# Open eClass 2.3

Το repository αυτό περιέχει μια **παλιά και μη ασφαλή** έκδοση του eclass.
Προορίζεται για χρήση στα πλαίσια του μαθήματος
Προστασία & Ασφάλεια Υπολογιστικών Συστημάτων (ΥΣ13), **μην τη χρησιμοποιήσετε για κάνενα άλλο σκοπό**.

## Χρήση μέσω docker

```
# create and start (the first run takes time to build the image)
docker-compose up -d

# stop/restart
docker-compose stop
docker-compose start

# stop and remove
docker-compose down -v
```

Το site είναι διαθέσιμο στο http://localhost:8001/. Την πρώτη φορά θα πρέπει να τρέξετε τον οδηγό εγκατάστασης.

## Ρυθμίσεις eclass

Στο οδηγό εγκατάστασης του eclass, χρησιμοποιήστε **οπωσδήποτε** τις παρακάτω ρυθμίσεις:

- Ρυθμίσεις της MySQL
  - Εξυπηρέτης Βάσης Δεδομένων: `db`
  - Όνομα Χρήστη για τη Βάση Δεδομένων: `root`
  - Συνθηματικό για τη Βάση Δεδομένων: `1234`
- Ρυθμίσεις συστήματος
  - URL του Open eClass : `http://localhost:8001/` (προσοχή στο τελικό `/`)
  - Όνομα Χρήστη του Διαχειριστή : `drunkadmin`

Αν κάνετε κάποιο λάθος στις ρυθμίσεις, ή για οποιοδήποτε λόγο θέλετε να ρυθμίσετε το openeclass από την αρχή, διαγράψτε το directory, `openeclass/config` και ο οδηγός εγκατάστασης θα τρέξει ξανά.

# 2022 Project 1

Εκφώνηση: https://ys13.chatzi.org/assets/projects/project1.pdf

## Μέλη ομάδας

- 1115201800332, Νικόλας Ηλιόπουλος
- 1115201800022, Μιχάλης Βολάκης

# Defenses

### CSRF Token

Προσθέσαμε ένα CSRF Token για κάθε CRUD action μπορεί να γίνει στο site.

Έτσι αποτρέπουμε το κάθε site από το να γίνει κάποιο action (επειδή στάλθηκε ένα request) χωρίς να έχει ζητηθεί η φόρμα.
Για να γίνει ένα action δηλαδή θα πρέπει να ζητηθεί η φόρμα πρώτα!
Με την ζήτηση της φόρμας θα γίνει generate (και store) ένα token που θα σταλθεί μαζί με την φόρμα.
Όταν θα γίνει submit η φόρμα θα σταλθεί μαζί το token και θα ελέγξει αν είναι ίδιο με αυτό που αποθηκεύτηκε πριν στο server.
Κάθε φορά που γίνεται το check αυτό, γίνεται και regenarate του token.

Στο αρχείο **openeclass/kerberosclan/csrf_utils.php** υπάρχουν τα utils για το csrf token.
Και προσθέσαμε τον κώδικα(με τροποποιησεις) αυτό σε κάθε αρχείο που είχε form ή action που θέλαμε να ασφαλήσουμε.

```
if (!isset($_SESSION['first_entry'])) { // first time entering this page
in this session
    $csrf_token = start_csrf_session('csrf_token');  // generate a new
token and store it to session
    $_SESSION['first_entry'] = true;
} else {
  // all the actions that we want to defense
    if (
        isset($_REQUEST['new_assign']) ||
        isset($_REQUEST['download']) ||
        isset($_REQUEST['choice']) ||
        isset($_REQUEST['submit']) ||
        isset($_REQUEST['hide'])
    ) {
    // check if the token is valid
    // and if not destroy session and redirect to login page
        $csrf_token = check_csrf_attack('csrf_token',
$_REQUEST['csrf_token']);
    }
  // load the token in order to send it with the form
    $csrf_token = get_sessions_csrf_token('csrf_token');
}
```

### SQL Injection

Χρησιμοποιήσαμε **Prepare Statements**.
Συγκεκριμένα σε κάθε query που γίνεται δημιουργία,επεξεργασία ή διαγραφή στοιχείων από την βάση χρησιμοποιούμε την stmt_prepare της mysqli ώστε να ξεχωρίσουμε την εντολή της SQL από τα δεδομένα του χρήστη τα οποία γίνονται bind ως παράμετροι πριν εκτελεστεί το κάθε command. Σε παραμέτρους όπου μπορεί να προστεθεί κείμενο χρησιμοποιήσαμε επιπλέον την htmlspecialchars().

```
$statement = mysqli_stmt_init($connection);
    mysqli_stmt_prepare($statement, $query);
    mysqli_stmt_bind_param(
        $statement,
        "s",
        $user_id
    );
    mysqli_stmt_execute($statement);

    $result = mysqli_stmt_get_result($statement);
```

### XSS

Χρησιμοποιήσαμε το function **htmlspecialchars()** ώστε να αποτρέψουμε τον χρήστη από το να εισάγει ειδικούς χαρακτήρες και html tags όπως το script για να εισάγει κάποιο κακόβουλο script. Επιπλέον αλλάξαμε το PHP_SELF σε SCRIPT_NAME ώστε να μην εισάγεται κάποιο script μέσω του url της πλατφόρμας.

```
$title_temp = htmlspecialchars($row['title']);
```

### RFI

Για να αποτρέψουμε τις επιθέσεις απο RFI χρησιμοποιήσαμε τις εξής τεχνικές:

- Προσθέσαμε ένα randomly generated prefix στα filenames τα οποία γίνοντα upload/submit ώστε να μην μπορούν να προβλεφθούν.

```
$encoded_dropbox_filename = bin2hex(openssl_random_pseudo_bytes(32)) .
$dropbox_filename;
```

- Ως τελευταίο μέτρο αλλάξαμε τα permissions των αρχείων σε write only (0222) έτσι ώστε αν ο χρήστης εν τέλει καταφέρει να αποκτήσει πρόσβαση σε αυτά δεν θα μπορέσει να τα εκτελέσει.

## Attacks

### SQL Injection

Αρχικά προσπαθήσαμε να πειραματιστούμε με τα query params στα διάφορα module του μαθήματος. Είδαμε ότι στην περιοχή συζητήσεων του reply page (reply.php) όπου δίνεται το topic id μπορούμε να κάνουμε escape το sql string και να εκτελέσουμε το δικό μας select query.

Έτσι επιλέξαμε τον κωδικό του admin από το user table και είδαμε ότι εμφανίζεται στο breadcrumb. Δυστυχώς όμως ο κωδικός αποθηκεύεται στην βάση αφότου γίνει MD5 hash και δοκιμάσαμε να τον κάνουμε decrypt χωρίς όμως κάποια επιτυχία.

Προσπαθήσαμε να πειραματιστούμε με το login page εισάγοντας τα εξής credentials:

> username: drunkadmin
>
> password: ' OR 1=1;

ώστε να κάνουμε escape τα quotes από το select query. Απ'ότι φαινόταν όμως τα queries είχαν προστατευθεί με htmlspecialchars και prepared statements εφόσον δεν καταφέραμε να δούμε τα αποτελέσματα του select.

Δοκιμάσαμε επίσης να εκμεταλλευτούμε τα db queries που γίνονται στην υποβολή εργασιών και συγκεκριμένα στα σχόλια της εργασίας:

```
'); SELECT * FROM logins; --
'); DELETE FROM users; --
') OR 1=1; --
```

Σε όλες τις περιπτώσεις τα queries απέτυχαν επειδή η mysql_query δεν υποστηρίζει multiple statements οπότε το μόνο που εμφανιζόταν ήταν ότι υπήρχε syntax error στο query.

Παρατηρήσαμε επίσης ότι στην σελίδα για την αναβάθμιση της βάσης (upgrade/index.php) δεν χρειάζεται κάποιο session id διαχειριστή αλλά υπάρχει ξεχωριστό login page. Θεωρώντας απίθανο να έχει προστατευθεί και αυτό όσο το login στο index.php δοκιμάσαμε να κάνουμε και εκεί sql injection:

> username: ' OR user.user_id='1' OR '5'='2;
>
> password: ' OR user.user_id='1' OR '5'='2;

Πράγματι οδηγηθήκαμε στην σελίδα αναβάθμισης της βάσης και ανοίγοντας το eclass σε ξεχωριστό tab είδαμε ότι είχαμε admin access στο μάθημα. Επομένως μπορούσαμε να επεξεργαστούμε το μάθημα, να ενεργοποιήσουμε/απενεργοποιήσουμε εργαλεία κτλ. Δυστυχώς η διαχείριση πλατφόρμας δεν ήταν ενεργοποιημένη άρα το session id ήταν για user εκπαιδευτή και όχι για τον admin της πλατφόρμας. Σε επόμενο attack καταφέραμε να πάρουμε admin priviledges με XSS.

## XSS

Παρατηρήσαμε ότι υπάρχει ευπάθεια στην περιοχή συζητήσεων του μαθήματος στο σημείο όπου δίνεται το query parameter topic στο reply.php όπου μπορούμε να κάνουμε escape το sql query και να εκτελέσουμε ένα JS script. Αρχικά δοκιμάσαμε να κάνουμε alert() για να ελέγξουμε αν εκτελείται το script και εφόσον είδαμε ότι εμφανίζεται το alert δοκιμάσαμε να κάνουμε redirect.

Συγκεκριμένα με το εξής URL:

> http://sloppy-buffoons-2.csec.chatzi.org/modules/phpbb/reply.php?topic=2 AND 10=12) UNION SELECT FROM eclass.user -- &forum=1

κάναμε redirect στο cookie stealer site μας (προσθέτοντας το PHPSESSIONID στο url). Με αυτόν τον τρόπο μπορούμε να αποκτήσουμε το cookie οποιουδήποτε επισκέπτεται αυτό το URL. Επομένως στείλαμε ένα email στον admin παρατρέποντας τον να επισκεφτεί το site και έτσι αποκτήσαμε admin priviledges από το session id του:

> PHPSESSID=2ggn0l7h3cgv3pdh47cvchh485

Το επόμενο βήμα ήταν να κάνουμε login ως admin χρησιμοποιώντας το cookie που κλέψαμε και να μπούμε στο eclassconf.php για να δούμε τον κωδικό της βάσης (κάνοντας inspect εφόσον το html element είχε το password type). Ο κωδικός της βάσης είναι ο ίδιος με αυτόν του admin οπότε αλλάξαμε τον κωδικό για να έχουμε πρόσβαση και μετά την λήξη του session id:

> Old password: CCV1Y8kJcj
>
> New password: abcdefg

### Deface

Έχοντας admin priviledges κάναμε τις εξής ενέργειες:

- Δοκιμάσαμε να χρησιμοποιήσουμε το "Ανέβασμα ιστοσελίδας" από την ενεργοποίηση εργαλείων όπου υπό κανονικές συνθήκες ο admin μπορεί να ανεβάσει μόνο html αρχεία. Εφόσον όμως ο έλεγχος για το file type γίνεται στον client απλά αφαιρέσαμε το allow attribute στο αντίστοιχο κουμπί και ανεβάσαμε ένα php file το οποίο εκτελείται από τον server. Η πρώτη μας σκέψη ήταν να δοκιμάσουμε να κάνουμε rename & unlink του contact.php έτσι ώστε να αντικαταστήσουμε την contact page με δικό μας περιεχόμενο. Δυστυχώς όλα τα αρχεία του apache δεν είχαν write permissions αλλά μόνο read+execute (755) και το chmod δεν επιτρεπόταν εφόσον δεν είχαμε το root password του vm. Επομένως αρκεστήκαμε στο να προσθέσουμε ένα extra site το οποίο έκανε redirect σε ένα δικό μας deface site από το puppies.

- Μπήκαμε στο phpmyadmin από το admin panel έτσι ώστε να εκτελέσουμε queries στην βάση χωρίς τους περιορισμούς(htmlspecialchars,intval) που υπήρχαν όταν κάναμε form submits. Η πρώτη μας ενέργεια ήταν να αλλάξουμε το description του μαθήματος με το εξής script:

  ```
  <script>
      document.location = 'https://bit.ly/3PcmMm8';
  </script>
  ```

  Κάθε φορά που ο χρήστης προσπαθεί να μπει στο μάθημα θα γίνεται αυτόματα redirect στο shortened link που δίνουμε οδηγώντας τον σε ένα defaced site του kerberosclan. Έτσι δεν θα έχει πλέον πρόσβαση στα εργαλεία του μαθήματος.

- Η τελευταία μας ενέργεια ήταν να αντικαταστήσουμε εξολοκλήρου την αρχική σελίδα με τον ίδιο τρόπο. Έτσι προσθέσαμε (από το phpmyadmin) ένα script στο description ενός admin announcement το οποίο φορτώνεται στο index.php και κάναμε redirect σε ένα GIF page με ένα skip button για να μην αποκλειστούμε εντελώς από την πλατφόρμα.

```
<script>
const urlParams = new URLSearchParams(window.location.search);
const param = urlParams.get("defaced");
if(!param){
document.location="http://kerberosclan.puppies.chatzi.org/skip.php"
```

```
}
</script>
```

Για να επιστρέψουμε στην πλατφόρμα ελέγχουμε αν υπάρχει το query param "skip" και αν ναι επιστρέφουμε τον χρήστη στο πραγματικό index.php page.

## Επίλογος

Από την στιγμή που είχαμε admin priviledges ήταν αναμενόμενο ότι θα μπορούσαμε να κάνουμε deface. Αν δεν είχαμε καταφέρει να μπούμε στα εργαλεία του admin θα δοκιμάζαμε κάποιο RFI ή CSRF attack ώστε να κάνουμε επιθέσεις με μικρότερο αντίκτυπο (πχ. διαγραφή εργασίας). Επομένως το cookie stealer page μας ήταν αυτό που μας έδωσε την δυνατότητα να κάνουμε τις μεγάλες αλλαγές (deface) στο site.

Επίσης συνειδητοποιήσαμε ότι τα περισσότερα είδη attacks είναι δύσκολα και αρκετά χρονοβόρα. Πόσο μάλλον σε περιπτώσεις όπου δεν γνωρίζουμε τον server-side code που εκτελείται ώστε να βρούμε κάποιο exploit.
Μπορούμε να φανταστούμε πόσο πιο δύσκολο θα ήταν να κάνουμε το deface στο site όταν δεν έχουμε ήδη γνώση του κώδικα που εκτελείται στον server όπως είχαμε στα πλαίσια της εργασίας.