοντος του Eclipse.

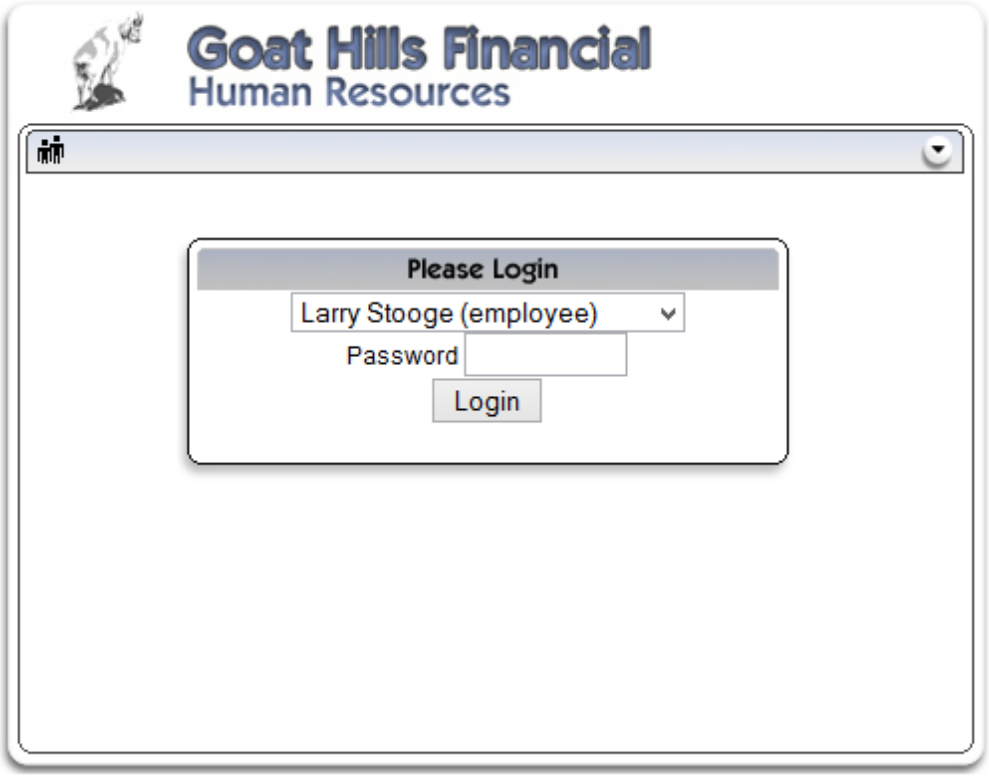
SQL Injection

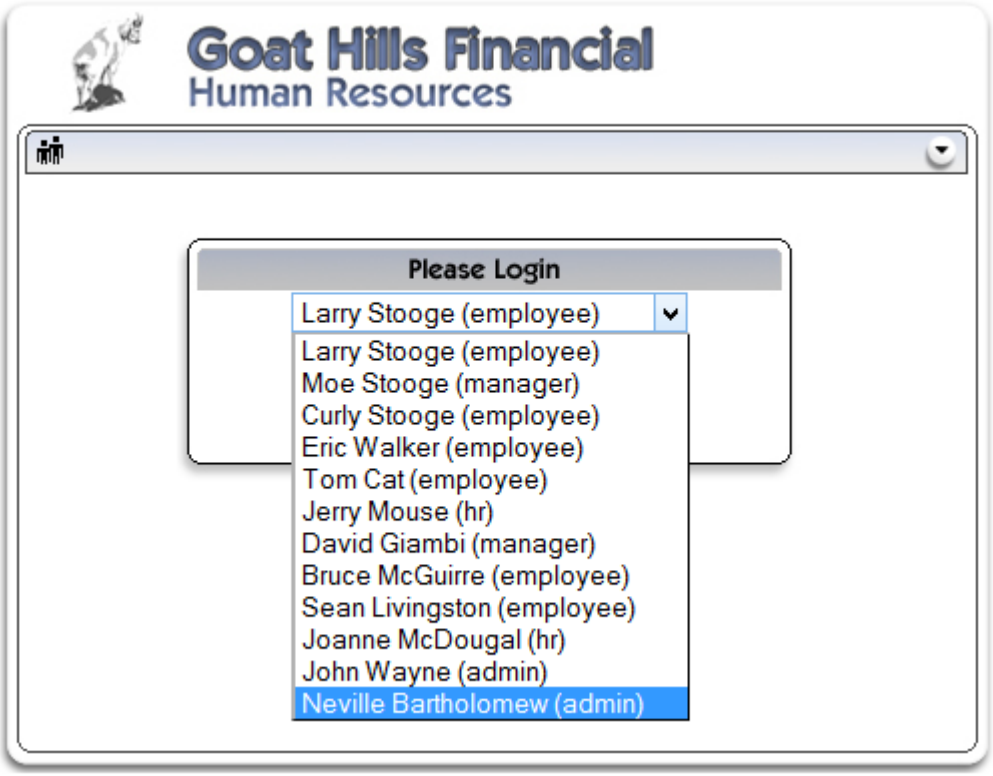
Γνωστή και συνήθης μέθοδος των hacker για παραβίαση ιστοσελίδων με κενά ασφαλείας είναι τα 'sql injection'. Ο hacker προσπαθεί μέσω queries που θέτει στη βάση δεδομένων της ιστοσελίδας, να την 'τρυπήσει' και να αποκτήσει πρόσβαση σε ιδιωτικά δεδομένα ή διαχειριστικές δυνατότητες που δεν του αναλογούν νόμιμα.

Παράδειγμα SQL Injection στο WebGoat

Η σελίδα που θα επιχειρήσουμε να ασφαλίσουμε από κενά ασφαλείας στην συγκεκριμένη εργασία, είναι η υποθετική σελίδα μελών μίας υποθετικής εταιρίας οικονομικών η οποία μπορεί να παραβιαστεί με δύο τρόπους: τόσο με String SQL Injection, όσο και με Numeric SQL Injection.

Στην παρακάτω εικόνα, μπορούμε να δούμε την κεντρική φόρμα σύνδεσης των μελών. Όπως βλέπουμε στην αμέσως επόμενη εικόνα, διατίθεται ένα dropdown menu με το όνομα και τον ρόλο του.





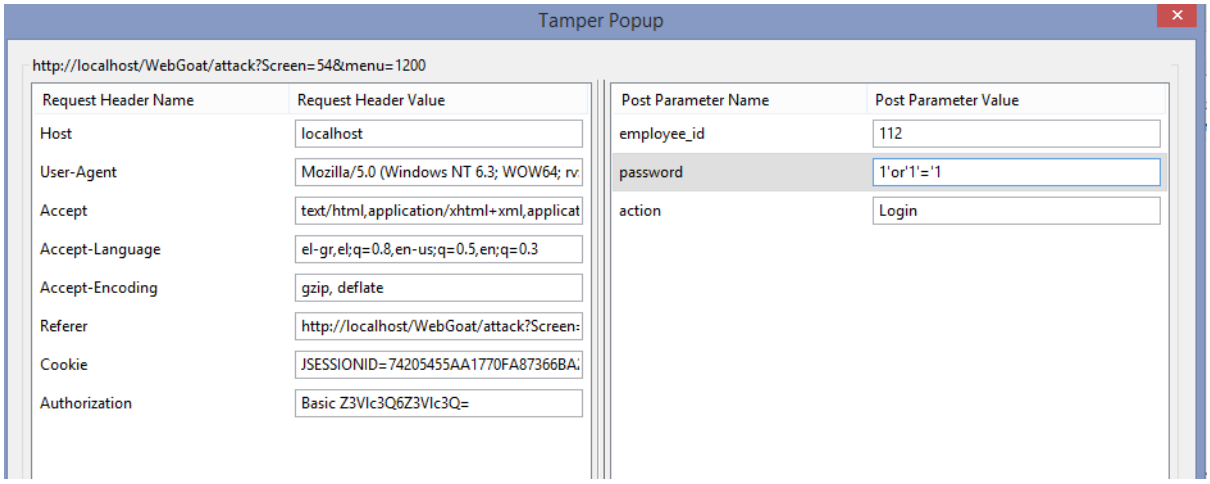
String SQL Injection

Σε πρώτη φάση θα προσπαθήσουμε να παραβιάσουμε ως απλοί επισκέπτες το σύστημα, εισχωρώντας στο σύστημα ως διαχειριστές, παραβιάζοντας τον λογαριασμό του διαχειριστή Neville Bartholomew μέσω String SQL Injection.

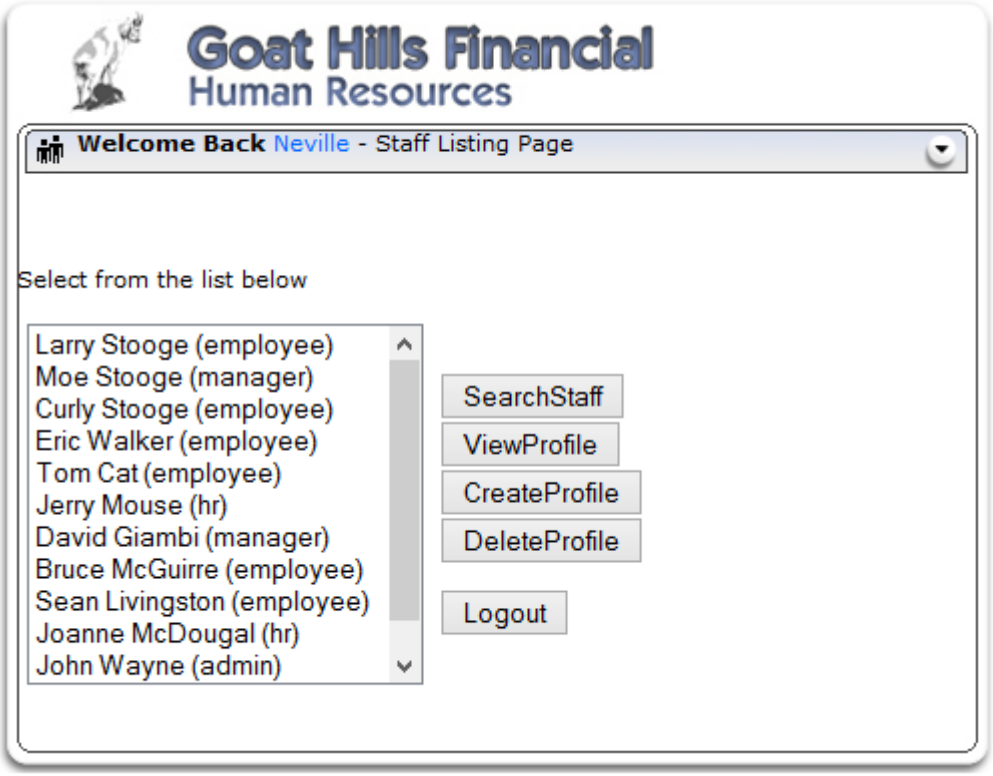
Για τις ανάγκες του 'hacking' που θα επιχειρήσουμε, θα χρειαστούμε το πρόσθετο εργαλείο "Tamper Data" το οποίο δίνει τη δυνατότητα για προβολή, καταγραφή και τροποποίηση εξερχόμενων http αιτημάτων και μπορεί να χρησιμοποιηθεί ως ένα εργαλείο pen-test ελέγχου κενών ασφαλείας ιστοσελίδων και διαδικτυακών εφαρμογών.

Επιλέγουμε λοιπόν, να συνδεθούμε ως 'Neville Bartholomew' και πριν πατήσουμε 'Login' επιλέγουμε Start Tamper στο Tamper Data (μενού "Εργαλεία" του Firefox). Πατώντας λοιπόν 'Login' το Tamper Data μας ζητά τις παραμέτρους που θέλουμε να βάλουμε στα πεδία και μάλιστα όχι κρυπτογραφημένα.

Εδώ λοιπόν, εμείς, θα χρησιμοποιήσουμε μία αρκετά συνήθη τακτική hacking κατά την οποία θα επιστρέψουμε ως ερώτημα στη βάση (στο πεδίο του κωδικού) μία πάντοτε αληθή συνθήκη (δηλαδή ότι 1 = 1) κατά την οποία το σύστημα (εφόσον είναι ανασφαλές και δεν έχει μεριμνήσει επί τούτου) θα δεχτεί ως αληθή και αποδεκτή, οπότε και εμείς θα συνδεθούμε στον λογαριασμό του διαχειριστή, χωρίς να γνωρίζουμε καν τον κωδικό! Συγκεκριμένα, θα δώσουμε στο πεδίο 'password' την τιμή «1' or '1' = '1» και ως αποτέλεσμα είναι να περάσει στο πεδίο του κωδικού αυτό ακριβώς το string. Το γιατί 'πέρασε' θα το δούμε παρακάτω.



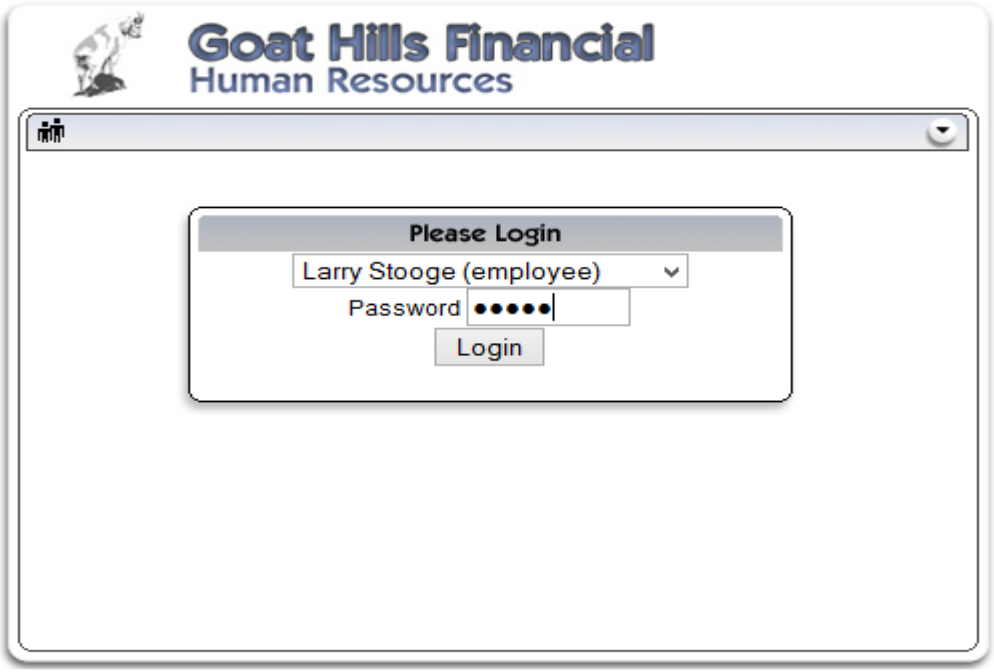
Στην παρακάτω εικόνα βλέπουμε ότι όντως καταφέραμε (χωρίς καν να διαθέτουμε λογαριασμό στο σύστημα) να συνδεθούμε με λογαριασμό διαχειριστή με τον οποίο μπορούμε να κάνουμε ότι θέλουμε (ακόμα και να διαγράψουμε) τους λογαριασμούς των στελεχών της εταιρίας.



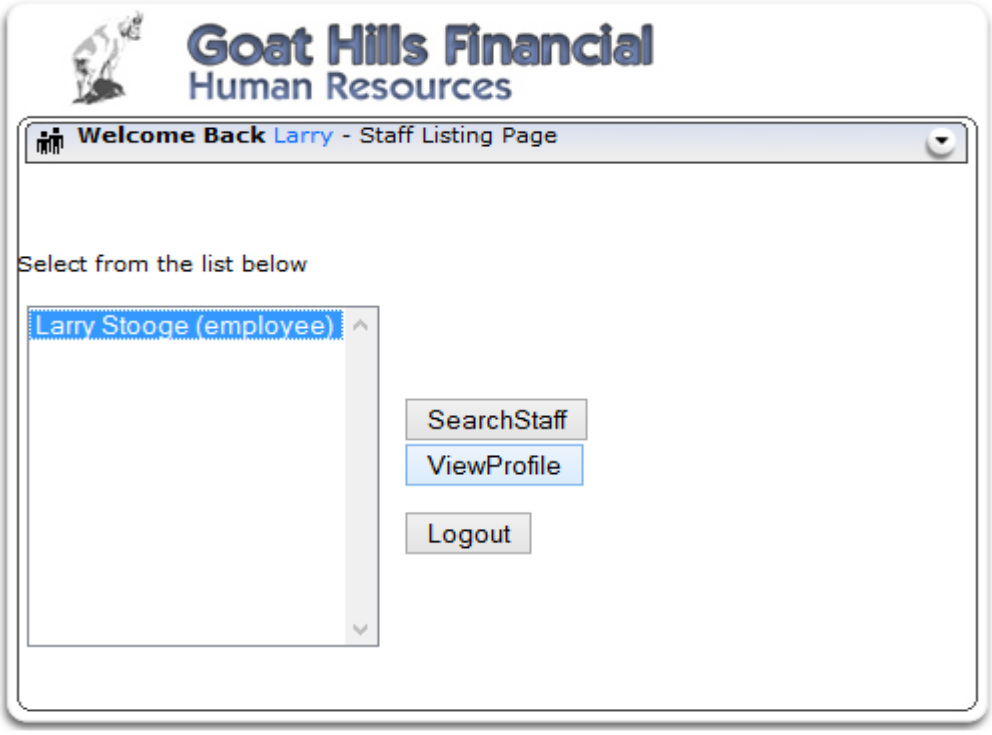
Numeric SQL Injection

Ο δεύτερος τρόπος με τον οποίον θα προσπαθήσουμε να παραβιάσουμε το σύστημα, είναι να συνδεθούμε με απλό ρόλο εργαζομένου και μέσα από εκεί να δούμε το προφίλ κάποιου διαχειριστή, στο οποίο υπό κανονικές συνθήκες δεν θα είχαμε πρόσβαση μέσω Numeric SQL Injection.

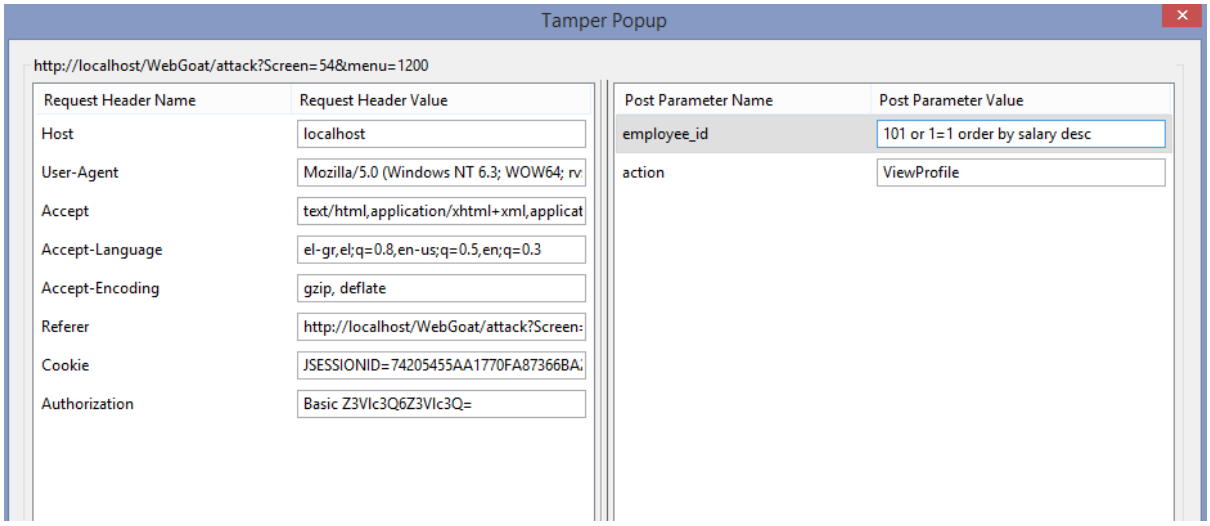
Αρχικά συνδεόμαστε με τον λογαριασμό μας: Υποτίθεται ότι είμαστε ο Larry (με κωδικό πρόσβασης “Larry”) και θέλουμε να δούμε τα στοιχεία του αφεντικού μας (Neville Bartholomew) με σκοπό να τα υποκλέψουμε (ακόμα και τον αριθμό της πιστωτικής του κάρτας).



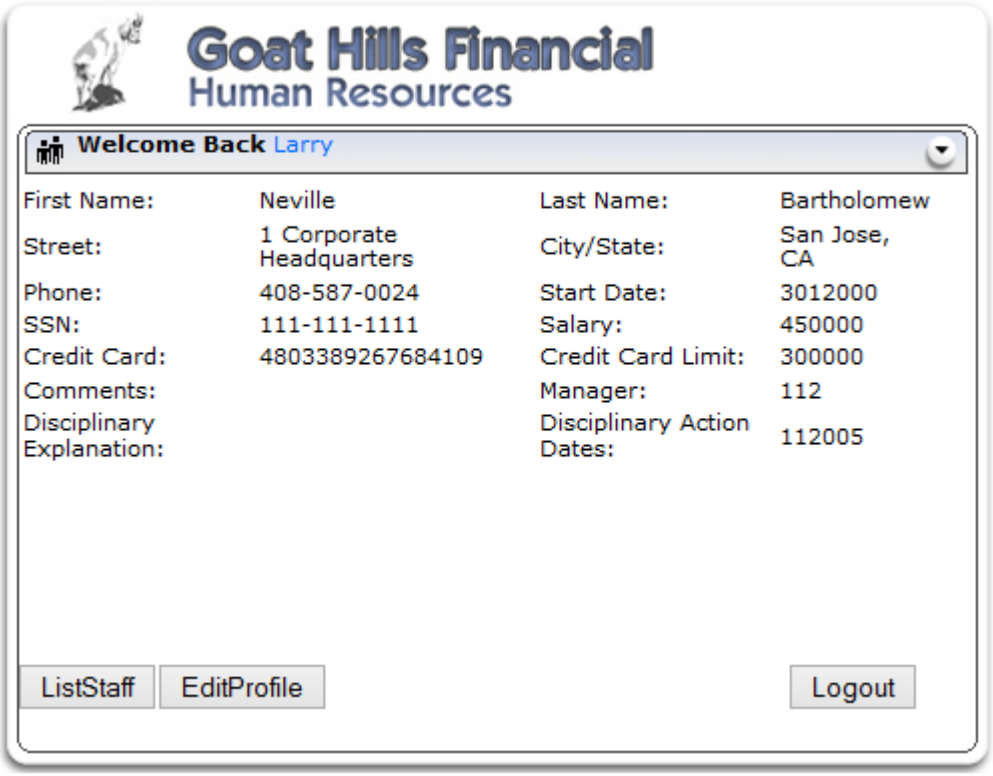
Εφόσον συνδεθούμε, έχοντας επιλεγμένο το προφίλ μας, κάνουμε ξανά ‘Start Tamper’ και επιλέγουμε ‘View Profile’. Υπό κανονικές συνθήκες, θα έπρεπε να δούμε το προφίλ μας, μέσω του Tamper Data όμως θα προσπαθήσουμε να ορίσουμε δικές μας παραμέτρους ούτως ώστε να εισχωρήσουμε σε άλλο προφίλ!



Λειτουργώντας με παρόμοιο τρόπο, θα ορίσουμε μία μονίμως αληθή συνθήκη, αυτή τη φορά στο πεδίο 'employee_id'. Συγκεκριμένα, θα επιστρέψουμε στη βάση το ερώτημα «101 or 1=1 order by salary desc» το οποίο, έχοντας υποθέσει ότι υπάρχει το πεδίο salary στη βάση (δηλαδή ο μισθός), με φθίνουσα ταξινόμηση θα μας επιστρέψει τον κύριο Bartholomew ο οποίος έχει και τον μεγαλύτερο μισθό στην εταιρία.



Όντως, παρακάτω βλέπουμε ότι καταφέραμε με επιτυχία –συνδεδεμένοι ως απλοί εργαζόμενοι- να δούμε το προφίλ του διαχειριστή του συστήματος, στο οποίο υπάρχουν σημαντικά και ευαίσθητα προσωπικά στοιχεία. Μάλιστα, μας δίνεται ακόμα και δυνατότητα τροποποίησης του προφίλ αυτού!



Διάγνωση προβλήματος – κενών ασφαλείας

Για να ανακαλύψουμε τα κενά ασφαλείας του κώδικα αυτής της σελίδας θα πρέπει να ανατρέξουμε στα αρχεία Login.java και ViewProfile.java αντίστοιχα. Στον κώδικα αυτών των αρχείων θα πρέπει να δούμε σε ποιο σημείο γίνεται η διασύνδεση με τη βάση, πως γίνεται η διασύνδεση αυτή – με ποιο τρόπο περνάει ο κώδικας το query στην βάση δεδομένων – τι παραμέτρους επιτρέπει ο κώδικας να περάσουν και να διορθώσουμε τις ατέλειες.

Login.Java

Στις γραμμές 125-131 και 160-166 του συγκεκριμένου αρχείου βλέπουμε τον παρακάτω κώδικα:

```
String query = "SELECT * FROM employee WHERE userid = " + userId + " and
password = '" + password + "'";
// System.out.println("Query:" + query);
try
{
Statement answer_statement = WebSession.getConnection(s)

.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_READ_ONLY);
ResultSet answer_results =
answer_statement.executeQuery(query);
```

Όπως βλέπουμε μέσα στο SQL Query δεν χρησιμοποιεί παραμέτρους, ενώ αφήνει οποιοδήποτε string να 'περάσει', κάτι που είναι εντελώς ανασφαλές. Έτσι κι εμείς προηγουμένως μέσα στα «αυτάκια» (") κατορθώσαμε να περάσουμε το string που θέλαμε (Το τελικό ερώτημα που θα καταλάβει η βάση, θα είναι δηλαδή SELECT * FROM employee WHERE userid = 101 and password = 1 or 1=1.

ViewProfile.Java

Στις γραμμές 100-108 του αρχείου, βλέπουμε τον παρακάτω κώδικα:

```
String query = "SELECT employee.* "
+ "FROM employee,ownership WHERE employee.userid =
ownership.employee_id and "
+ "ownership.employer_id = " + userId + " and ownership.employee_id
= " + subjectUserId;
try
{
Statement answer_statement = WebSession.getConnection(s)
```

```
.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
ResultSet.CONCUR_READ_ONLY);

ResultSet answer_results = answer_statement.executeQuery(query);
```

Επίσης στις γραμμές 151-157 τον παρακάτω κώδικα:

```
String query = "SELECT * FROM employee WHERE userid = " + subjectUserId;

                    try

                    {

                            Statement answer_statement = WebSession.getConnection(s)


        .createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_READ_ONLY);

                            ResultSet answer_results =
answer_statement.executeQuery(query);
```

Οι γραμμές αυτές αποτελούν κώδικα της μεθόδου getEmployeeProfile που επιστρέφει το προφίλ του συνδεδεμένου χρήστη. Ουσιαστικά συμβαίνει ότι και πριν, απλά εδώ περιμένει να του δώσει κάποιο νούμερο.

Επίλυση – κάλυψη κενών ασφαλείας

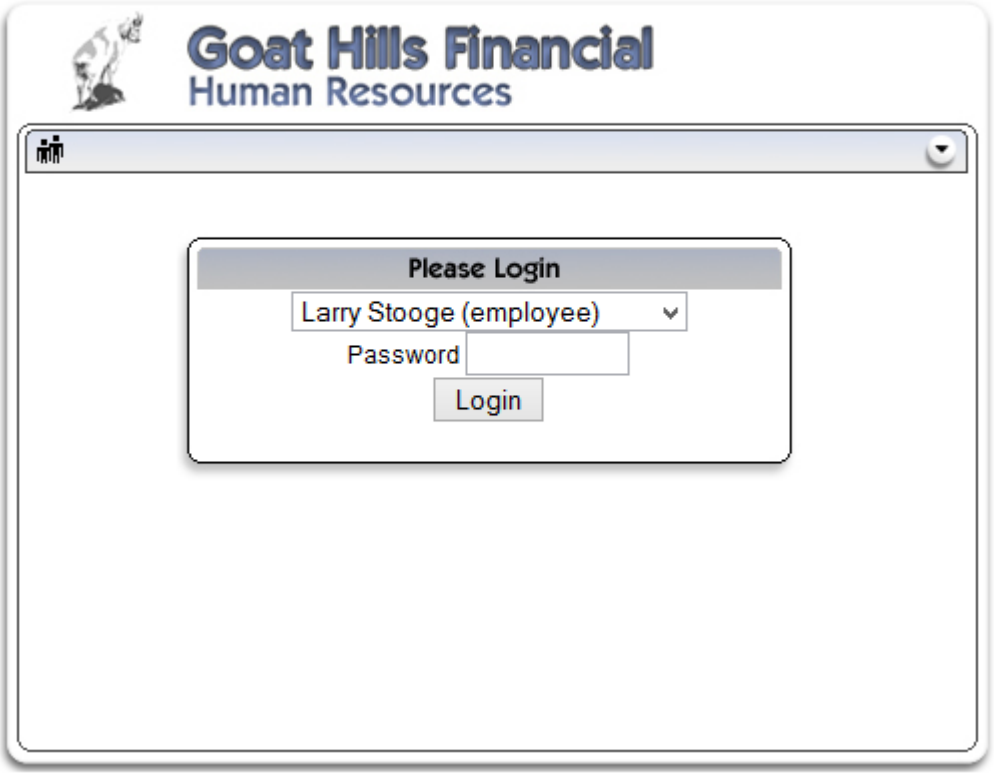
Για να καλύψουμε αυτά τα κενά ασφαλείας, θα πρέπει να ‘κλειδώσουμε’ τον κώδικα μας από το να δέχεται οποιασδήποτε μορφής παραμέτρους, φιλτράροντας τις εισερχόμενες τιμές.

Παρακάτω είναι ο διορθωμένος κώδικας για την πρώτη περίπτωση (string injection), τον οποίο και θα εξηγήσουμε στη συνέχεια.

```
String query = "SELECT * FROM employee WHERE userid = ? and password = ?";

try

{

        Connection connection = WebSession.getConnections(s);

        PreparedStatement statement = connection.prepareStatement(query,
ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_READ_ONLY);

        statement.setString(1, userId);

        statement.setString(2, password);

        ResultSet answer_results = statement.executeQuery();
```

Όπως βλέπουμε στο SQL Query εκεί που ήταν τα «αυτάκια» τώρα υπάρχουν τα «ερωτηματικά». Ο κώδικας πάει και παίρνει τις δοθείσες τιμές από το statement και τις αντικαθιστά στα 1 και 2 (πρώτο και δεύτερο ερωτηματικό). Εντός του statement εμπεριέχονται μέθοδοι ελέγχου που κόβουν τις παραμέτρους, όπως αυτές που είδαμε πριν.

* Login failed



Η παραπάνω εικόνα είναι το αποτέλεσμα προσπάθειας String SQL Injection, με τον ίδιο ακριβώς τρόπο όπως πριν, η οποία όμως απέτυχε παταγωδώς κατόπιν της ανανέωσης και ασφάλισης του κώδικα μας.

Ακολούθως θα ασφαλίσουμε την σελίδα και για Numeric SQL Injection. Παρακάτω παρατίθεται ο κώδικας, ο οποίος εξηγείται αμέσως μετά.

```
String query = "SELECT employee.* "
   + "FROM employee,ownership WHERE employee.userid = ownership.employee_id and "
   + "ownership.employer_id = ? and ownership.employee_id = ?";
try
{
  Connection connection = WebSession.getConnections(s);
  PreparedStatement statement = connection.prepareStatement(query,
ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_READ_ONLY);
          statement.setInt(1, Integer.parseInt(userId));
          statement.setInt(2, Integer.parseInt(subjectUserId));
  ResultSet answer_results = statement.executeQuery();
```

Με παρόμοιο τρόπο, όπως και πριν, περνάνε οι αριθμητικές τιμές που δίνει ο χρήστης (στον παλιό ανασφαλή κώδικα μπορούσε να περάσει και string). Η παρακάτω εικόνα είναι το αποτέλεσμα αποτυχημένης προσπάθειας Numeric SQL Injection, κατόπιν της ασφάλισης του κώδικας μας.