ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΑΤΤΙΚΗΣ

ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΠΡΟΣΤΑΣΙΑ ΚΑΙ ΑΝΙΧΝΕΥΣΗ ΑΠΟ ΕΠΙΘΕΣΕΙΣ ΤΥΠΟΥ SQL INJECTION

ΠΑΝΑΓΙΩΤΟΠΟΥΛΟΣ ΕΥΑΓΓΕΛΟΣ

Επιβλέπουσα καθηγήτρια: Καντζάβελου Ιωάννα

Αθήνα

Ιούλιος 2021

ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΑΤΤΙΚΗΣ

ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΠΡΟΣΤΑΣΙΑ ΚΑΙ ΑΝΙΧΝΕΥΣΗ ΑΠΟ ΕΠΙΘΕΣΕΙΣ ΤΥΠΟΥ SQL INJECTION

Παναγιωτόπουλος Ευάγγελος – Α.Μ.: 44003

Επιβλέπουσα καθηγήτρια: Καντζάβελου Ιωάννα

Εγκρίθηκε απο την τριμελή εξεταστική επιτροπή στις 14 Ιουλίου 2021.

Εξεταστική Επιτροπή Διπλωματικής Εργασίας:

Ioanna Kantzavelou Digitally signed by Ioanna Kantzavelou Date: 2021.07.23 14:47:51 +03'00'

Καντζάβελου Ιωάννα	Λιμνιώτης Κώστας	Γεωργούλας Άγγελος
Επ. Καθηγήτρια ΠΑ.Δ.Α	Ακαδ. Υπότροφος ΠΑ.Δ.Α	Ακαδ. Υπότροφος ΠΑ.Δ.Α

ΔΗΛΩΣΗ ΣΥΓΓΡΑΦΕΑ ΔΙΠΛΩΜΑΤΙΚΗΣ ΕΡΓΑΣΙΑΣ

Ο κάτωθι υπογεγραμμένος Παναγιωτόπουλος Ευάγγελος του Κωνσταντίνου, με αριθμό μητρώου 44003, φοιτητής του Πανεπιστημίου Δυτικής Αττικής της Σχολής Μηχανικών του Τμήματος Μηχανικών Πληροφορικής και Υπολογιστών, δηλώνω υπεύθυνα ότι:

«Είμαι συγγραφέας αυτής της διπλωματικής εργασίας και ότι κάθε βοήθεια την οποία είχα για την προετοιμασία της είναι πλήρως αναγνωρισμένη και αναφέρεται στην εργασία. Επίσης, οι όποιες πηγές απο τις οποίες έκανα χρήση δεδομένων, ιδεών ή λέξεων, είτε ακριβώς είτε παραφρασμένες, αναφέρονται στο σύνολό τους, με πλήρη αναφορά στους συγγραφείς, τον εκδοτικό οίκο ή το περιοδικό, συμπεριλαμβανομένων και των πηγών που ενδεχομένως χρησιμοποιήθηκαν απο το διαδίκτυο. Επίσης, βεβαιώνω ότι αυτή η εργασία έχει συγγραφεί απο εμένα αποκλειστικά και αποτελεί προϊόν πνευματικής ιδιοκτησίας τόσο δικής μου, όσο και του Ιδρύματος. Παράβαση της ανωτέρω ακαδημαϊκής μου ευθύνης αποτελεί ουσιώδη λόγο για την ανάκληση του πτυχίου μου».

Ο δηλών

ΠΕΡΙΛΗΨΗ

Η προσπάθεια έγχυσης κώδικα με σκοπό να αποκτηθεί παράνομη πρόσβαση, είναι ένα απο τα πιο φλέγον ζητήματα στην κυβερνοασφάλεια και ειδικότερα στον τομέα του hacking. Σε αυτή την διπλωματική θα αναφερθούν κάποιες βασικές έννοιες έτσι ώστε και ένας αναγνώστης χωρίς εξειδικευμένες γνώσεις να μπορεί να συμβαδίζει με την εξέλιξη της διπλωματικής. Θα γίνει εκτεταμένη έρευνα και εμβάθυνση στο κομμάτι του SQL Injection, όπως για παράδειγμα σε ποιές κατηγορίες χωρίζεται ή πώς πραγματοποιείται μία τέτοια επίθεση. Στη συνέχεια θα προταθούν αρκετοί τρόποι προστασίας απο την επίθεση αυτή τόσο κατα την διάρκεια ανάπτυξης μιας ιστοσελίδας ή μιας βάσης δεδομένων, όσο και αφού έχει πέσει θύμα αυτής. Μετά ακολουθεί η ανίχνευση, όπου θα αναλυθεί πως μπορούμε να ανιχνεύσουμε ενδεχομένως, μία βάση δεδομένων ή μία ιστοσελίδα προκειμένου να μην είναι ευάλωτη και τέλος προσεγγίζεται μέσω μιας άλλης οπτικής η ανίχνευση της SQL Injection όπου θα αναπτυχθεί εργαλείο σε γλώσσα προγραμματισμού Python, που θα ανιχνεύει αν μία βάση δεδομένων είναι ευάλωτη στην συγκεκριμένη κατηγορία SQL Injection, error-based δηλαδή βασιζόμενο στα σφάλματα που μπορεί να επιστρέψει σαν αποτέλεσμα στον χρήστη μία βάση δεδομένων.

Λέξεις – κλειδιά:

SQL, SQL Injection, βάσεις δεδομένων, MySQL, Oracle, Microsoft SQL Server, Python

ABSTRACT

Attempting to inject code in order to gain illegal access is one of the hottest issues in cybersecurity and especially in the topic of hacking. In this thesis, some basic definitions will be mentioned, so that a reader without specialized knowledge can keep up reading and make sense. Extensive research will be done on the part of SQL Injection such as, in which categories it is divided or how an attack like this can occur. Later on, there will be suggested a lot of ways to protect a database during its development as well as after being victim from such an attack. Then follows the detection, where it will be analyzed how we can possibly detect if a database or a website is vulnerable and finally on the last chapter the detection of SQL Injection is approached through a different perspective where a tool written in Python will be developed and will try to detect if a database is vulnerable to the specific category of SQL Injection, error-based meaning that it is based on the errors that a database can return as a result to the user.

Keywords:

SQL, SQL Injection, databases, MySQL, Oracle, Microsoft SQL Server, Python

ΕΥΧΑΡΙΣΤΙΕΣ

Σε αυτό το σημείο θα ήθελα να ευχαριστήσω την αγαπημένη μου Τασούλα Ευαγγελία, την οικογένεια μου καθώς και την καθηγήτρια μου κυρία Καντζάβελου Ιωάννα, που όλοι με τον τρόπο τους συνέβαλαν στην εκπόνηση και τελικά στην ολοκλήρωση της διπλωματικής εργασίας μου.

ΠΕΡΙΕΧΟΜΕΝΑ

ΕΙΣΑΓΩΓΗ

Η παγκόσμια ανάπτυξη βασίζεται κατα το πλείστον σε γνώσεις και εξελίξεις της τεχνολογίας. Ζούμε αναμφισβήτητα στην εποχή της πληροφορίας όπου οι άνθρωποι έχουν την δυνατότητα να ανταλλάσουν και να μεταφέρουν πληροφορίες ελεύθερα και να έχουν άμεση πρόσβαση σε γνώσεις που στο παρελθόν θα ήταν δύσκολο να βρεθούν [1]. Υπάρχει μία γενική αντίληψη ότι είμαστε υπερφορτωμένοι με δεδομένα, καθιστώντας την δυνατότητα αποθήκευσης, ανάλυσης και επεξεργασίας αυτών των δεδομένων ως πρωταρχικό μέλημα της εποχής μας. Ειδικά στις πολυεθνικές εταιρείες όπου ο όγκος δεδομένων είναι τεράστιος. Έτσι λοιπόν γεννιέται το ερώτημα, πώς επεξεργαζόμαστε όλο αυτό τον όγκο δεδομένων, διατηρώντας παράλληλα την αξιοπιστία, το απόρρητο και την διαθεσιμότητα του. Σε πολλές περιπτώσεις, ο μεγάλος αυτός όγκος δεδομένων οργανώνεται σε συσχετισμένους πίνακες και μετά αποθηκεύεται σε μία βάση δεδομένων γνωστή ως σχεσιακή βάση δεδομένων, όπου παρέχει έναν μηχανισμό για ανάγνωση, εγγραφή και τροποποίηση των δεδομένων [2]. Σε αυτές τις βάσεις δεδομένων έχουν πρόσβαση οι ιστοσελίδες και διάφορες εφαρμογές (οι οποίες προφανώς έχουν μία βάση δεδομένων για να αποθηκεύουν τις πληροφορίες τους) μέσω μίας γλώσσας προγραμματισμού γνωστή ως SQL (Structured Query Language) [3]. Πολλές απο αυτές τις ιστοσελίδες είναι ευάλωτες σε χάκερ που επιδιώκουν να εκτελέσουν επιθέσεις λόγω κάποιων τρωτών σημείων που μπορεί να έχουν ανακαλύψει ή λόγω χαμηλών μέτρων ασφαλείας της βάσης δεδομένων. Συγκεκριμένοι ιστότοποι ενδέχεται να γίνονται στόχοι λόγω της αξίας των δεδομένων που ελπίζει να αποκτήσει πρόσβαση ο χάκερ. Το SQL Injection και το Cross Site Scripting (XSS) είναι 2 απο τους πιο συνηθισμένους τύπους επιθέσεων που χρησιμοποιούν οι χάκερ για να αποκτήσουν πρόσβαση σε πληροφορίες που δεν θα έπρεπε να έχουν [4]. Το μόνο σίγουρο είναι ότι δεν υπάρχει τέλειο σύστημα προστασίας καθώς η τεχνολογία εξελίσσεται καθημερινά και θα βρίσκονται ολοένα και περισσότεροι τρόποι παραβίασης των βάσεων δεδομένων ή πολύτιμων πληροφοριών που δεν πρέπει να αποκτήσει πρόσβαση κάποιος τρίτος.

ΣΤΟΧΟΣ ΚΑΙ ΟΡΓΑΝΩΣΗ ΔΙΠΛΩΜΑΤΙΚΗΣ

Τα website και οι εφαρμογές που πέφτουν θύματα τις επίθεσης SQLI αυξάνεται καθημερινά. Η ιστορία έχει δείξει πως το να βασιζόμαστε στους προγραμματιστές να φτιάξουν τον τέλειο κώδικα που να ελέγχει στο έπακρο την είσοδο του χρήστη, έχει αποτύχει, οπότε χρειάζεται μια διαφορετική προσέγγιση για την καλύτερη προστασία των βάσεων δεδομένων αλλά και των ιστοσελίδων ή των εφαρμογών που επικοινωνούν με μία βάση δεδομένων.

Στόχος της διπλωματικής αυτής είναι να δοθούν λεπτομερείς πληροφορίες της διαδικτυακής επίθεσης SQL Injection, έτσι ώστε να αποφευχθούν όσο το δυνατόν περισσότερες επιθέσεις και να ενημερώσει περαιτέρω τους ήδη ενασχολούντες με το αντικείμενο με απλές αλλά και προηγμένες τεχνικές ασφαλείας.

Στο 1ο κεφάλαιο θα αναφερθούν κάποιες εισαγωγικές έννοιες για να εξοικειωθεί ο αναγνώστης με το αντικείμενο και να μπορέσει να συνεχίσει να διαβάζει την υπόλοιπη διπλωματική.

Στο 2ο κεφάλαιο θα γίνει εμβάθυνση στην επίθεση SQL Injection.

Στο 3ο κεφάλαιο θα αναλυθούν τρόποι προστασίας απο SQL Injection εξαρχής αλλά και αφού έχει γίνει μία επίθεση σε κάποια βάση δεδομένων.

Στο 4ο κεφάλαιο θα γίνει ανίχνευση της επίθεσης σε ήδη εκτεθειμένη βάση δεδομένων.

Στο 5ο κεφάλαιο θα αναπτυχθεί εργαλείο σε γλώσσα προγραμματισμού Python, το οποίο θα ανιχνεύει αν μία βάση δεδομένων είναι ευάλωτη σε SQL Injection, error-based.

ΚΕΦΑΛΑΙΟ 1. ΕΙΣΑΓΩΓΙΚΕΣ ΕΝΝΟΙΕΣ

1.1 Τι είναι μία βάση δεδομένων SQL

Η SQL (Structured Query Language) (προφέρεται και 'sequel'='σίκουελ') είναι μία γλώσσα προγραμματισμού υψηλού επιπέδου και έχει σχεδιαστεί για να διαχειρίζεται δεδομένα που διατηρούνται σε ένα σχεσιακό σύστημα διαχείρισης βάσεων δεδομένων (RDBMS = Relational Database Management System) ή για να επεξεργάζεται την ροή σε ενα σχεσιακό σύστημα διαχείρισης ροής δεδομένων (RDSMS = Relational Data Stream Management System). Είναι ιδιαίτερα χρήσιμη και αξιόπιστη στο χειρισμό δομημένων δεδομένων, δηλαδή δεδομένων που ενσωματώνουν σχέσεις μεταξύ των οντοτήτων και των μεταβλητών. Εξαρτάται σε μεγάλο βαθμό απο τη σχεσιακή άλγεβρα και το σχεσιακό λογισμό [3, 5, 6].

Η SQL αποτελείται απο δηλωτικά στοιχεία όπως ερωτήματα, εκφράσεις , δηλώσεις κτλπ. Οι βασικές διαφορές μιας γλώσσας προγραμματισμού με μία γλώσσα ερωτημάτων (όπως η SQL) είναι ότι οι γλώσσες ερωτημάτων δεν αναμένονται να έχουν πληρότητα Turing, δηλαδή δεν μπορούν να χρησιμοποιηθούν για να προσομοιώσουν οποιαδήποτε μηχανή Turing, δεν έχουν δημιουργηθεί για να κάνουν πολύπλοκους υπολογισμούς αλλά έχουν εξαιρετική απόδοση στο να χειρίζονται μεγάλο όγκο δεδομένων [7, 8, 9].

Όπως αναφέρθηκε και παραπάνω για να λειτουργήσει η SQL βασίζεται στην σχεσιακή άλγεβρα (όπου είναι το λειτουργικό μέρος) και στον σχεσιακό λογισμό (όπου είναι το δηλωτικό μέρος). Η σχεσιακή άλγεβρα παραπέμπει σε μία συγκεκριμένη ακολουθία εντολών για την εκτέλεση ενός συγκεκριμένου αιτήματος, ενώ ο σχεσιακός λογισμός παρέχει μόνο την περιγραφή ενός ερωτήματος αλλα όχι την μέθοδο για να λυθεί το ερώτημα. Επίσης, επεξηγεί τι πρέπει να γίνει αλλά καμία πληροφορία σχετικά με την ακολουθία λειτουργίας που απαιτείται για την επεξεργασία ενός αιτήματος [10, 11, 12].

Όταν ένας ιστότοπος χρειάζεται να χειριστεί μία βάση δεδομένων, πρέπει να δημιουργήσει ένα ερώτημα SQL για αυτή την ενέργεια. Σε γενικές γραμμές, οι εφαρμογές διαδικτύου κατασκευάζουν ένα ερώτημα SQL το οποίο συνδυάζει τον κώδικα που έχει γραφτεί απο τον προγραμματιστή και τα δεδομένα που θα εισάγει ο χρήστης.

Π.χ. SELECT name, text from book WHERE id=$id (Το $id θα συμπληρωθεί απο τον χρήστη)

Το ερώτημα μεταφράζεται απο τον διερμηνέα SQL και τα αποτελέσματα επιστρέφουν στην ιστοσελίδα σε μορφή απλού κειμένου. Οι πιο συνηθισμένες εντολές στην SQL είναι οι εξής:

SELECT – (Καθορίζει ποιές στήλες των δεδομένων θα εμφανιστούν στα αποτελέσματα)

SELECT * FROM orders;

(Εμφανίζει όλα τα στοιχεία απο τον πίνακα ORDERS) (Το * υποδεικνύει να παρθούν όλα τα δεδομένα του πίνακα. Σε διαφορετική περίπτωση μπορούμε να επιλέξουμε να εμφανίσει συγκεκριμένη/ες στήλες του πίνακα.)

CREATE TABLE – (Δημιουργεί έναν νέο πίνακα στην βάση δεδομένων)

CRATE TABLE orders (ID int, Price int, Address varchar (255));

INSERT – (Προσθέτει μία νέα σειρά δεδομένων στον πίνακα)

INSERT INTO orders (ID, Price, Address)

VALUES (525, 50, Naxou 42);

DELETE – (Διαγράφει μερικά ή όλα τα δεδομένα ενός πίνακα, ανάλογα την συνθήκη WHERE)

DELETE FROM orders WHERE ID = 525;

INNER JOIN – (Συνδυάζει γραμμές διαφορετικών πινάκων αν η συνθήκη JOIN είναι αληθής)

SELECT column_name FROM table_1 JOIN table_2

ON table_1.column_name = table_2.column_name;

UNION – (Ενώνει 2 ή παραπάνω κομμάτια εντολών για να εκτελεστούν μαζί)

SELECT column_name FROM table1 UNION SELECT column_name FROM table2;

Επιπλέον εντολές που είναι χρήσιμες είναι το UPDATE, ALTER TABLE, GROUP BY, ORDER BY, CASE κτλπ [13, 14, 15].

Συνοψίζοντας η SQL είναι ένα σχεσιακό μοντέλο που ορίζεται αυστηρά με απλότητα και τη δύναμη της άλγεβρας για την αποτελεσματική εκτέλεση όλων των εργασιών που σχετίζονται με την βάση δεδομένων έχοντας την καλύτερη δυνατή βελτιστοποίηση.

1.1.1 Τα δημοφιλέστερα Συστήματα Διαχείρισης Βάσεων Δεδομένων (ΣΔΒΣ)

Τα 3 πιο γνωστά και τα πιο χρησιμοποιούμενα συστήματα διαχείρισης βάσεων δεδομένων είναι η Oracle, η MySQL και η Microsoft SQL Server κατακτώντας περίπου το 60% της αγοράς φεβρουάριος 2021) σύμφωνα με το TOPDB index [57].

Worldwide, Feb 2021 compared to a year ago:

Rank	Change	Database	Share	Trend
1		Oracle	30.2 %	+1.2 %
2		MySQL	16.65 %	-1.2 %
3		SQL Server	13.21 %	-0.9 %

Εικόνα 1.1.1.i [57]

Τα δεδομένα συλλέγονται με βάση το πόσο συχνά γίνεται αναζήτηση για την κάθε βάση στο Google. Τα στατιστικά που αναφέρονται στην στήλη trend συγκρίνονται με βάση 1 χρόνο πριν δηλαδή τον φεβρουάριο του 2020. Η Oracle εδώ και τουλάχιστον 20 χρόνια κατέχει τα ηνία της πρώτης θέσης και με αρκετά μεγάλη διαφορά απο την 2^{η} πιο δημοφιλή βάση.

ORACLE:

Η Oracle ιδρύθηκε το 1979 και είναι ένα ΣΔΒΔ που αναπτύχθηκε απο την Oracle. Η βάση αυτή υποστηρίζει πολλαπλά μοντέλα δεδομένων και συνήθως χρησιμοποιείται για την επεξεργασία διαδικτυακών συναλλαγών, για την αποθήκευση δεδομένων και για τον συνδυασμό αυτών των δύο. Τα διάφορα μοντέλα δεδομένων που αναφέρθηκαν προηγουμένως είναι τα γραφήματα, τα έγγραφα και οι τιμές-κλειδιά [58, 59].

MySQL:

Η MySQL ιδρύθηκε το 1995 και είναι ενα σύστημα διαχείρισης σχεσιακής βάσης δεδομένων (ΣΔΣΒΔ) ανοιχτού κώδικα που αναπτύχθηκε απο την MySQL AB αλλά πλέον η ανάπτυξη και η υποστήριξη παρέχεται απο την Oracle. Η βάση αυτή οργανώνει τα δεδομένα της σε έναν ή περισσότερους πίνακες όπου οι τύποι δεδομένων τους συσχετίζονται μεταξύ τους βοηθώντας στην καλύτερη δομή των δεδομένων. Χρησιμοποιείται κυρίως για την αποθήκευση δεδομένων των ιστοσελίδων [60, 61].

Microsoft SQL Server:

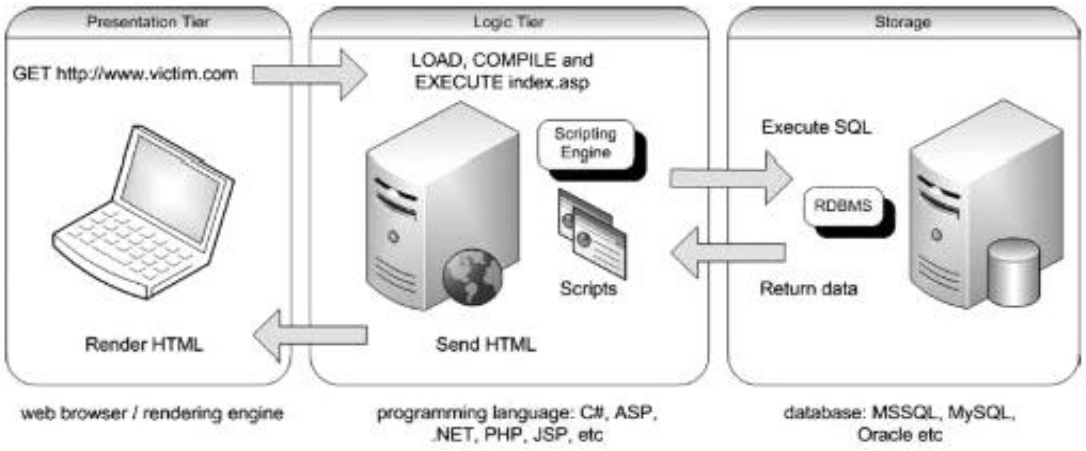
Η Microsoft SQL Server ιδρύθηκε το 1989 και είναι ενα ΣΔΣΒΔ που αναπτύχθηκε απο την Microsoft. Εκτός απο την αποθήκευση και τροποποίηση των δεδομένων παρέχει ανάλυση και προστασία καθώς αποτελείται απο ένα ολόκληρο συμπλεγμα εφαρμογών. Κυρίως χρησιμοποιείται στον εταιρικό τομέα [60,62].

1.2 Επικοινωνία ιστότοπου με βάση δεδομένων

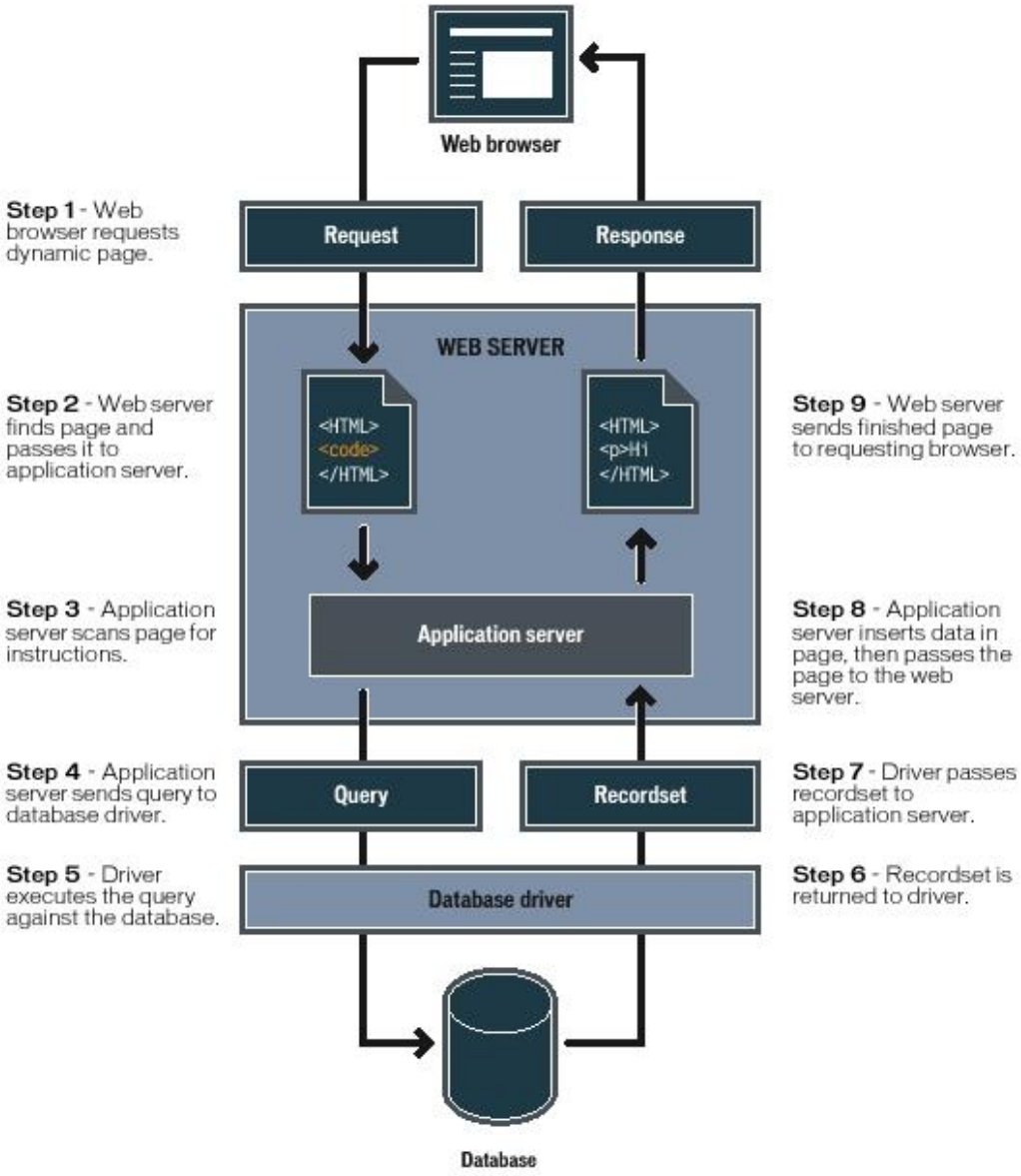
Για να κατανοήσουμε πλήρως τις βάσεις δεδομένων πρέπει να καταλάβουμε και πως γίνεται η επικοινωνία μεταξύ του ιστότοπου και της βάσης δεδομένων του.

Στην αρχή έχουμε τον Web Browser όπου εκεί ο απλός χρήστης πλοηγείται στο διαδίκτυο (έχοντας σύνδεση στο internet) και εισάγει την διεύθυνση του ιστού που θέλει να επισκεφθεί. Ο Server στέλνει ένα αντίγραφο μίας φόρμας στον χρήστη. Ο χρήστης συμπληρώνει την φόρμα αυτή (είτε για να κάνει σύνδεση στον λογαριασμό του, είτε για να κάνει αναζήτηση κάποιου προϊόντος / υπηρεσίας) και στέλνεται στον Web Server. Ο Application Server σαρώνει αυτή την φορμα, παίρνει τις πληροφορίες που χρειάζεται, στέλνει ερώτημα στον driver της βάσης δεδομένων για το αν ισχύουν τα δεδομένα και αυτός με την σειρά του στέλνει το ερώτημα στην βάση δεδομένων (ο driver της βάσης δεδομένων είναι ένα λογισμικό και λειτουργεί σαν διερμηνέας μεταξύ του application server και της βάσης). Η βάση με την σειρά της αφού βρεί δεδομένα συσχετιζόμενα με την αναζήτηση ή την προσπάθεια σύνδεσης του χρήστη, στέλνει με την ίδια σειρά προς τα πίσω αυτά τα δεδομένα, τα οποία τελικά καταλήγουν στον Web Browser σε απλό κείμενο για να διαβαστούν απο τον χρήστη [16, 17].

Στα παρακάτω σχήματα φαίνεται αρκετά αναλυτικά όλη η διαδικασία που πραγματοποιείται:



Εικόνα 1.2.i [65]



Εικόνα 1.2.ii [16]

1.3 Οι πιο συνηθισμένες επιθέσεις στον κυβερνοχώρο

Επίθεση στον κυβερνοχώρο θεωρείται κάθε είδος κακόβουλης ενέργειας που στοχεύει σε συστήματα πληροφορικής, δίκτυα υπολογιστών και γενικότερα ηλεκτρονικούς υπολογιστές. Οι επιθέσεις έχουν σκοπό την κλοπή, διαστρέβλωση ή ακόμα και την διαγραφή των δεδομένων. Οι εταιρείες πέφτουν θύματα τέτοιων επιθέσεων καθημερινά, καθώς οι επίδοξοι ληστές προσπαθούν να επωφεληθούν απο ευάλωτα επιχειρηματικά συστήματα. Σύμφωνα με τα λεγόμενα του John

Chambers [πρώην CEO της CISCO (πολυεθνική εταιρεία που σχεδιάζει και εμπορεύεται ηλ. προϊόντα)] υπάρχουν 2 είδη εταιρειών: Αυτές που έχουν πέσει θύμα hacking και αυτές που δεν γνωρίζουν ότι έχουν πέσει θύμα hacking [18, 19].

Παρακάτω θα γίνει μία αναφορά σε κάποιες απο τις πιο γνωστές και συνηθισμένες επιθέσεις στον κυβερνοχώρο, με μία μικρή επεξήγηση για την καθεμία.

i) Malware:

> Ότι έχει να κάνει με κακόβουλο λογισμικό (ιοί, spyware , worms κτλπ) εντάσσεται σε αυτή την κατηγορία. Τρόπος μόλυνσης: Μέσω κάποιου e-mail ή κάποιου συνδέσμου όπου αν ο χρήστης πατήσει και μεταφερθεί εκεί, εγκαθιστάται επικίνδυνο λογισμικό.

ii) Dos & DDoS:

> Μία DoS επίθεση έχει σκοπό να εξαντλήσει τους πόρους και το εύρος ζώνης ενός συστήματος ή ενός server ή ενός δικτύου φορτώνοντας το με αμέτρητα αιτήματα, με αποτέλεσμα να μην μπορεί να τα εξυπηρετήσει όλα και να γίνεται μη διαθέσιμο (offline) στους χρήστες. Στην περίπτωση του DDoS γίνεται ακριβώς το ίδιο με μόνη διαφορά ότι τα αιτήματα στέλνονται απο χακαρισμένες συσκευές που ελέγχει ο επιτιθέμενος.

iii) SQL Injection:

> Μία SQL Injection επίθεση επιτυγχάνεται όταν ο επιτιθέμενος εισάγει κακόβουλο κώδικα στον διακομιστή που χρησιμοποιεί βάση δεδομένων αναγκάζοντας τον διακομιστή να αποκαλύψει πληροφορίες που δεν θα έπρεπε. Αυτή η επίθεση μπορεί να πραγματοποιηθεί ακόμη και απο το πλαίσιο αναζήτησης ευάλωτης ιστοσελίδας. Περισσότερα για αυτή την επίθεση θα αναπτυχθούν στο επόμενο κεφάλαιο.

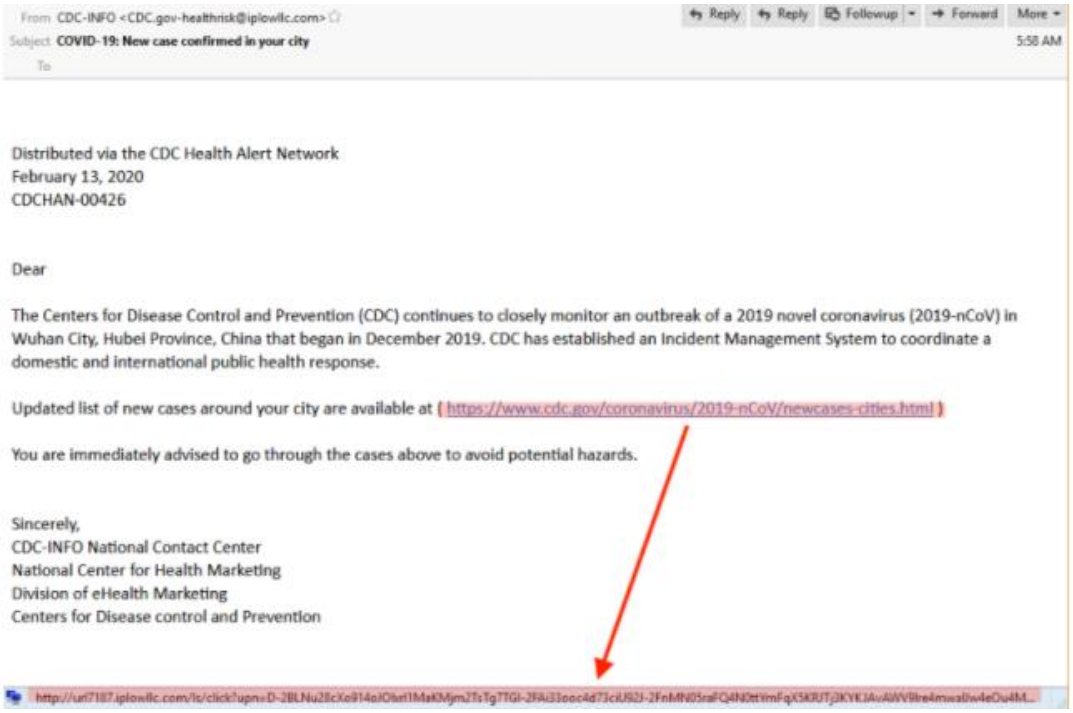
iv) Man-in-the-middle:

> Αυτό το είδος της επίθεσης είναι όταν ο επιτιθέμενος εισβάλλει στην επικοινωνία 2 ατόμων χώρις να γίνεται αντιληπτός και συνομιλεί κανονικά με το άλλο άτομο. Είναι γνωστή σαν επίθεση υποκλοπής, αφού μολις εισβάλλει στην συνομιλία μπορεί να χειριστεί και να κλέψει ευαίσθητες πληροφορίες.

v) Phishing:

> Στο Phishing κατατάσσεται η τακτική του να στέλνουν δόλια μηνύματα ή email που φαίνεται να προέρχονται απο αξιόπιστη πηγή, με σκοπό ο χρήστης να ανοίξει το μήνυμα / email, να εγκατασταθεί κακόβουλο λογισμικό και στην συνέχεια να κλαπούν στοιχεία σύνδεσης, πιστωτικές κάρτες κτλπ [18, 19, 20, 21].

Σύμφωνα με την Purplesec υπάρχει αύξηση του κυβερνοεγκλήματος κατα 600% λόγω της πανδημίας COVID-19. Έχει παρατηρηθεί μία τεράστια αύξηση στο Phishing όπου κακόβουλοι χρήστες στέλνουν μαζικά email σε ανυποψίαστους χρήστες παριστάνοντας ότι είναι από τον Παγκόσμιο Οργανισμό Υγείας προτρέποντας τους να πατήσουν σε ένα σύνδεσμο για να ενημερωθούν για τα κρούσματα της περιοχής τους ενώ στην πραγματικότητα εγκιθίσταται ένας ψηφιακός ιός στον υπολογιστή τους [54].



Εικόνα 1.3.i [54]

Υπάρχουν ακόμα πολλές τακτικές για να επιτύχει μία διαδικτυακή επίθεση και να κλαπούν ευαίσθητα δεδομένα. Υπάρχουν όμως και πολλοί τρόποι να αποφύγουμε αυτές τις επιθέσεις όπως να αναβαθμίζουμε τακτικά το antivirus, να αλλάζουμε τακτικά τους κωδικούς μας, να κάνουμε backup τα αρχεία μας κτλπ.

1.4 Στατιστικά εταιρειών διαδικτυακής προστασίας ευπαθειών

1.4.1 Imperva statistics

Παρακάτω θα αναφερθούν κάποια στατιστικά στοιχεία απο την Imperva για τις ευπάθειες που έχουν βρεθεί και πιο συγκεκριμένα για την επίθεση SQLI.

Ευπάθεια χαρακτηρίζεται κάθε είδος αδυναμίας στην εφαρμογή/ιστοσελίδα (κάποιο ελάττωμα της εφαρμογής ή κάποιο bug) την οποία μπορεί να εκμεταλλευτεί κάποιος εισβολέας και να προκαλέσει ζημιά στην εφαρμογή / ιστοσελίδα / βάση δεδομένων ή όποιο άλλο μέρος περιλαμβάνεται στο σύστημα [53].

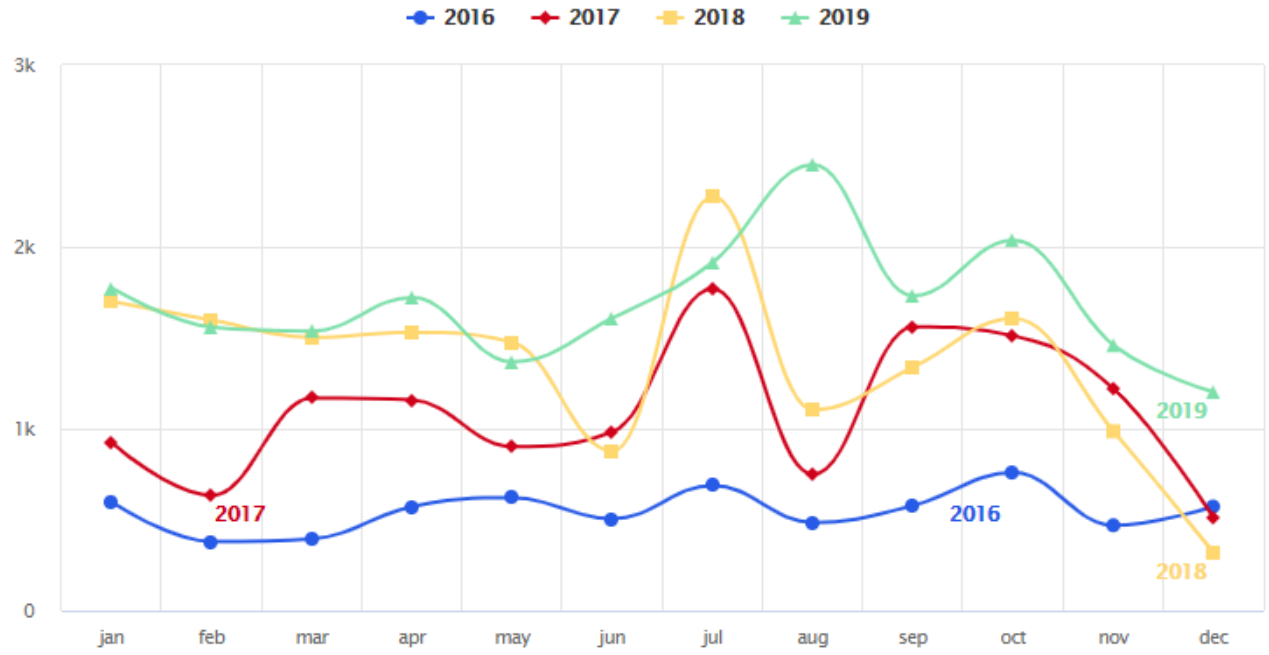


Figure 1: Number of vulnerabilities

Εικόνα 3.3.1.i [52]

Στο διάγραμμα παραπάνω έχουν καταγραφεί οι ευπάθειες που έχουυν εντοπιστεί ανα μήνα, τα τελευταία 4 χρόνια. Το σύνολο των νέων ευπαθειών για το 2019 φτάνει στις 20.362, αυξημένες κατά 17.6% απο το 2018 και 44.5% απο το 2017. Το 8% χαρακτηρίστηκε ως χαμηλής επικινδυνότητας, το 61% μεσσαίας, το 18% υψηλής και το 13% κρίσιμης [52].

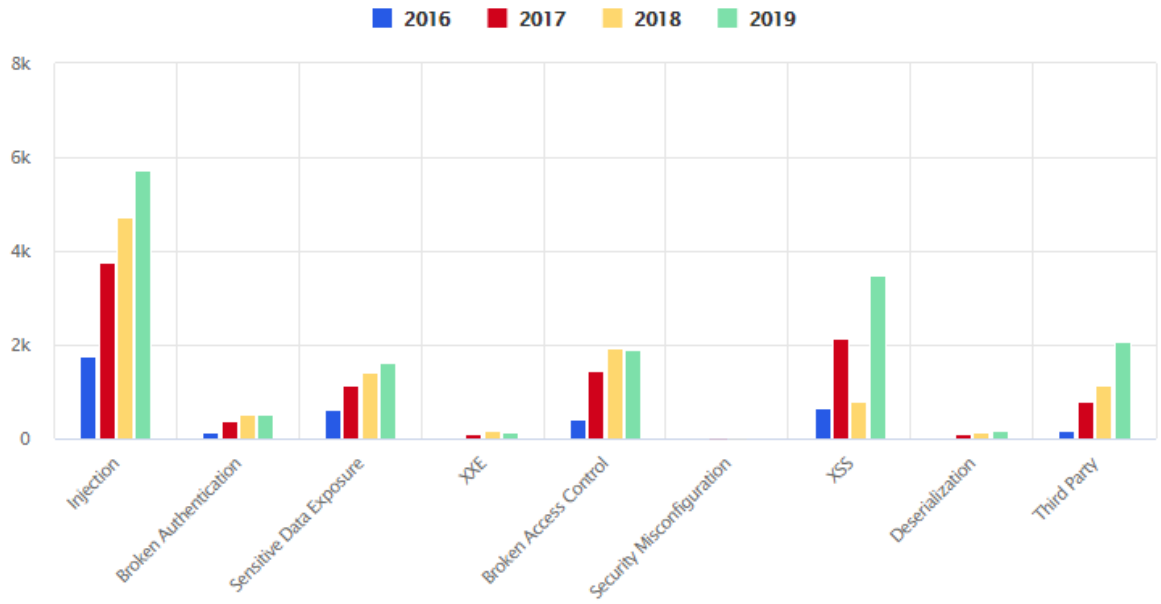


Figure 2: Vulnerabilities into OWASP categories

Εικόνα 3.3.1.ii [52]

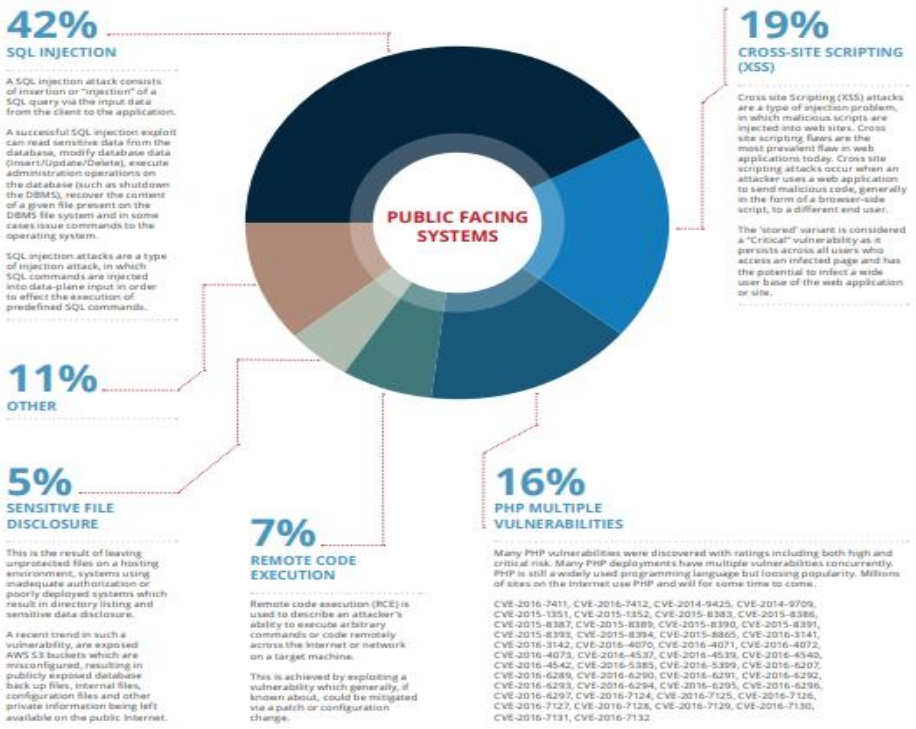
Οι 5.730 απο τις ευπάθειες αναφέρονται σε επιθέσεις τύπου Injection, ενώ μόνο οι 1.610 (8%) στην SQLI [52]. Όπως φαίνεται και απο το διάγραμμα κάθε χρόνο έχουν μία ανοδική τάση αυτού του είδους οι επιθέσεις.

1.4.2 Edgescan statistics

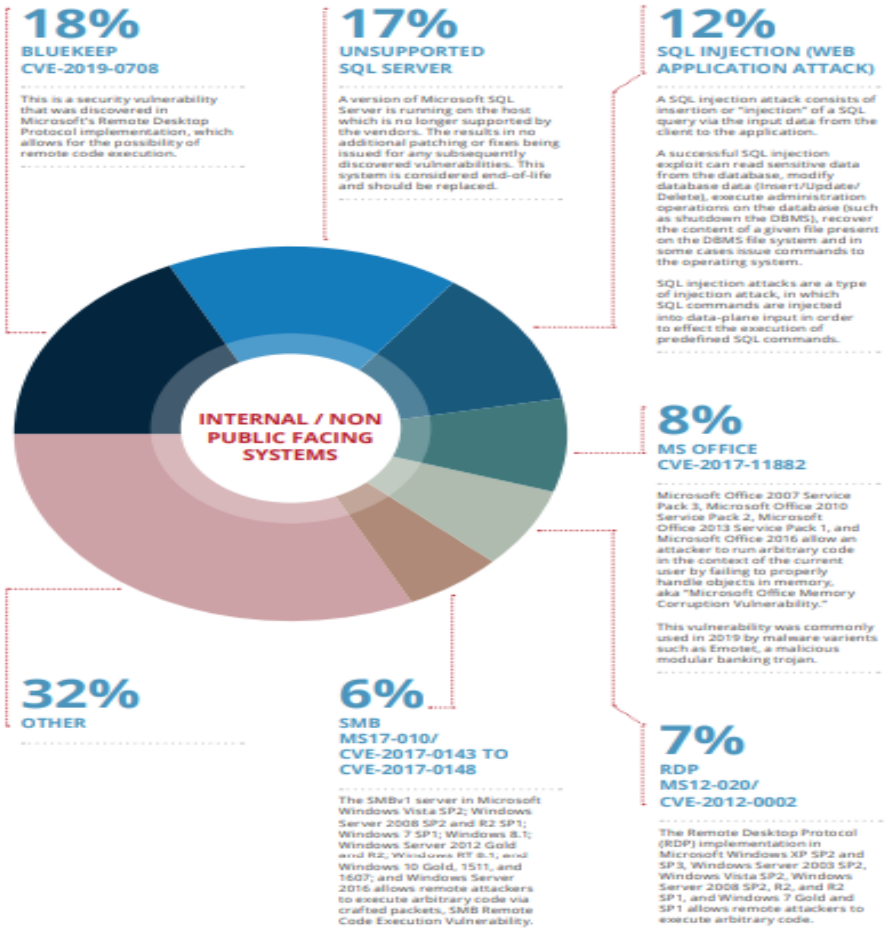
Η Edgescan εστιάζει στις επιθέσεις που έγιναν και είχαν πολύ κρίσιμο βαθμό επικινδυνότητας και αναφέρει χαρακτηριστικά πως στα συστήματα που υπήρχε δημόσια πρόσβαση, το 42% ήταν απο SQLI, ενώ σε αυτά που δεν υπήρχε δημόσια πρόσβαση μόλις το 12% [32].

Κρίσιμης επικινδυνότητας θεωρείται μία επίθεση όταν υπάρχει πιθανότητα να κυριαρχηθεί όλο το σύστημα απο τον εισβολέα και να κλαπούν/ διαγραφτούν όλα τα δεδομένα που υπάρχουν στην βάση δεδομένων [32].

Παρακάτω ακολουθούν τα 2 διάγραμματα με λεπτομερείς πληροφορίες για όλες τις επιθέσεις που είχαν κρίσιμη επικινδυνότητα:



Εικόνα 1.4.i [32]



Εικόνα 1.4.ii [32]

Αναλύοντας τα γραφήματα παρατηρούμε ότι στα συστήματα που είναι διαθέσιμα δημόσια οι επιθέσεις τύπου SQLI έχουν μεγαλύτερη αποτελεσματικότητα σε σχέση με τα συστήματα που δεν είναι δημόσια [32].

Οπότε λόγω και της αυξημένης επικινδυνότητας και κρισιμότητας της επίθεσης αυτής θα επιμείνουμε και θα σταθούμε για να αναλυθεί όσο περισσότερο γίνεται σε βάθος.

ΚΕΦΑΛΑΙΟ 2. SQL Injection

2.1 Τι είναι το SQL Injection

Το SQL Injection (απο εδώ και στο εξής SQLI για συντομία) όπως αναφέρθηκε και σε προηγούμενη ενότητα είναι μία απο τις πιο συνηθισμένες και δημοφιλής διαδικτυακή επίθεση. Είναι ο τύπος επίθεσης που καθιστά δυνατή την εκτέλεση κακόβουλων δηλώσεων σε μία βαση δεδομένων SQL. Ο κακόβουλος κώδικας εισάγεται στο πεδίο που παρέχεται στον χρήστη και μετά καταλήγει στον διακομιστή της βάσης δεδομένων για ανάλυση και εκτέλεση. Αυτές οι δηλώσεις έχουν σκοπό να ελέγξουν τον διακομιστή της βάσης δεδομένων. Οι εισβολείς χρησιμοποιούν τις ευπάθειες του SQLI για να παρακάμψουν τα μέτρα ασφαλείας. Μπορούν να ελέγξουν την διαδικασία ταυτοποίησης του χρήστη και την εξουσιοδότηση μιας ιστοσελίδας και να ανακτήσουν όλο το περιεχόμενο της συνδεδεμένης βάσης δεδομένων SQL. Επίσης έχουν την δυνατότητα να προσθέσουν, να τροποποιήσουν ακόμη και να διαγράψουν δεδομένα απο την βάση [22].

Υπάρχουν 4 τρόποι επίθεσης SQLI και αυτοί είναι:

1) Χειραγώγηση της SQL (SQL Manipulation):

Ο τρόπος αυτός περιλαμβάνει την τροποποίηση των δηλώσεων SQL που στέλνονται στην βάση δεδομένων μέσω των κατηγοριών που θα αναλυθούν παρακάτω (Union-based) και την τροποποίηση της συνθήκης WHERE για να εμφανιστεί κάποιο άλλο αποτέλεσμα [64].

2) Έγχυση κώδικα (Code Injection):

Η έγχυση κώδικα έχει σκοπό την προσθήκη περισσότερων SQL ερωτημάτων στο ήδη υπάρχων ερώτημα SQL με σκοπό να εμφανιστούν παραπάνω πληροφορίες απο τα δεδομένα της βάσης ή να παρθούν περισσότερες πληροφορίες για την ίδια την βάση (ποιό είδος βάσης δεδομένων χρησιμοποιείται, ποιά έκδοση κτλπ) [64].

3) Έγχυση κλήσης συνάρτησης (Function Call Injection):

Αυτός ο τρόπος είναι για βάσεις που χρησιμοποιούν Oracle και αυτό που κάνει είναι, μέσω της έγχυσης να καλεί έτοιμες συναρτήσεις της Oracle ή κάποια φτιαχτή του εισβολέα σε ένα ευπαθές ερώτημα SQL με σκοπό να προκαλέσει κάποια χειραγώγηση των δεδομένων της βάσης ή να εκτελεστούν κάποιες εντολές στο λειτουργικό σύστημα [64].

4) Υπερχείλιση του Buffer (Buffer Overflow):

Και αυτός ο τρόπος έχει εφαρμογή στις βάσεις Oracle και χρησιμοποιείται για να υπερχειλίσει τον Buffer ο οποίος έχει τμήματα μνήμης που διατίθενται για την αποθήκευση δεδομένων. Η

υπερχείλιση πραγματοποιείται κάνοντας έγχυση κλήσης συνάρτησης (3ος τρόπος) κάποιων συγκεκριμένων συναρτήσεων που υπάρχουν ήδη στην βάση. Κάποιες απο αυτές που προκαλούν υπερχείλιση είναι: TZ_OFFSET, TO_TIMESTAMP_TZ, BFILENAME, NUMTOYMINTERVAL και NUMTODSINTERVAL. Εκμεταλεύοντας την υπερχείλιση του buffer με το SQLI μπορεί να αποκτηθεί απομακρυσμένη πρόσβαση στην βάση [64].

2.2 Ποιά συστήματα απειλούνται απο μία τέτοια επίθεση & τι δεδομένα υποκλέπτονται

Μία ευπάθεια του SQLI μπορεί να προσβάλλει οποιαδήποτε ιστοσελίδα η οποία έχει συνδεδεμένη απο πίσω της μία βάση δεδομένων όπως η MySQL, Oracle, Microsoft SQL Server κτλπ. Αυτή η ευπάθεια χρησιμοποιείται για να αποκτηθεί πρόσβαση σε ευαίσθητα δεδομένα όπως πληροφορίες πιστωτικών καρτών, στοιχεία πελατών, κωδικοί πρόσβασης και γενικά οποιοδήποτε δεδομένο θεωρείται προσωπικό και δεν έχει σκοπό να δημοσιοποιηθεί σε τρίτους. Συνήθως τα δεδομένα αυτά υποκλέπτοπται για να βγάλει κάποιο κέρδος ο επιτιθέμενος. Παλαιότερα υπήρχαν περιπτώσεις που γινόταν για να αποκτήσουν φήμη και να δείξουν τις ικανότητες τους, σε άλλους του κλάδου [22].

2.3 Κατηγοριοποίηση επιθέσεων SQLI

Υπάρχουν 3 είδη επιθέσεων SQLI και αυτές κατηγοριοποιούνται ανάλογα με τον τρόπο της επίθεσης που εκτελεί ο χάκερ. Θα αναφερθούμε σε αυτα τα 3 είδη και σε κάποιες απο τις υποκατηγορίες τους. Χωρίζονται σε:

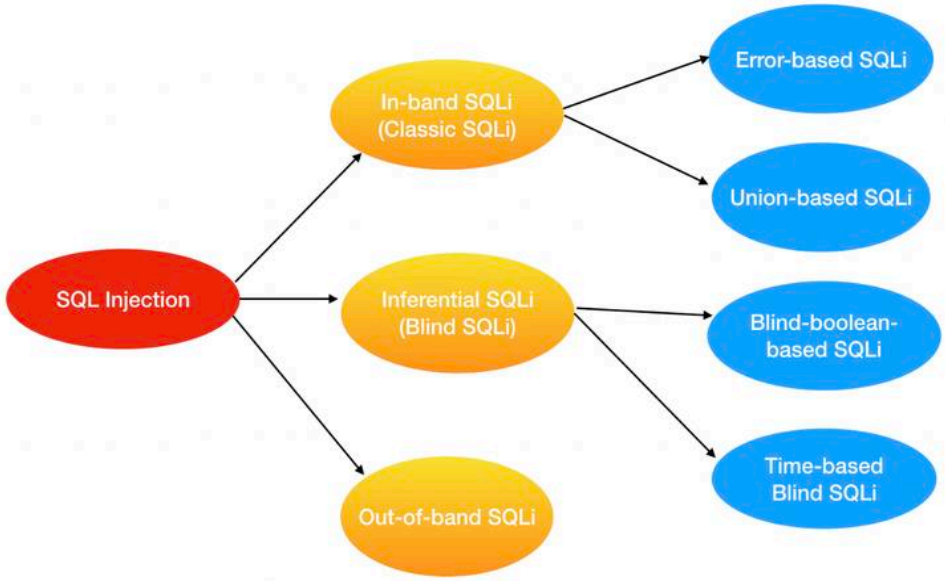
1) In-band SQLI

- Error-based SQLI
- Union-Based SQLI

2) Inferential SQLI (Blind SQLI)

- Boolean-based blind SQLI
- Time-based blind SQLI

3) Out-of-band SQLI



Εικόνα 2.3.i [23]

Ας αναλύσουμε κάθε ένα απο τα είδη και τις κατηγορίες τους.

1) In-band SQLI (Classic SQLI):

Είναι η πιο συνήθης και πιο εύκολη στην πραγματοποίηση της επίθεση SQLI. Το είδος αυτό είναι όταν ο επιτιθέμενος χρησιμοποιεί το ίδιο κανάλι επικοινωνίας τόσο για να κάνει την επίθεση του προς την βάση όσο και να εμφανίσει τα αποτελέσματα που αναζητά. Στέλνει τις εντολές του μέσω αίτησης GET ή POST HTTP και τα δεδομένα που απέσπασε εμφανίζονται στην ιστοσελίδα σε απλό κείμενο [24, 25].

Υποκατηγορίες:

i) Error-based SQLI:

Εδώ ο επιτιθέμενος βασίζεται στα μηνύματα λάθους (error messages) που θα ‘επιστρέψουν’ απο τον διακομιστή της βάσης με σκοπό να λάβει περαιτέρω πληροφορίες για την βάση. Για κάποιους πολυ εξοικιωμένους με την επίθεση αυτή, αρκούν μόνο τα μηνύματα λάθους για να κατανοήσουν πλήρως την δομή της βάσης και στη συνέχεια να την ελέγξουν ολοκληρωτικά. Αν δεν είναι τόσο έμπειρος, στέλνει κιάλλα παρόμοια ερωτήματα SQL για την καλύτερη κατανόηση της δομής της μέχρι να συλλέξει αρκετές πληροφορίες ώσπου να την παραβιάσει. Για αυτού του είδους επίθεσης έχει μεγάλο κομμάτι ευθύνης ο προγραμματιστής καθώς τα μηνύματα λάθους δεν θα έπρεπε να εμφανίζονται σε οποιοδήποτε χρήστη της ιστοσελίδας αλλά να καταγράφονται σε κάποιο άλλο αρχείο με περιορισμένη πρόσβαση [24, 25].

ii) Union-based SQLI

Σε αυτή την τεχνική ο επιτιθέμενος χρησιμοποιεί την εντολή UNION (Μία απο τις πολλές εντολές που χρησιμοποιούνται στις βάσεις δεδομένων) προκειμένου να συνδυάσει τα αποτελέσματα περισσότερων δηλώσεων SELECT σε μία μόνο απάντηση η οποία θα εμφανιστεί στον browser του επιτιθέμενου και έτσι συλλέγοντας τις πληροφορίες που χρειάζεται θα μπορέσει να κάνει το σωστό ερώτημα SQL και να εισχωρήσει στην βάση με ελεύθερη πρόσβαση [24, 25, 27].

2) Inferential SQLI (Blind SQLI):

Αυτό το είδος επίθεσης λέγεται εμπιστευτική ή τυφλή επίθεση SQLI και τον λόγο θα τον δούμε παρακάτω. Ο επιτιθέμενος εδώ στέλνει συγκεκριμένα ερωτήματα SQL προς την βάση και βγάζει συμπεράσματα απο την απόκριση και την συμπεριφορά του server, καθώς εδώ τα αποτελέσματα δεν εμφανίζονται στον browser του επιτιθέμενου και ενεργεί τυφλά (εξού και το όνομα της επίθεσης), έχοντας μόνο γνώμονα πως συμπεριφέρεται ο server . Η επίθεση αυτή είναι πιο χρονοβόρα σε σχέση με την in-band SQLI αλλά έχει εξίσου καταστροφικά αποτελέσματα [24, 27].

Υποκατηγορίες:

i) Boolean-based blind SQLI:

Στην boolean-based εκδοχή ο επιτιθέμενος στέλνει ενα SQL ερώτημα με σκοπό να επιστρέψει στην οθόνη του κάποιο αποτέλεσμα. Με βάση το αν το ερώτημα που θα στείλει ο εισβολέας είναι αληθές ή ψευδές, προσαρμόζεται και το αποτέλεμα που θα εμφανιστεί στον browser του, δηλαδή για παράδειγμα, σε μια περίπτωση αν είναι αληθές θα εμφανιστεί κανονικά η ιστοσελίδα ειδάλλως θα εμφανιστεί μία κενή σελίδα [24, 25].

ii) Time-based blind SQLI:

Στο time-based ο επιτιθέμενος στέλνει ένα ερώτημα SQL, το οποίο αναγκάζει την βάση δεδομένων να περιμένει για μερικά δευτερόλεπτα προτού σταλθεί η απάντηση πίσω. Με βάση το πόσο καθυστέρησε να σταλθεί η απάντηση καταλαβαίνει ο εισβολέας αν το ερώτημα που έστειλε ήταν αληθές η ψευδές [24, 27].

3) Out-of-band SQLI

Η τελευταία και η λιγότερο διαδεδομένη και χρησιμοποιημένη είναι η επίθεση out-of-band. Συνήθως χρησιμοποιείται σαν εναλλακτική των άλλων 2. Όταν ο επιτιθέμενος δεν μπορεί να χρησιμοποιήσει το ίδιο κανάλι επικοινωνίας όπως γίνεται στην περίπτωση της in-band επίθεσης, τότε καταφεύγει σε αυτή την λύση όπου χρησιμοποιείται διαφορετικό κανάλι επικοινωνίας. Αρχικά, για να λειτουργήσει μία τέτοια επίθεση θα πρέπει ορισμένες δυνατότητες του διακομιστή

βάσης να είναι ενεργοποιημένες, οι οποίες είναι απενεργοποιημένες απο προεπιλογή, οπότε αυτό το καθιστά μερικές φορές αδύνατο να επιτύχει μία επίθεση. Όμως σε περίπτωση που ο server είναι υπερφορτωμένος ή έχει μεγάλο χρόνο απόκρισης είναι προτιμότερη αυτή η τεχνική παρά της time-based, καθώς εδώ ο εισβολέας μπορεί να αποσπάσει μεγάλα κομμάτια δεδομένων σε αντίθεση με το time-based που αποσπάται χαρακτήρας - χαρακτήρας. Αυτή η επίθεση βασίζεται στην ικανότητα του server της βάσης να φτιάχνει αιτήματα DNS ή HTTP και να τα στέλνει πίσω στον επιτιθέμενο [24, 25, 27].

2.3.1 Άλλοι τρόποι έγχυσης δεδομένων

- Μέσω των Cookies:

Τα cookies στέλνονται στον περιηγητή του χρήστη και με κάθε νέα αίτηση HTTP στέλνονται πίσω στον διακομιστή της ιστοσελίδας αυτόματα. Συνήθως χρησιμοποιούνται για να κρατάνε συγκεκριμένες πληροφορίες του χρήστη όπως αυθεντικοποίηση της ταυτότητας, προτιμήσεις όπως για ποια προϊόντα ή υπηρεσίες μπορεί να τον ενδιαφέρουν ή ακόμα και η αποθήκευση του ονόματος χρήστη και του κωδικού για ευκολότερη μελλοντική σύνδεση. Θεωρούνται επικίνδυνα για τέτοιου είδους επιθέσεις καθώς είναι ένας άλλος τρόπος εισαγωγής δεδομένων ή εντολών απο τον χρήστη καθώς έχει πλήρη έλεγχο τι θα σταλθεί πίσω στον διακομιστή [65].

- Μέσω των κεφαλίδων Host, Referer και User-Agent του περιηγητή:

Όπως τα cookies έτσι και αυτές οι κεφαλίδες στέλνονται μέσω αιτημάτων HTTP. Πιο συγκεκριμένα, το πεδίο Host προσδιορίζει τον παροχέα του διαδικτύου και ποιον αριθμό θύρας του πόρου αιτήθηκε, το πεδίο Referer προσδιορίζει τον πόρο απο τον οποίο αποκτήθηκε το τρέχον αίτημα και το User-Agent προσδιορίζει τον περιηγητή ιστού που χρησιμοποιεί ο χρήστης. Η επίθεση μέσω αυτών των κεφαλίδων είναι πιο σπάνια καθώς μπορεί να γίνει μόνο αν η ιστοσελίδα χρησιμοποιεί αυτά τα δεδομένα για να βγάζει διάφορα στατιστικά και χρειάζεται να τα αποθηκεύει σε κάποια βάση δεδομένων [65].

2.4 Συνέπειες μιας επιτυχούς επίθεσης SQLI

Οι συνέπειες μιας τέτοιας επίθεσης μπορούν να αποβούν μοιραίες και πολυ επικίνδυνες για τους κατόχους των βάσεων δεδομένων αλλά και των πελατών τους (αν μιλάμε για εταιρεία) και για οποιονδήποτε άλλο χρήστη έχει μέσα στην βάση δεδομένων προσωπικές πληροφορίες. Πολλές απο τις παραβιάσεις που έχουν γίνει διαδικτυακά χρησιμοποιήθηκε αυτή η μέθοδος, που οδήγησαν σε τεράστιες ζημιές, πρόστιμα και κακή φήμη. Κάποιες απο τις περιπτώσεις ο εισβολέας είχε βρεί

τρόπο να εισέλθει στο σύστημα και για μεγάλο χρονικό διάστημα δεν είχε γίνει αντιληπτός ενώ εκμεταλευόταν πόρους σε όλη την διάρκεια.

- Παράκαμψη ελέγχου ταυτότητας :

Ο επιτιθέμενος μπορεί να συνδεθεί στην ιστοσελίδα χωρίς να παρέχει τα κατάλληλα στοιχεία σύνδεσης.

- Απόκτηση πρόσβασης και ελέγχου σε δεδομένα που δεν έχει εξουσιοδότηση :

Όχι μόνο μπορεί να αποκτήσει πρόσβαση στα δεδομένα της βάσης αλλά και να τα τροποποιήσει ή ακόμη και να τα διαγράψει.

- Απόκτηση δικαιώματα διαχειριστή :

Έχοντας τον πλήρη έλεγχο πλέον, μπορεί ακόμη και να κλείσει τελείως τον διακομιστή της βάσης δεδομένων [28, 29].

2.5 Παραδείγματα SQLI

Για τον σκοπό της διπλωματικής και την πρακτική κατανόηση της επίθεσης SQLI, έχουμε δημιουργήσει μία βάση δεδομένων Users, η οποία αποτελείται απο 2 πίνακες και πιο συγκεκριμένα ο πίνακας που μας ενδιαφέρει είναι ο credential που περιέχει κάποιες στήλες όπως το ID, το όνομα , τον μισθό κτλπ ενός υπαλληλου.

```
mysql> use Users;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> show tables;
+-----------------+
| Tables_in_Users |
+-----------------+
| credential      |
+-----------------+
1 row in set (0.02 sec)

mysql> describe credential;
+-------------+-----------------+------+-----+---------+----------------+
| Field       | Type            | Null | Key | Default | Extra          |
+-------------+-----------------+------+-----+---------+----------------+
| ID          | int(6) unsigned | NO   | PRI | NULL    | auto_increment |
| Name        | varchar(30)     | NO   |     | NULL    |                |
| EID         | varchar(20)     | YES  |     | NULL    |                |
| Salary      | int(9)          | YES  |     | NULL    |                |
| birth       | varchar(20)     | YES  |     | NULL    |                |
| SSN         | varchar(20)     | YES  |     | NULL    |                |
| PhoneNumber | varchar(20)     | YES  |     | NULL    |                |
| Address     | varchar(300)    | YES  |     | NULL    |                |
| Email       | varchar(300)    | YES  |     | NULL    |                |
| NickName    | varchar(300)    | YES  |     | NULL    |                |
| Password    | varchar(300)    | YES  |     | NULL    |                |
+-------------+-----------------+------+-----+---------+----------------+
11 rows in set (0.07 sec)
```

Εικόνα 2.5.i

Γεμίζουμε τον πίνακα με κάποιες πληροφορίες υπαλλήλων.

```
mysql> select * from credential;
+----+-------+-------+--------+-------+----------+-------------+---------+-------+----------+------------------------------------------+
| ID | Name  | EID   | Salary | birth | SSN      | PhoneNumber | Address | Email | NickName | Password                                 |
+----+-------+-------+--------+-------+----------+-------------+---------+-------+----------+------------------------------------------+
|  1 | Alice | 10000 |  20000 | 9/20  | 10211002 |             |         |       |          | fdbe918bdae83000aa54747fc95fe0470fff4976 |
|  2 | Boby  | 20000 |  30000 | 4/20  | 10213352 |             |         |       |          | b78ed97677c161c1c82c142906674ad15242b2d4 |
|  3 | Ryan  | 30000 |  50000 | 4/10  | 98993524 |             |         |       |          | a3c50276cb120637cca669eb38fb9928b017e9ef |
|  4 | Samy  | 40000 |  90000 | 1/11  | 32193525 |             |         |       |          | 995b8b8c183f349b3cab0ae7fccd39133508d2af |
|  5 | Ted   | 50000 | 110000 | 11/3  | 32111111 |             |         |       |          | 99343bff28a7bb51cb6f22cb20a618701a2c2f58 |
|  6 | Admin | 99999 | 400000 | 3/5   | 43254314 |             |         |       |          | a5bdf35a1df4ea895905f6f6618e83951a6effc0 |
+----+-------+-------+--------+-------+----------+-------------+---------+-------+----------+------------------------------------------+
6 rows in set (0.00 sec)
```

Εικόνα 2.5.ii

Προτού ξεκινήσουμε την επίθεση, πρέπει να ξεκαθαρίσουμε ότι:

Πρώτον η ιστοσελίδα και η βάση της είναι τοπικά προσβάσιμη δηλαδή μόνο απο τον υπολογιστή μας. Δεύτερον τα στοιχεία που περιέχονται είναι μη πραγματικά. Τρίτον έχουμε κάνει ρύθμιση έτσι ώστε η βάση μας να είναι ευάλωτη σε επιθέσεις SQLI και τέταρτον για λόγους εκπαιδευτικούς η επίθεση είναι πιο απλοϊκή απ' όσο είναι στην πραγματικότητα.

Ο ιστότοπος που θα προσπαθήσουμε να κάνουμε την επίθεση είναι μία απλή εφαρμογή διαχείρισης υπαλλήλων. Ας υποθέσουμε ότι στα πεδία Username και Password για την επαλήθευση των χρηστών κρύβεται ο ακόλουθος κώδικας απο πίσω.

```
"SELECT id, name, eid, salary, birth, ssn, address, email,
        nickname, Password
 FROM credential
 WHERE name= '$input_uname' and Password='$hashed_pwd'";
```

Εικόνα 2.5.iii (όπου Username = name)

Για το input_name αντιστοιχεί η στήλη της βάσης με όνομα name και για το hashed_pwd αντιστοιχεί η στήλη password, αλλά την hashed τιμή με βάση τον αλγόριθμο sha1. Στην περίπτωση του Admin έχει ρυθμιστεί να εμφανίζει όλες τις πληροφορίες των υπαλλήλων.

Εαν θελήσει κάποιος επιτιθέμενος να εισβάλλει σε αυτή τη βάση και να αποκτήσει πληροφορίες γνωρίζοντας μόνο το username και οχι το password, μπορεί να πληκτρολογήσει στο πεδίο username το εξής:

Username:Admin'—[και το πεδίο password να το αφήσει κενό (Classic SQLI)] Ή να χρησιμοποιήσει ταυτολογία που είναι πάντα αληθής Username:' OR 1=1— (Boolean-based). Αν βγάλει κάποιο σφάλμα του τύπου (You have an error in your SQL syntax...) σημαίνει ότι έχει γίνει κάποιο συντακτικό λάθος και με μερικές ακόμα εντολές θα είναι εύκολο να παρακαμφθεί.

Με αυτό τον τρόπο αυθεντικοποιεί το username, κλείνει το πεδίο εισαγωγής τιμής με την απόστροφο ' και κάνει σχόλιο τον υπόλοιπο κώδικα που ακολουθεί με το σύμβολο -- και έτσι συνδέθηκε χωρίς να χρειάζεται το password.

Αφού έχει εισέλθει στην βάση και θέλει να τροποποιήσει όποιο απο τα δεδομένα υπάρχουν θα πρέπει να βρεί κάποιο πεδίο το οποίο ούτως ή αλλώς κάνει αυτή την ενέργεια όπως π.χ. να πάει

στο edit profile και να συμπληρώσει εκεί τις κακόβουλες εντολές. Στην ενότητα edit profile τρέχει ο παρακάτω κώδικας:

```
$sql = "UPDATE credential SET
   nickname='$input_nickname',
   email='$input_email',
   address='$input_address',
   Password='$hashed_pwd',
   PhoneNumber='$input_phonenumber'
   WHERE ID=$id;";
```

Εικόνα 2.5.iv

Όπως βλέπουμε το ερώτημα SQL ξεκινάει με το UPDATE ενώ στην πλατφόρμα σύνδεσης χρήστη το ερώτημα SQL ήταν SELECT. Εδώ μπορεί να τροποποιήσει τα δεδομένα του χρήστη που έχει εισέλθει (απλά αλλάζοντας τις τιμές μέσω των πεδίων που παρέχει η ιστοσελίδα), να τροποποιήσει δεδομένα που δεν του δίνεται η δυνατοτητα να αλλάξει επειδή είναι κρυφά και δεν τα καθορίζει ο ίδιος ο χρήστης (όπως ο μισθός) αλλά και να αλλάξει στοιχεία άλλων χρηστών βάζοντας την κατάλληλη συνθήκη. Μία εύκολη τροποποίηση μισθού εφόσον ο δράστης γνωρίζει τον τίτλο της στήλης του μισθού είναι να γράψει το ακόλουθο σε οποιοδήποτε πεδίο.

Π.χ. είναι στο πεδίο Phone number : 6912345678', Salary='10000

Εφόσον ο τίτλος της στήλης του μισθού λέγεται Salary, θα ενημερωθεί το τηλέφωνο και ο μισθός του. Ουσιαστικά προσθέτει άλλη μία γραμμή κώδικα στον κώδικα της ιστοσελίδας.

Για να πειράξει τα στοιχεία αλλουνού υπαλλήλου ο εισβολέας μπορεί να γράψει το ακόλουθο:

Π.χ. είναι στο πεδίο Email και θέλει να αλλάξει τον κωδικό ενός άλλου χρήστη. Η περίπτωση της αλλαγής του κωδικού είναι λίγο πιο ιδιαίτερη καθώς ο νέος κωδικός πρέπει πρώτα να μετατραπεί με τον αλγόριθμο SHA1.

Κωδικός = sqlinjection = 665af2cd4e3dce8334baf6c9c70f140bfbc6a255

Οπότε Email:',Password='665af2cd4e3dce8334baf6c9c70f140bfbc6a255' Where Name='(το username που θέλει να κάνει την αλλαγή)'--.

Μέσω αυτής της εντολής μπορεί να αλλάξει οποιόδηποτε άλλο δεδομένο θελήσει ο εισβολέας προσαρμόζοντας κατάλληλα τους τίτλους των στηλών που θέλει να αλλάξει καθώς και την συνθήκη WHERE για να ορίσει σε ποιόν υπάλληλο να γίνει η αλλαγή [30].

Στην περίπτωση που ο χρήστης θέλει να επιτεθεί μέσω ενός πεδίου αναζήτησης (Σχεδόν όλες οι ιστοσελίδες έχουν από ένα) και θέλει να εμφανίσει όλα τα στοιχεία των χρηστών της ιστοσελίδας μπορεί να γράψει τα ακόλουθα. Έστω ότι το ερώτημα SQL είναι το εξής:

SEECT ItemName, ItemDesc FROM Items WHERE ItemID = '…'

Πληκτρολογεί ένα ItemID π.χ. 50' UNION SELECT Username, Password FROM USERS;

Έτσι με το UNION ενώνει 2 εντολές σε μία και εμφανίζει τα στοιχεία των χρηστών [27].

Για το time-based SQLI:

SELECT * FROM table WHERE id = 1-SLEEP(20)

Αν αργήσει να αποκριθεί το σύστημα τότε σημαίνει ότι είναι ευάλωτο σε SQLI επίθεση και λογικά η βάση δεδομένων ειναι MySQL. Στο επόμενο στάδιο στέλνεται το παρακάτω για να διαπιστωθεί η έκδοση της βάσης και έτσι σιγά σιγά αποκτιούνται περισσότερες πληροφορίες [56].

SELECT * FROM table WHERE id = 1-IF(MID(VERSION(),1,1) = '5', SLEEP(20), 0)

ΚΕΦΑΛΑΙΟ 3. Προστασία απο SQL Injection

3.1 Εισαγωγή

Η προστασία της βάσης δεδομένων είναι η καλύτερη πρόληψη για την αποφυγή έκθεσης των δεδομένων που αυτή περιέχει. Είναι πολύ σημαντικό προτού απαριθμηστούν οι τρόποι προστασίας να αναφερθεί ότι θα ήταν πρέπον όλοι όσοι ασχολούνται με την κατασκευή της ιστοσελίδας και την σύνδεση της με την βάση να γνωρίζουν τους κινδύνους και τις ευπάθειες που μπορεί να υποπέσει, γιατί αν δεν γνωρίζει τι μπορεί να βλάψει την βάση πως θα το αντιμετωπίσει; Συνεπώς, πρώτα απ' όλα θα πρέπει να γίνει σωστή πληροφόρηση του προσωπικού και γνωστοποίηση των κινδύνων [39].

3.2 Τρόποι προστασίας απο SQLI

Για την καλύτερη προστασία μίας βάσης ενδεικνυται η τήρηση των ακόλουθων τεχνικών:

1) Χρήση παραμετροποιημένων ερωτημάτων (Parameterized Queries)

2) Χρήση αποθηκευμένων διαδικασιών (Stored Procedures)

3) Επικύρωση τύπου δεδομένων εισόδου (Input Data Type Validation) & Έλεγχος μήκους μεταβλητών εισόδου (Input Variable Length Checking)

4) Δημιουργία λίστας επιτρεπόμενων μεταβλητών (White List Filtering)

5) Δημιουργία λίστας απαγορευμένων μεταβλητών (Black List Filtering)

6) Απομόνωση των δεδομένων που εισάγει ο χρήστης (Escaping Input)

7) Αποφυγή διαχειριστικών προνομίων (Avoiding Administrative Privileges)

8) Αρχή του λιγότερο προνομιούχου (Principle of Least Privilege)

9) Κρυπτογράφηση κωδικών πρόσβασης (Password hashing)

10) Τείχος προστασίας ιστοσελίδας (Website Firewall)

3.2.1 Παραμετροποιημένα ερωτήματα (Parameterized Queries)

Μέσω αυτής της τεχνικής θα έπρεπε να διδάσκονται όλοι οι προγραμματιστές να γράφουν τα ερωτήματα για την βάση δεδομένων. Είναι εύκολα στην σύνταξη τους και πιο κατανοητά απο τα δυναμικά ερωτήματα. Τα παραμετροποιημένα ερωτήματα είναι σαν ένα είδος προ-μεταγλώττισης ενός ερωτήματος SQL έτσι ώστε στην συνέχεια να δοθούν και οι παράμετροι για να εκτελεστεί η εντολή. Έτσι η βάση έχει την δυνατότητα να ξεχωρίζει τον κώδικα με τις μεταβλητές εισόδου που δίνει ο χρήστης. Η είσοδος που στέλνει ο χρήστης μπαίνει αυτόματα σε αυτάκια ‘’ με αποτέλεσμα να προλαμβάνει αρκετά είδη επιθέσεων SQLI.

```
// Define which user we want to find.
String email = "user@email.com";

// Connect to the database.
Connection conn = DriverManager.getConnection(URL, USER, PASS);
Statement stmt = conn.createStatement();

// Construct the SQL statement we want to run, specifying the parameter.
String sql = "SELECT * FROM users WHERE email = ?";

// Run the query, passing the 'email' parameter value...
ResultSet results = stmt.executeQuery(sql, email);

while (results.next()) {
  // ...do something with the data returned.
}
```

Εικόνα 3.2.1.i [45]

Όπως βλέπουμε στο παράδειγμα της παραπάνω εικόνας έχει γίνει χρήση παραμετροποιημένου ερωτήματος στην μέθοδο executeQuery. Το παραμετροποιημένο string και οι παράμετροι μπαίνουν στην βάση ξεχωριστά και εκτελούνται σωστά χωρίς κίνδυνο επίθεσης [45, 46]. Οι επιθέσεις που αποτρέπει είναι όταν χρησιμοποιείται ταυτολογία, σχόλιο στο τέλος της γραμμής, με βάση τον χρόνο και αυτές που έχουν σκοπό να τροποποιήσουν τα δεδομένα. Ακόμη, η εφαρμογή θα τρέχει γρηγορότερα καθώς τα παραμετροποιημένα ερωτήματα καταναλώνουν λιγότερους πόρους συστήματος.

3.2.2 Χρήση αποθηκευμένων διαδικασιών (Stored Procedures)

Μία αποθηκευμένη διαδικασία είναι ένα τμήμα κώδικα που βρίσκεται στην βάση δεδομένων και μπορεί να κληθεί απο το πρόγραμμα ή μία αλλη αποθηκευμένη διαδικασία. Δεν είναι τοσο ασφαλείς όσο τα παραμετροποιημένα ερωτήματα αλλά αν χρησιμοποιηθούν σωστά έχουν το ίδιο αποτέλεσμα. Η διαφορά τους είναι οτι ο κώδικας της αποθηκευμένης διαδικασίας καθορίζεται και παραμένει στην ίδια την βάση δεδομένων και μετά καλείται απο την ιστοσελίδα. Κάποια απο τα

πλεονεκτήματα της είναι ότι: Η εφαρμογή τρέχει πιο γρήγορα, αφού χρειάζεται να μεταγλωττιστεί μία φορά στην αρχή, μπορεί να χρησιμοποιηθεί αρκετές φορές και είναι ασφαλής. Όμως απο την άλλη είναι δύσκολο να εντοπιστούν τυχόν λάθη που έχουν γίνει [47, 48].

```
DELIMITER //
CREATE PROCEDURE GetAllMovies()
BEGIN
        SELECT * FROM MOVIES;
END //
DELIMITER ;
```

Εικόνα 3.2.2.i [48]

Στην εικόνα παραπάνω βλέπουμε ένα απλό παράδειγμα πως φτιάχνεται ένα stored procedure. Καλείται με την εντολή CALL GetAllMovies(); [48].

3.2.3 Επικύρωση τύπου δεδομένων εισόδου (Input Data Type Validation) & Έλεγχος μήκους μεταβλητών εισόδου (Input Variable Length Checking)

Οι επιθέσεις SQLI μπορούν να πραγματοποιηθούν από οποιοδήποτε πεδίο συμπλήρωσης στοιχείων είτε αυτή είναι προορισμένη να δεχθεί μόνο χαρακτήρες (π.χ. όνομα) είτε μόνο αριθμούς (π.χ. τηλέφωνο). Με βάση αυτή την ιδεολογία αποτρέποντας την βάση να δεχθεί χαρακτήρες σε πεδίο που ήταν να συμπληρωθούν μόνο αριθμοί, έχει ήδη μειώσει τις πιθανότητες επίθεσης. Επίσης περιορίζοντας το μήκος που θα δέχεται σαν απάντηση σε κάθε πεδίο συμπλήρωσης (ανάλογα το ζητούμενο) να μην ξεπερνάει ένα όριο, είναι βέβαιο ότι σε περίπτωση παραβίασης της βάσης, ο εισβολέας δεν θα μπορέσει να ενώσει άπειρες έντολες και να τις στείλει προς την βάση, περιορίζοντας έτσι την ζημιά. Αυτά τα μέτρα ασφαλείας προτρέπουν επιθέσεις τύπου Union, Blind, Tautology και Time-based [46].

3.2.4 Δημιουργία μιας λίστας επιτρεπόμενων μεταβλητών (White List Filtering)

Η White List είναι μία λίστα που την δημιουργεί ο κατασκευαστής της βάσης δεδομένων, η οποία φιλτράρει τις εισόδους του κάθε χρήστη και αποφασίζει αν θα περάσει το ερώτημα SQL στην βάση. Είναι συχνό φαινόμενο η χρήση μιας Black List αλλά δεν ενδείκνυται καθώς δεν αποτρέπει ενδεχόμενες επιθέσεις στο έπακρο. Οι Black Lists συνήθως στοχεύουν στην αποτροπή επιθέσεων που περιέχουν ταυτολογίες (OR 1=1), αποστρόφους (‘’), διάφορα tags όπως <script> τα οποία είναι εύκολο να παρακαμφθούν από έναν έμπειρο hacker. Η White List από την άλλη μπορεί να χρησιμοποηθεί και να φιλτράρει όλα τα πεδία που είναι να συμπληρωθούν απο τον χρήστη και καθορίζει ακριβώς τι επιτρέπεται να εισέλθει στη βάση, ενώ όλα τα άλλα απορρίπτονται. Άν η

είσοδος έχει να κάνει με email, Τ.Κ. , ονοματεπώνυμα, ημερομηνίες κτλπ τότε πρέπει να οριστεί ένα πολύ δυνατό πρότυπο πιστοποίησης που συνήθως γίνεται με κανονικές εκφράσεις. Άν η είσοδος είναι απο λίστες με προσχεδιασμένες τιμές δηλαδή πεδία που επιλέγεις κάποια απο τις διαθέσιμες τιμές, τότε στην πιστοποίηση θα πρέπει να ταιριάζει απόλυτα με την τιμή που περιέχει στην λίστα σε περίπτωση που έχει προσπαθήσει να προσθέσει εκεί κακόβουλες εντολές [47, 49].

3.2.5 Δημιουργία λίστας απαγορευμένων μεταβλητών (Black List Filtering)

Προηγουμένως αναφέρθηκε οτι δεν ενδείκνυται η χρησιμοποίηση των Black List και είναι πολύ ορθό. Όμως το να φτιαχτεί μια Black List σε συνδυασμό με μία White List είναι ακόμη πιο σωστό και ασφαλέστερο. Μία Black List μπορεί να περιέχει εντολές ή σύμβολα που απαγορεύεται να χρησιμοποιήσει ο χρήστης όπως για παράδειγμα: ‘ , -- , /* */, @, AND, OR, DROP, DELETE, CREATE, ALTER, INSERT, UPDATE και πολλές άλλες λέξεις που χρησιμοποιούνται απο την βάση για τον χειρισμό των δεδομένων της με σκοπο πάντα να μην αποκτήσει πρόσβαση ο εισβολέας [47].

3.2.6 Απομόνωση των δεδομένων που εισάγει ο χρήστης (Escaping Input)

Κάθε σύστημα διαχείρισης βάσης δεδομένων περιέχει κάποια προσχεδιασμένα κομμάτια κώδικα για την αποφυγή των στοιχείων που εισάγει ο χρήστης με σκοπό να μην συνδυαστεί με τον κώδικα του προγραμματιστή και σταλθεί κακόβουλο ερώτημα προς την βάση. Δηλαδή λειτουργεί σαν άμυνα λίγο πριν μπεί στην βάση δεδομένων. Συνίσταται μόνο για χρήση σε παλιούς κώδικες για την εκσυχρόνιση τους και σε άλλες περιπτώσεις σαν έσχατη λύση καθώς δεν είναι τόσο αποδοτικό. Είναι ορθότερη η χρησιμοποίηση της White List εφόσον μπορεί ευκολότερα να εμποδίσει επιθέσεις SQLI [46, 47].

```php
<?php
$mysqli = new mysqli("localhost","my_user","my_password","my_db");

if ($mysqli -> connect_errno) {
  echo "Failed to connect to MySQL: " . $mysqli -> connect_error;
  exit();
}

// Escape special characters, if any
$firstname = $mysqli -> real_escape_string($_POST['firstname']);
$lastname = $mysqli -> real_escape_string($_POST['lastname']);
$age = $mysqli -> real_escape_string($_POST['age']);

$sql="INSERT INTO Persons (FirstName, LastName, Age) VALUES ('$firstname', '$lastname', '$age')";

if (!$mysqli -> query($sql)) {
  printf("%d Row inserted.\n", $mysqli->affected_rows);
}

$mysqli -> close();
?>
```

Εικόνα 3.2.6.i [50]

Στην εικόνα παραπάνω βλέπουμε την χρήση της μεθόδου mysqli_real_escape_string(). Η μέθοδος απομονώνει τους ειδικούς χαρακτήρες ,αν υπάρχουν, και το στέλνει φιλτραρισμένο στην βάση δεδομένων [50].

3.2.7 Αποφυγή διαχειριστικών προνομίων (Avoiding Administrative Privileges)

Για να τηρηθεί αυτό το μέτρο προστασίας πρέπει ο λογαριασμός που έχει πλήρη διαχειριστικά προνόμια (root access) της ιστοσελίδας να μην συνδέεται με την βάση δεδομένων, μόνο σε ακραίες περιπτώσεις που είναι απολύτως απαραίτητο καθώς αν ένας εισβολέας αποκτήσει πρόσβαση με αυτόν τον λογαριασμό εκθέτει όλη την ιστοσελίδα και την βάση μαζί. Ακόμη και λογαριασμοί με λιγότερα δικαιώματα μπορούν να βλάψουν την βάση αλλά έτσι περιορίζεται η ζημιά. Συνίσταται η χρήση της αρχής Least Privilege που θα αναλυθεί παρακάτω [46].

3.2.8 Αρχή του λιγότερο προνομιούχου (Principle of Least Privilege)

Για να ελαχιστοποιηθεί το ρίσκο μιας επιτυχημένης επίθεσης SQLI, θα πρέπει να εφαρμόσουμε αυτή την αρχή. Ο κανόνας είναι απλός, όπως οι υπάλληλοι μιας επιχείρησης έχουν διαφορετικές βαθμίδες και προνόμια έτσι θα πρέπει να λειτουργούν και οι λογαριασμοί στην βάση δεδομένων [45].

Για την καλύτερη εφαρμογή της αρχής, σωστό θα ήταν ο προγραμματιστής να ξεκινήσει από τους λογαριασμούς με τα λιγότερα προνόμια, να διαπιστώσει τι δικαιώματα θα πρέπει να έχουν για την σωστή λειτουργία τους και όχι ποια δικαιώματα πρέπει να καταργήσει, ανεβαίνοντας σιγά σιγά προς τα πάνω της ιεραρχίας, ακολουθώντας την ίδια τακτική. π.χ. σε έναν λογαριασμό που έχει δικαίωμα να κάνει μόνο ανάγνωση των περιεχομένων ενός πίνακα πρέπει να σιγουρευτούμε ότι τα άλλα δικαιώματα απαγορευόνται (τροποποίηση και διαγραφή δεδομένων).

Μία άλλη τεχνική επι του θέματος που είναι και πολύ αποδοτική είναι να δημιουργήσουμε μία όψη του πίνακα στον οποίο έχει πρόσβαση ο κάθε χρήστης (Μία όψη ενός πίνακα είναι η αντιγραφή του αυθεντικού πίνακα αλλά άν γίνουν τροποποιήσεις δεν έχουν αντίκτυπο στον αυθεντικό) .

Άν στη βάση είναι έτσι δομημένος ο κώδικας όπου όλα λειτουργούν με αποθηκευμένες διαδικασίες, τα δικαιώματα των χρηστών θα πρέπει να περιοριστούν στο να εκτελούν μόνο αυτές τις διαδικασίες, αφαιρώντας έτσι το δικαίωμα να στέλνουν τα δικά τους SQL ερωτήματα απευθείας στην βάση δεδομένων .

Κατα την λειτουργία της ιστοσελίδας τα δικαιώματα των χρηστών θα πρέπει να περιοριστούν έτσι ώστε να μπορούν να εκτελούν μόνο δηλώσεις DML και οχι DDL, δηλαδή μόνο τροποποίηση των δεδομένων αλλά όχι τροποποίοησης της δομής των πινάκων της βάσης καθώς αυτό σπάνια χρειάζεται να αλλαχτεί ενώ είναι online η βάση. Αυτό συνήθως γίνεται όταν η βάση είναι offline και παραχωρούνται προσωρινά αυξημένα δικαιώματα στους χρήστες που διαχειρίζονται την βάση [47].

3.2.9 Κρυπτογράφηση κωδικών πρόσβασης (Password hashing)

Το να αποθηκεύονται οι κωδικοί σαν απλό κείμενο χωρίς να είναι κρυπτογραφημένοι είναι απο μόνο του μία ευπάθεια. Μόλις αποκτήσει πρόσβαση ο εισβολέας στην βάση, έχει αυτόματα και τους κωδικούς όλων των χρηστών που έχουν δημιουργήσει λογαριασμό στην ιστοσελίδα και πολύ συχνά αυτοί οι κωδικοί είναι ίδιοι με άλλους λογαριασμούς σε άλλες ιστοσελίδες με αποτέλεσμα να επεκτείνεται η διαρροή πληροφοριών. Οι κωδικοί θα πρέπει να κρυπτογραφούνται με κρυπτογραφήσεις τύπου MD5, SHA-1 ή SHA-2 [51] μάζι με ένα χρονικό token το οποίο αλλάζει συνεχώς ή μαζί με κάποιο άλλο τυχαίο δεδομένο ονόματι salt το οποίο θα αποθηκεύεται ξεχωριστά έτσι ώστε αν διαρρεύσουν οι κωδικοί να μην είναι αξιοποιήσιμοι. Μία επιπλέον ασπίδα

προστασίας είναι να αποσύρει όλων των χρηστών τα δικαιώματα απο τον πίνακα με τους κωδικούς (εκτός του admin) και αντ' αυτού να εκτυπώνεται η όψη (όπως αναφέρθηκε στην προηγούμενη υποενότητα) με τους κρυπτογραφημένους κωδικούς, αν κάποιος επιχειρήσει να τους υποκλέψει. Να σημειωθεί ότι όσοι κωδικοί είναι κρυπτογραφημένοι με τους παραπάνω τρόπους είναι πολύ δύσκολο να σπάσουν έως και αδύνατο [45, 47].

3.2.10 Τείχος προστασίας ιστοσελίδας (Website Firewall)

Τελευταίο αλλά εξίσου σημαντικό είναι οι ιστοσελίδες να προστατεύονται από ένα τείχος προστασίας ιστοσελίδας. Ελέγχει σε πραγματικό χρόνο όλη την εισαγωγή και εξαγωγή πληροφοριών που πραγματοποιείται στον διακομιστή της ιστοσελίδας, κάνοντας εκτίμηση αν πραγματοποιείται κάποια επίθεση ή αν υπάρχει κάποια επικείμενη απειλή. Παρακολουθεί τις αιτήσεις POST & GET και μπλοκάρει κακόβουλες εντολές. Ακόμη, η ιστοσελίδα προστατεύεται όχι μόνο απο SQLI αλλά και απο αρκετές άλλες επιθέσεις όπως DDoS, XSS, Cookie poisoning κ.α. [46].

Συνοψίζοντας, κάθε μία απο τις τεχνικές προστασίας που αναλύθηκαν παραπάνω βοηθάει στην ασφαλέστερη λειτουργία μιας βάσης δεδομένων αποτρέποντας βέβαια μερικούς απο τους τύπους επιθέσεων SQLI. Συνδυάζοντας όμως παραπάνω από έναν μηχανισμό προστασίας ή εφαρμόζοντας και όλους, έχουμε πολύ μεγαλύτερη ασφάλεια, θωρακίζοντας την βάση και η πιθανότητα για μία επιτυχή επίθεση εκμηδενίζεται. Οπότε εν' ολίγοις, η εκπαίδευση και πληροφόρηση όσων εμπλέκονται στην δημιουργία της βάσης είναι απαραίτητη, το φιλτράρισμα κάθε εισόδου απο τους χρήστες θα πρέπει να γίνεται σε κάθε περίπτωση, η χρησιμοποίηση των White Lists σε συνδυασμό με Black Lists δίνει μεγάλο ποσοστό ασφάλειας, να γίνεται ενημέρωση σε πιο πρόσφατες διαθέσιμες εκδόσεις των εργαλείων που χρησιμοποιούνται (διακομιστές, ιστοσελίδες, γλώσσες κτλπ) και όσο το δυνατόν πιο τακτικούς ελέγχους στην βάση για τυχόν ευπάθειες.

ΚΕΦΑΛΑΙΟ 4. Ανίχνευση επίθεσης SQLI

4.1 Εισαγωγή

Ακόμη και σήμερα υπάρχουν χιλιάδες ιστοσελίδες που είναι ευάλωτες σε επίθεση SQLI. Κάποιες απο αυτές έχουν παραβιαστεί και δεν το έχουν ανιχνεύσει με αποτέλεσμα ο εισβολέας να επωφελείται μακροχρόνια. Γιαυτό το λόγο η ανίχνευση της επίθεσης παίζει μεγάλο ρόλο.

Ακόμη και αν όλες οι προφυλάξεις έχουν παρθεί, είναι σημαντικό να μπορούμε να εντοπίσουμε μία επίθεση που γίνεται προς την βάση και αν αυτή ήταν επιτυχής ή όχι. Υπάρχουν πολλές επιλογές στον εντοπισμό της επίθεσης SQLI και αυτές θα αναλυθούν παρακάτω.

4.2 Ανίχνευση επίθεσης SQLI

Για να αναλυθεί σωστότερα η ανίχνευση, είναι ζωτικής σημασίας να κατανοηθεί πότε αλληλεπιδρά ο διακομιστής της βάσης με την ιστοσελίδα. Οι περιπτώσεις χωρίζονται σε:

Α) Όταν πραγματοποιείται έλεγχος ταυτότητας χρήστη (login). Οι περισσότερες των πιθανοτήτων είναι ότι η επιβεβαίωση γίνεται μέσω μίας βάσης δεδομένων που περιέχει όλους τους κωδικούς των χρηστών.

Β) Όταν γίνεται αναζήτηση κάποιου προϊόντος ή κάποιας υπηρεσίας μέσω ενός πεδίου αναζήτησης στην ίδια την ιστοσελίδα. Η λέξη/εις που δίνει ο χρήστης διασταυρώνεται με τα δεδομένα της βάσης και επιστρέφει σαν αποτέλεσμα όλα τα συσχετιζόμενα δεδομένα (Προφανώς είναι διαφορετικός πίνακας απο αυτόν με τους κωδικούς).

Γ) Όταν γίνεται αναζήτηση μέσω μιάς μηχανής αναζήτησης (π.χ. Google). Τα δεδομένα που εισάγει ο χρήστης διασταυρώνονται και αυτά με κάποια βάση [55].

Για την μέγιστη δυνατή ανίχνευση, προτείνεται να δημιουργηθεί μία λίστα που περιλαμβάνει όλα τα πεδία που μπορούν να συμπληρωθούν απο τον χρήστη και θα δημιουργήσουν ενα ερώτημα SQL προς την βάση, συμπεριλαμβανομένων των κρυφών αιτημάτων POST και στη συνέχεια να αληλλεπιδράσουν με το κάθε πεδίο ξεχωριστά προσπαθώντας να το παρακάμψουν και να εκμαιεύσουν πληροφορίες που δεν θα έπρεπε να έχουν πρόσβαση [55]. Παρακάτω θα αναλυθούν συνοπτικά τα αιτήματα POST και GET που δημιουργούνται απο τον περιηγητή για την καλύτερη κατανόηση της ανίχνευσης μέσω αυτών.

Αιτήματα GET:

Τα αιτήματα GET τα οποία τα δημιουργεί ο περιηγητής της ιστοσελίδας, είναι μία μέθοδος του HTTP πρωτοκόλλου (Hypertext Transfer Protocol) όπου ζητά απο τον διακομιστή της ιστοσελίδας να εμφανιστούν όποιες πληροφορίες αναφέρονται στο link του URL [65]. π.χ. :

```
GET /search.aspx?text=lcd%20monitors&cat=1&num=20 HTTP/1.1
Host:www.victim.com
User-Agent: Mozilla/5.0 (X11; U; Linux x86_64; en-US; rv:1.8.1.19)
Gecko/20081216 Ubuntu/8.04 (hardy) Firefox/2.0.0.19

Accept: text/xml,application/xml,application/xhtml+xml,
text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5
Accept-Language: en-gb,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Proxy-Connection: keep-alive
```

Εικόνα 4.2.i & Εικόνα 4.2.ii [65]

Υπάρχουν 3 παράμετροι στο παραπάνω παράδειγμα, το text, το cat και το num στις οποίες μπορούν να αλλαχτούν οι τιμές τους χειροκίνητα και από την γραμμή του URL πάνω στον browser. Ακόμη αποστέλλονται πληροφορίες όπως η έκδοση του περιηγητή, η γλώσσα , η κωδικοποίηση κτλπ [65].

Αιτήματα POST:

Τα αιτήματα POST είναι και αυτά μία μέθοδος του HTTP για να στείλει πληροφορίες στον διακομιστή της ιστοσελίδας. Αυτό το αίτημα δημιουργείται όταν συμπληρωθεί μία φόρμα με στοιχεία και πατηθεί το κουμπί να αποσταλθούν τα δεδομένα. π.χ. :

```
POST /contact/index.asp HTTP/1.1
Host:www.victim.com
User-Agent: Mozilla/5.0 (X11; U; Linux x86_64; en-US; rv:1.8.1.19) Gecko/20081216
Ubuntu/8.04 (hardy) Firefox/2.0.0.19
Accept: text/xml,application/xml,application/xhtml+xml,
text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5
Accept-Language: en-gb,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Referer: http://www.victim.com/contact/index.asp
Content-Type: application/x-www-form-urlencoded
Content-Length: 129

first=John&last=Doe&email=john@doe.com&phone=555123456&title=Mr&country=US&comments=
I%20would%20like%20to%20request%20information
```

Εικόνα 4.2.iii [65]

Όπως στα αίτηματα GET έτσι και στα POST στέλνονται οι πληροφορίες του περιηγητή και στο τέλος της εικόνας φαίνονται τα στοιχεία που συμπληρώθηκαν απο τον χρήστη [65].

Στην ανίχνευση, για αρχή, ένας πολύ απλός έλεγχος είναι η συμπλήρωση μιας απόστροφου (‘) ή ενός ελληνικού ερωτηματικού (;). Η απόστροφος σηματοδοτεί την ολοκλήρωση ενός string, ενώ το ελληνικό ερωτηματικό την ολοκλήρωση μιας εντολής. Και τα δύο αν δεν φιλτραριστούν προτού φτάσουν στην βάση θα οδηγήσουν σε κάποιο σφάλμα ή σε προσπέλαση ελέγχου ταυτότητας. Ακόμη μπορούν να εισαχθούν οριοθέτες σχολίου (-- ή /* */) και λογικές εκφράσεις όπως AND και OR. Τέλος η συμπλήρωση ενός αριθμού σε ένα πεδίο που επιτρέπει μόνο χαρακτήρες ή το ανάποδο [55].

Οι επιθέσεις συνήθως είναι περίπλοκες και πιθανόν αόρατες, αφού ο εισβολέας κάνει αρκετές δοκιμές στέλνοντας πολλαπλά ερωτήματα στην SQL τα οποία περιλαμβάνουν και ερωτήματα στα οποία επιστρέφονται κάποιες εγγραφές σφαλμάτων. Από τις εγγραφές αυτές μαζεύει τις πληροφορίες που χρειάζεται για να καταφέρει να εισβάλλει. Ο εισβολέας εδώ βασίζεται στο γεγονός ότι τα σφάλματα αυτά δεν καταγράφονται καθόλου ή οτι δεν εντοπίζονται απο το σύστημα παρακολούθησης της βάσης. Οπότε το σύστημα παρακολούθησης για να καταγράψει τέτοια σφάλματα που παραπέμπουν σε SQLI πρέπει να έχει ενεργοποιηθεί η επιλογή εκτεταμένα συμβάντα. Τα σφάλματα αυτά όμως παρέχουν και στον προγραμματιστή πολύτιμες πληροφορίες για την διόρθωση τους και γιαυτό πρέπει να παρακολουθούνται και να καταγράφονται, αλλά σε ασφαλές αρχείο με περιορισμένη πρόσβαση [33, 53].

Είναι πολύ συχνό φαινόμενο να υπάρχουν τέστ παραβίασης στους δοκιμαστικούς ελέγχους της βάσης δεδομένων προτού λειτουργήσει, για να βεβαιωθούν ότι τα ευαίσθητα δεδομένα που δεν θέλουν να διαρρεύσουν είναι ασφαλή και ότι μπορεί να αποτρέψει μία τέτοια επίθεση [33].

Υπάρχουν διάφορες ιστοσελίδες που προσφέρουν την υπηρεσία να σαρώσουν την βάση δεδομένων μας και να μας απαριθμήσουν τις ευπαθειές της, άν αυτές υπάρχουν. Η σάρωση περιλαμβάνει την καταγραφή δεδομένων χειρισμού (DML) και ορισμού (DDL) δηλαδή όταν αλλαχτούν οι κωδικοί πρόσβασης ή τα δικαιώματα κάποιου χρήστη ή αν έχει γίνει επαναφορά του συστήματος κτλπ. Καλό θα ήταν να γίνονται τακτικοί έλεγχοι σε μία βάση έτσι ώστε να γνωρίζουμε άν έχει παραβιαστεί ή όχι και με ποιόν τρόπο. [33, 34, 35] Μερικά απο τα διαθέσιμα εργαλεία για σάρωση της βάσης είναι τα εξής:

i) **<u>SQLMap:</u>**

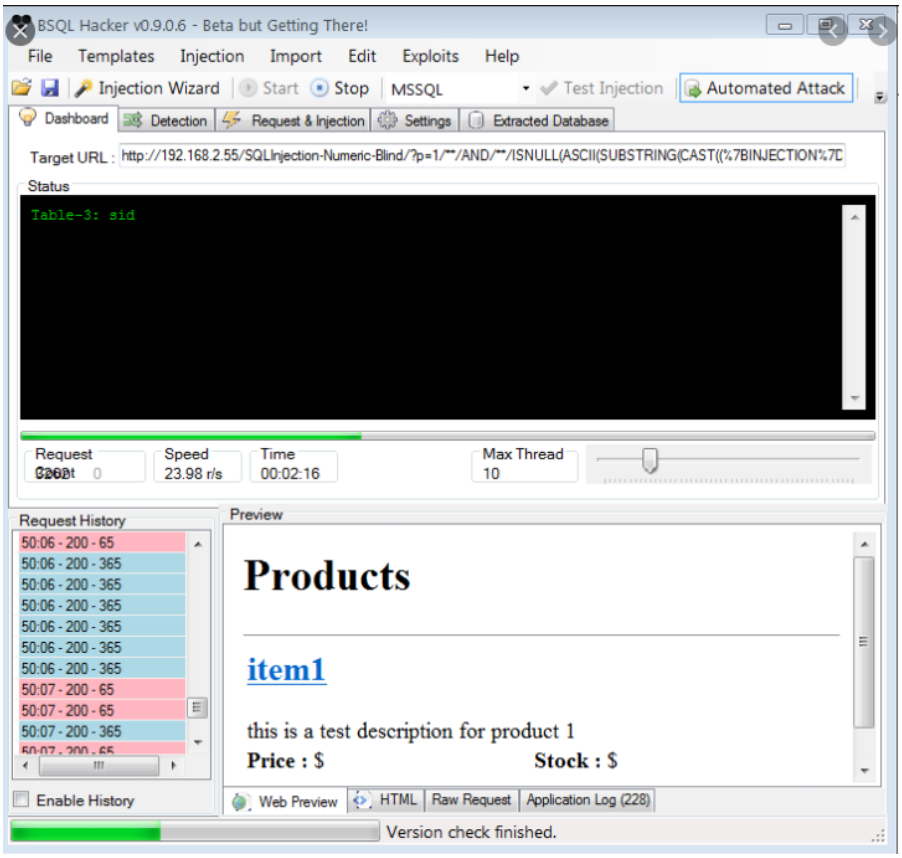
Το SQLMap είναι εφαρμογή ανοιχτού κώδικα και χαρακτηρίζεται ως το πιο γνωστό εργαλείο για αυτού του είδους επίθεσης. Υποστηρίζει την σάρωση σχεδόν σε όλες τις βάσεις δεδομένων (συμπεριλαμβανομένων των MySQL, Oracle, Microsoft SQL Server, Microsoft Access, PostgreSQL και άλλες). Εντοπίζει τις περισσότερες απο τις ευπάθειες SQLI (Boolean-based, time-based, error-based, UNION-based, stacked queries και out-of-band) και παρέχει μία δυνατότητα όπου προσπαθεί να αποσπάσει τον κωδικό πρόσβασης και έπειτα να αποκρυπτογραφήσει το hash του κωδικού αυτού με επίθεση τύπου dictionary attack. Αφού συνδεθεί με την βάση μπορεί να κάνει αναζήτηση για κάποια συγκεκριμένη βάση ή πίνακα ή στήλη εντός της βάσης. Για τις βάσεις MySQL, PostgreSQL και Microsoft SQL Server υπάρχει

επιπλέον δυνατότητα να ανεβει ή να κατεβει κάποιο αρχείο απο τον server και να εκτελεί αυθαίρετες εντολές [44, 63].

```
$ python sqlmap.py -u "http://debiandev/sqlmap/mysql/get_int.php?id=1" --batch
        ___
       __H__
 ___ ___[,]_____ ___ ___  {1.3.4.44#dev}
|_ -| . [']     | .'| . |
|___|_  [,]_|_|_|__,|  _|
      |_|V...       |_|   http://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent i
s illegal. It is the end user's responsibility to obey all applicable local, state and fed
eral laws. Developers assume no liability and are not responsible for any misuse or damage
 caused by this program

[*] starting @ 10:44:53 /2019-04-30/

[10:44:54] [INFO] testing connection to the target URL
[10:44:54] [INFO] heuristics detected web page charset 'ascii'
[10:44:54] [INFO] checking if the target is protected by some kind of WAF/IPS
[10:44:54] [INFO] testing if the target URL content is stable
[10:44:55] [INFO] target URL content is stable
[10:44:55] [INFO] testing if GET parameter 'id' is dynamic
[10:44:55] [INFO] GET parameter 'id' appears to be dynamic
[10:44:55] [INFO] heuristic (basic) test shows that GET parameter 'id' might be injectable
 (possible DBMS: 'MySQL')
```

Εικόνα 4.2.iv

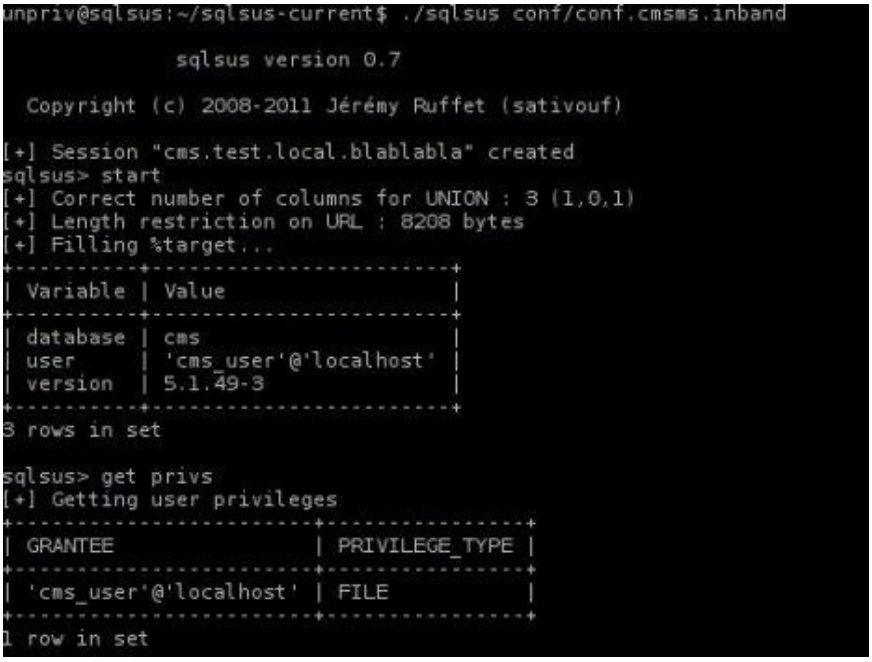
ii) **<u>BSQL Hacker:</u>**



Εικόνα 4.2.v

Το BSQL Hacker είναι ένα αυτοματοποιήμενο εργαλείο για επιθέσεις SQLI και ειδικεύεται στις Blind SQLI. Δηλαδή υποστηρίζει τις επιθέσεις time-based, error-based και deep-blind (προχωρημένου επιπέδου χρονοκαθυστέρηση). Υποστηρίζει πολλαπλά σημεία για επίθεση όπως κεφαλίδες HTTP, αιτήματα POST, cookies και ερωτήματα συμβολοσειράς. Είναι για τις βάσεις Microsoft SQL Server, Oracle και MySQL (πειραματικό επίπεδο) [63].

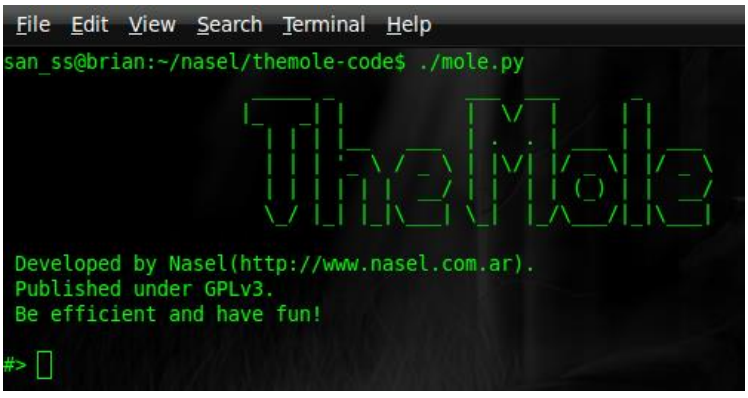
iii) **SQLSus:**



Εικόνα 4.2.iv

Το SQLSus είναι εργαλείο ανοιχτού κώδικα και είναι μόνο για επίθεση SQLI στην MySQL. Το εργαλείο έχει γραφτεί σε γλώσσα Perl και μπορεί να προστεθεί κώδικας για επιπλέον δυνατότητες. Χρησιμοποιεί πολλαπλά νήματα για την γρηγορότερη επεξεργασία και εκτέλεση της επίθεσης. Ειδικεύεται στην επίθεση τύπου blind SQLI και κάνει επίθεση μέσω αιτημάτων POST και GET. Μπορεί να ανακτήσει δυαδικά δεδομένα, να αυθεντικοποιήσει το HTTP και να ελέγξει τα cookies [63].

iv) **Mole:**

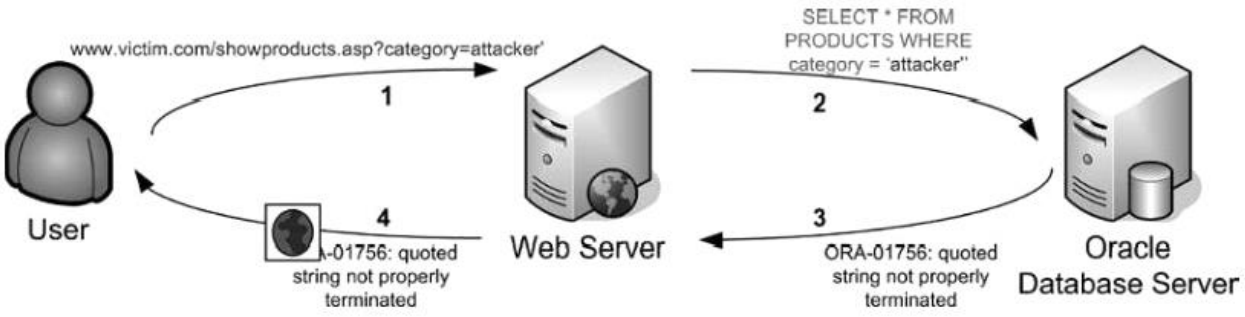


Εικόνα 4.2.v

Το Mole είναι εργαλείο ανοιχτού κώδικα γραμμένο σε Python και υποστηρίζει επιθέσεις SQLI για τις βάσεις MySQL, Microsoft SQL Server και PostgreSQL. Είναι και αυτό αυτοματοποιημένο εργαλείο και το μόνο που χρειάζεται είναι να συμπληρώσεις την διεύθυνση URL για να πραγματοποιήσει επιθέσεις τύπου Union και Boolean αλλά και επιθέσεις μέσω αιτημάτων POST, GET και cookies. Τέλος, προσφέρει και την δυνατότητα για χειροκίνητες επιθέσεις αλλά αυτό προϋποθέτει να γνωρίζει τις εντολές ο χρήστης [63].

4.3 Συνηθισμένα μηνύματα σφάλματος

Σε μία προσπάθεια ενός εισβολέα να στείλει κάποιο κακόβουλο ερώτημα SQLI συμβαίνει η παρακάτω διαδικασία:



Εικόνα 4.3.i [65]

Στην αρχή ο εισβολέας κάνει αναζήτηση την λέξη attacker και βάζει μία απόστροφο στο τέλος της φράσης. Έπειτα ο διακομιστής ιστού δημιουργεί το ερώτημα SQL και ο διακομιστής της βάση δεδομένων επιστρέφει σφάλμα καθώς υπάρχει μία απόστροφος που δεν χρησιμεύει κάπου. Το σφάλμα αυτό εμφανίζεται στον εισβολέα, αλλά ανάλογα τον προγραμματιστή μπορεί να ορίσει να μην εμφανίζεται το σφάλμα στον χρήστη ή να του εμφανίζεται κάποιο άλλο μήνυμα λάθους για να τον αποπλανήσει και να μην καταφέρει να εισβάλλει στην βάση [65].

Τα σφάλματα που θα παρουσιαστούν παρακάτω δεν χρειάζεται να απομνημονευτούν απο έναν προγραμματιστή ή απο έναν εισβολέα, αρκεί μόνο η αναγνώριση τους και ο λόγος που εμφανίζονται [65].

4.3.1 Σφάλματα στην Oracle

Προσπαθώντας να γίνει παραβίαση σε εφαρμογή Java με Oracle ως βάση δεδομένων είναι συχνή η εμφάνιση του παρακάτω σφάλματος:

```
java.sql.SQLException: ORA-00933: SQL command not properly ended at
oracle.jdbc.dbaccess.DBError.throwSqlException(DBError.java:180) at
oracle.jdbc.ttc7.TTIoer.processError(TTIoer.java:208)
```

Εικόνα 4.3.1.i [65]

Το σφάλμα αυτό είναι πολύ γενικό και εκδηλώνει ότι έχει γίνει κάποιο συντακτικό λάθος. Αν η εισαγωγή του χρήστη είναι μία απλή απόστροφος τότε αυτό το σφάλμα θα εμφανιστεί:

```
Error: SQLException java.sql.SQLException: ORA-01756: quoted string not
properly terminated
```

Εικόνα 4.3.1.ii [65]

Αντίστοιχα μηνύματα εμφανίζονται σε .NET περιβάλλοντα και PHP [65].

4.3.2 Σφάλματα στο Microsoft SQL Server

Υποθέτοντας ότι ο εισβολέας βρίσκεται σε μία ιστοσελίδα και στο URL βλέπει το παρακάτω: http://.....com/showproduct.aspx?id=2

Αλλάζει το id=2 σε id=attacker και βλέπει το παρακάτω σφάλμα:

```
Server Error in '/' Application.
Invalid column name 'attacker'.
Description: An unhandled exception occurred during the execution of the
current web request. Please review the stack trace for more information
about the error and where it originated in the code.

Exception Details: System.Data.SqlClient.SqlException: Invalid column name
'attacker'.
```

Εικόνα 4.3.2.i [65]

Βασιζόμενος στο σφάλμα καταλαβαίνει ότι το ερώτημα SQL είναι της μορφής:

SELECT * FROM products WHERE idproduct=2

Στο idproduct ο server περιμένει μια αριθμητική τιμή και εφόσον δεν την δέχεται υποθέτει ότι η αλφαριθμητική τιμή που δόθηκε είναι στήλη κάποιου πίνακα της βάσης.

Ένας τρόπος για να αποσπάσει πληροφορίες όπως, την έκδοση του server, είναι να δημιουργήσει ένα σφάλμα όπως πρίν και να γράψει το εξής:

http://.....com/showproduct.aspx?category=bikes' and 1=0/@@version;-- [65]

```
Server Error in '/' Application.
Syntax error converting the nvarchar value 'Microsoft SQL Server 2000 -
8.00.760 (Intel X86) Dec 17 2002 14:22:05 Copyright (c) 1988-2003 Microsoft
Corporation Enterprise Edition on Windows NT 5.2 (Build 3790: ) ' to a
column of data type int.
Description: An unhandled exception occurred during the execution of the
current web request. Please review the stack trace for more information
about the error and where it originated in the code.
```

Εικόνα 4.3.2.ii [65]

Η βάση δεδομένων έστειλε σφάλμα προσπαθώντας να μετατρέψει την έκδοση της και έτσι εμφανίζει το περιεχόμενο της μεταβλητής version. Η ίδια τακτική μπορεί να ακολουθηθεί για να παρθούν πληροφορίες όπως ονόματα στηλών, πινάκων κτλπ [65].

4.3.3 Σφάλματα στην MySQL

Ένα σφάλμα που υποδεικνύει ότι η βάση δεδομένων MySQL είναι πιθανώς ευπαθής σε επιθέσεις SQLI είναι το ακόλουθο:

```
Warning: mysql_fetch_array(): supplied argument is not a valid MySQL result
resource in /var/www/      .com/showproduct.php on line 8
```

Εικόνα 4.3.3.i [65]

Στην MySQL τα σφάλματα είναι παρόμοια με της Microsoft SQL Server. Μια διαδικασία που εμπεριέχεται στην PHP και έχει ενδιαφέρον ονομάζεται mysql_error. Αυτή η διαδικασία καταγράφει τα σφάλματα που στέλνει η βάση δεδομένων κατα την διάρκεια εκτέλεσης του ερωτήματος SQL και μπορούν να σταλθούν πίσω στην ιστοσελίδα σαν απλό κείμενο [65].

4.4 Ελέγχος του πηγαίου κώδικα

Συχνά, μία γρήγορη μέθοδος για να ανακαλυφθούν τρωτά σημεία για επίθεση SQLI είναι η επανεξέταση του πηγαίου κώδικα της ιστοσελίδας. Σε αυτό που θα πρέπει να επιμείνει ο προγραμματιστής και να ελέγξει διεξοδικά είναι αν τα δεδομένα που εισέρχονται σε κάποιο SQL ερώτημα προέρχονται κατευθείαν απο τον χρήστη ή αν έχουν σωστά υποστεί κάποια κωδικοποίηση έτσι ώστε να μην είναι ευπαθές το ερώτημα. Υπάρχουν 2 τρόποι ανάλυσης του πηγαίου κώδικα, ο στατικός και ο δυναμικός. Ο στατικός περιλαμβάνει την ανάλυση του κώδικα χωρίς να εκτελεστεί ενώ ο δυναμικός με την εκτέλεση του. Ο έλεγχος μπορεί να γίνει χειροκίνητα ή χρησιμοποιώντας κάποιο εργαλείο τρίτου για να κάνει αυτή την δουλειά. Για τον χειροκίνητο έλεγχο ο προγραμματιστής μπορεί να βοηθηθεί γράφοντας μικρά script κάνοντας αυτοματη την διαδικασία δίνοντας βάση στα ερωτήματα και τις διαδικασίες (αποθηκευμένες ή μη) που δημιουργούν SQL ερωτήματα και αν φιλτράροντα τα δεδομένα που εισάγει ο χρήστης προτού καταλήξουν μέσα στο SQL ερώτημα. Για τον αυτοματοποιημένο έλεγχο θα αναφερθούν παρακάτω κάποια διαθέσιμα εργαλεία και τα χαρακτηριστικά τους ανάλογα βέβαια και την γλώσσα που είναι γραμμένη η ιστοσελίδα [65].

1) Veracode :

Υποστηρίζει σχεδόν όλες τις γλώσσες προγραμματισμού και μπορεί να κάνει μέχρι και 5 διαφορετικές αναλύσεις του κώδικα (στατική, δυναμική, σύνθεση λογισμικού, διαδραστικό έλεγχο ασφαλείας εφαρμογής και τέστ διείσδυσης) [66].

2) SonarQube :

Υποστηρίζει και αυτό σχεδόν όλες τις γλώσσες προγραμματισμού και είναι εργαλείο ανοιχτού κώδικα. Κάνει συνεχούς ελέγχους του κώδικα για ανίχνευση σφαλμάτων και ευπάθειες ασφάλειας [66].

4.5 Πραγματικές επιθέσεις SQLI

- VTech:
  Τα προσωπικά στοιχεία σχεδόν 5 εκατομμυρίων γονέων και 200 χιλιάδων παιδιών εκτέθηκαν απο επίθεση SQLI στην κινεζική εταιρεία παιχνιδιών VTech τον Νοέμβριο του 2015. Τα στοιχεία που εκτέθηκαν περιλαμβάνουν ονόματα, email, κωδικούς, διευθύνεις κατοικιών καθώς και τα ονόματα και τα γενέθλια των παιδιών. Διασταυρώνοντας αυτές τις πληροφορίες μπορούν να συνδεθούν τα παιδιά με τους γονείς τους εκθέτοντας έτσι ακόμη περισσότερες πληροφορίες για τα παιδιά. Όπως ανακοίνωσε η εταιρεία τα δεδομένα δεν δημοσιοποιήθηκαν, ούτε πωλήθηκαν σε τρίτους [40].

- MySQL:
  Σύμφωνα με την Acunetix, έγινε δημοσίευση στην Full Disclosure mailing list ότι στις 27 Μαρτίου του 2011 η επίσημη ιστοσελίδα της MySQL έπεσε θύμα ενός hacker (TinKode) χρησιμοποιώντας την τεχνική blind-based SQLI. Ο ίδιος ισχυρίστηκε ότι είχε βρεί και άλλη ευπάθεια μέσω XSS στην ίδια ιστοσελίδα. Δεν αποκάλυψε ακριβώς ποιά ήταν η ευπάθεια που είχε βρεί αλλά έδωσε αρκετά στοιχεία έτσι ώστε να πιστοποιηθεί το χακάρισμα. Μάλιστα έδωσαν την διεύθυνση URL απο όπου ξεκίνησε η επίθεση και ήταν η εξής: https://www.mysql.com/customers/view/index.html?id=1170 . Όπως βλέπουμε απο το URL ο εισβολέας παίρνει αρκετές πληροφορίες για την βάση δεδομένων όπως τον πίνακα customers και το id του πελάτη. Ακόμη άν αλλαχτεί χειροκίνητα το id και βάλουμε άλλον αριθμό θα εμφανιστεί άλλος πελάτης εκθέτοντας έτσι τους πελάτες της εταιρείας. Τέλος, εμφάνισαν και κάποιες άλλες ενδιαφέρουσες πληροφορίες που υπέκλεψαν απο το site, όπως την IP διεύθυνση του server, την έκδοση της PHP, το λειτουργικό σύστημα, τον τύπο επίθεσης που έκαναν και το όνομα και έκδοση του web server [36, 37].

```
Host IP              :   213.136.52.29
Web Server           :   Apache/2.2.15 (Fedora)
Powered-by           :   PHP/5.2.13
Injection Type       :   MySQL Blind
```

Εικόνα 4.3.i [36]

- Freepik & Falticon:
  Οι εταιρείες γραφικών Freepik και Falticon έπεσαν θύμα επίθεσης SQLI τον αύγουστο του 2020. Πιο συγκεκριμένα, δεδομένα 8,3 εκατ. χρηστών είχαν βρεθεί σε άμεσο κίνδυνο. Τα δεδομένα αυτά περιείχαν email και κωδικούς πρόσβασης των πελατών της εταιρείας. Οι πιο πολλοί απο αυτούς που πραγματικά επηρεάστηκαν απο αυτή την επίθεση ήταν επειδή είχαν πολύ αδύναμους κωδικούς ή είχαν συνδεθεί μέσω άλλης πλατφόρμας (Gmail, Facebook κτλπ) με αποτέλεσμα ο κωδικός τους να μην κρυπτογραφηθεί. Η εταιρεία απο την μεριά της ακύρωσε τους κωδικούς πρόσβασης που είχαν παραβιαστεί και έστειλε email στους κατόχους για την κατάσταση που προέκυψε. Τους προέτρεψε να αλλάξουν τους κωδικούς τους καθώς και κάθε άλλο κωδικό που ήταν ίδιος με αυτόν που είχε εκτεθεί και τον χρησιμοποιούν σε άλλες ιστοσελίδες. Μετά απο όλο αυτό τον πανικό, η εταιρεία αποφάσισε να δώσει μεγαλύτερη σημασία στην ασφάλεια των δεδομένων της και έτσι κάνει τακτικά ελέγχους για την πρόληψη και αντιμετώπιση νέων επιθέσεων [38, 39].

Αυτές και πολλές άλλες ιστοσελίδες έχουν πέσει θύμα hacking με SQLI απο τότε που πρωτοεμφανίστηκε. Ακόμη και σήμερα υπάρχουν χιλιάδες ιστοσελίδες ευάλωτες σε μία τέτοια επίθεση [31].

ΚΕΦΑΛΑΙΟ 5. Ανάπτυξη εργαλείου για την αυτόματη ανίχνευση του SQL Injection

5.1 Εισαγωγή

Όπως αναφέρθηκε και στο προηγούμενο κεφάλαιο, οι υπεράριθμες ιστοσελίδες που είναι ακόμα ευάλωτες σε επιθέσεις τύπου SQLI έχουν καταστήσει την ύπαρξη ενός εργαλείου που ανιχνεύει τέτοιου είδους ευπάθειες πολύ σημαντική. Τα εργαλεία αυτά και γνωστότερο όλων, το SQLMap, που έχει αναφερθεί και σε προηγούμενη ενότητα στοχεύουν στην ανίχνευση ενός ή πολλών διαφορετικών τρόπων έγχυσης εντολών στην βάση δεδομένων με σκοπό πάντα να εξαληφθούν όλα τα κενά ασφαλείας που μπορεί να έχουν δημιουργηθεί, για την ομαλότερη και ασφαλέστερη λειτουργία μιας ιστοσελίδας. Παρακάτω ακολουθεί η δική μου προσέγγιση στην ανίχνευση των ευπαθειών SQLI, ένα εργαλείο γραμμένο σε προγραμματιστική γλώσσα Python και μεγαλύτερη ειδίκευση στην ανίχνευση του error-based SQLI.

Οι προοδιαγραφές του εργαλείου για την αποτελεσματικότερη λειτουργία του έχουν ως εξής: Η ιστοσελίδα που στοχεύουμε να είναι διαθέσιμη στον κυβερνοχώρο και να είναι συνδεδεμένη με μία βάση δεδομένων. Οπότε εφόσον έχει μία βάση δεδομένων συνδεδεμένη θα υπάρχουν φόρμες συμπλήρωσης κειμένου για κάποια αναζήτηση ή κάποια διαστάυρωση στοιχείων για είσοδο χρήστη κτλπ. Το εργαλείο θα εντοπίσει τις φόρμες αυτές και θα ελέγξει σε σχέση με τα δεδομένα που εμείς θα στείλουμε απο ενα text αρχείο, αν είναι ευάλωτη η βάση δεδομένων με βάση την ανταπόκριση της βάσης αυτής, δηλαδή αν επιστρέψει κάποιο σφάλμα ή αν λειτουργεί κανονικά. Ειδικεύεται δηλαδή στα error-based SQLI.

5.2 Επεξήγηση του κώδικα

Ο κώδικας βρίσκεται στο παράρτημα a (Κώδικας του εργαλείου ανάπτυξης)

Για την ομαλότερη και ευκολότερη λειτουργία του εργαλείου ανίχνευσης έχουν χρησιμοποιηθεί οι ακόλουθες εξωτερικές βιβλιοθήκες: requests, BeautifulSoup4 (για λόγους συντομίας απο εδώ και στο εξής bs4), urllib.parse και η pprint. Βιβλιοθήκες οι οποίες αυτοματοποιούν κάποιες λειτουργίες της ανίχνευσης και θα αναλυθούν ξεχωριστά παρακάτω.

Στο ξεκίνημα αρχικοποιούμε μία συνεδρία αιτημάτων και ορίζουμε έναν χρήστη χρησιμοποιώντας την βιβλιοθήκη requests που κάνει όλες τις απαραίτητες διεργασίες για να το επιτύχει αυτό. Στην συνέχεια ακολουθούν 4 λειτουργίες (functions) και στο τέλος η main.

Η πρώτη λειτουργία είναι η get_all_forms με εισαγόμενη μεταβλητή το url, η οποία όπως προδίδει και το όνομα της είναι να επιστρέψει στον χρήστη όλες τις διαθέσιμες φόρμες που υπάρχουν στο url που έχει δοθεί. Αυτό το επιτυγχάνει με την βοήθεια της βιβλιοθήκης bs4 και σε συνδυασμό με

την βιβλιοθήκη requests όπου σώνουν το περιεχόμενο της ιστοσελίδας δηλαδη τον κώδικα html, σε μία μεταβλητή soup και στην συνέχεια η bs4 σαρώνει όλο αυτό το κώδικα για να επιστρέψει στον χρήστη όλες τις φόρμες που υπάρχουν.

Η δεύτερη λειτουργία είναι η get_form_details με εισαγόμενη μεταβλητή την form, όπου εξάγει κάποιες πληροφορίες για τις φόρμες που έχουν βρεθεί στην ιστοσελίδα. Πληροφορίες όπως, η ενέργεια της φόρμας (αυτή καθορίζει το αρχείο ή την σελίδα στην οποία υποβάλλεται η φόρμα), η μέθοδος (POST, GET κτλπ) καθώς και λεπτομέρειες για τις φόρμες εισαγωγής δεδομένων (τύπος, όνομα και τιμή). Επειδή οι λεπτομέρειες είναι 3 διαφορετικά δεδομένα, πρώτα αποθηκεύονται σε μία λίστα inputs και στην συνέχεια αποθηκεύονται σαν λίστα στο λεξικό details μαζί με την ενέργεια και την μέθοδο της φόρμας. Επιστρέφει στον χρήστη το λεξικό details οπου και θα χρησιμοποιηθεί παρακάτω.

Η τρίτη λειτουργία είναι η is_vulnerable με εισαγόμενη μεταβλητή το res (όπου res είναι το response της ιστοσελίδας) και ελέγχει αν είναι ευάλωτη σε SQLI μία ιστοσελίδα. Έχει δημιουργηθεί ένα λεξικό ονόματι errors όπου μέσα σε αυτό υπάρχουν αποσπάσματα σφαλμάτων που στέλνουν οι διάφορες βάσεις δεδομένων όταν κάτι έχει πάει στραβά ή μια εντολή προς την βάση δεν της είναι κατανοητή. Τα σφάλματα είναι απο βάσεις δεδομένων όπως MySQL, SQL Server, Oracle, PostgreSQL και Microsoft Access και βεβαίως μπορούν να προστεθούν κιάλλα σφάλματα από άλλες βάσεις δεδομένων αλλά και απο τις ήδη υπάρχουσες. Επιστρέφει True στον χρήστη άν αντιστοιχίσει κάποιο ενδεχόμενο σφάλμα που μπορεί να στείλει η ιστοσελίδα με αυτά που υπάρχουν μέσα στο λεξικό errors, αλλιώς αν δεν το αντιστοιχίσει επιστρέφει False.

Η τέταρτη λειτουργία είναι η scan_sql_injection με εισαγόμενη μεταβλητή το url, η οποία χρησιμοποιεί όλα τα παραπάνω εργαλεία για να διαπιστώσει εν τέλει αν μία ιστοσελίδα είναι ευάλωτη σε SQLI και ειδικότερα σε error-based. Με την βοήθεια του εργαλείου θα πραγματοποιηθούν 2 τύποι έγχυσης, μία μέσω του url της ιστοσελίδας και μία μέσω της φόρμας της ιστοσελίδας. Γιαυτό το σκοπό και για μεγαλύτερη ευκολία έχουν δημιουργηθεί 2 άλλα αρχεία text (url_lines.txt και html_submit.txt) όπου περιέχουν διάφορα σύμβολα ή εντολές που θα προστεθούν στο url και στην φόρμα με σκοπό να διαπιστωθεί άν είναι ευάλωτη η ιστοσελίδα σε SQLI. Κάποιο απο το περιεχόμενο των text αρχείων είναι το εξής: ‘ , “ , # , -- , # OR 1=1 κτλπ. Αρχικά διαβάζουμε το αρχείο url_lines.txt και μέσω μιας επανάληψης προσθέτουμε το περιεχόμενο του αρχείου στο τέλος του url και παίρνουμε το response απο το νεο url μέσω πάντα της βιβλιοθήκης requests. Εξάγουμε τις φόρμες άν υπάρχουν στα νεα url που θα δοκιμαστούν και εκτυπώνουμε τις λεπτομέρειές τους. Άν υπάρχει σφάλμα ελέγχεται μέσω της is_vulnerable και αν βρεθεί αυτό το σφάλμα στο λεξικό τότε εκτυπώνει ότι η ιστοσελίδα είναι ευάλωτη σε SQLI και περαιτέρω πληροφορίες. Ακόμη εκτυπώνονται και οι κωδικοί html του νέου url (κάποιοι απο αυτούς είναι το 200 δηλαδή ότι το αίτημα είναι επιτυχές, το 404 ότι δεν βρέθηκε ιστοσελίδα κτλπ). Αυτή η εκτύπωση δεν μας βοηθάει άμεσα να καταλάβουμε κάτι για την ευπάθεια SQLI αλλά μας βοηθάει να ξέρουμε άν το νέο url είναι όντως υπαρκτό ή οχι. Στην δεύτερη φάση διαβάζουμε το αρχείο html_submit.txt και επαναλαμβάνεται η προηγούμενη διαδικάσια με την διαφορά ότι τώρα το περιεχόμενο του αρχείου html_submit.txt θα τοποθετηθεί στην φόρμα που έχει εξαχθεί απο την ιστοσελίδα, και κατα συνέπεια θα πάρουμε το response της ιστοσελίδας και θα το ελέγξουμε μέσω

της is_vulnerable για να βγεί το πόρισμα αν είναι ευάλωτη σε SQLI, λαμβάνοντας υπόψιν πάντα τα errors που στέλνει η βάση δεδομένων.

Τέλος στην main υπάρχει δυνατότητα να γράφει ο χρήστης ποιά ιστοσελίδα θέλει να κάνει ανίχνευση γράφοντας την χειροκίνητα μέσω του terminal κάθε φορά που θέλει να εκκινήσει την εφαρμογή με την βοήθεια της βιβλιοθήκης sys και της εντολής url = sys.argv[1] ή να ορίσει μία μεταβλητή url μέσα στην main και να αλλάζει την ιστοσελίδα απο εκεί. Έπειτε καλείται η λειτουργία scan_sql_injection με το υπάρχων url και εκτελείται όλη η παραπάνω διαδικασία.

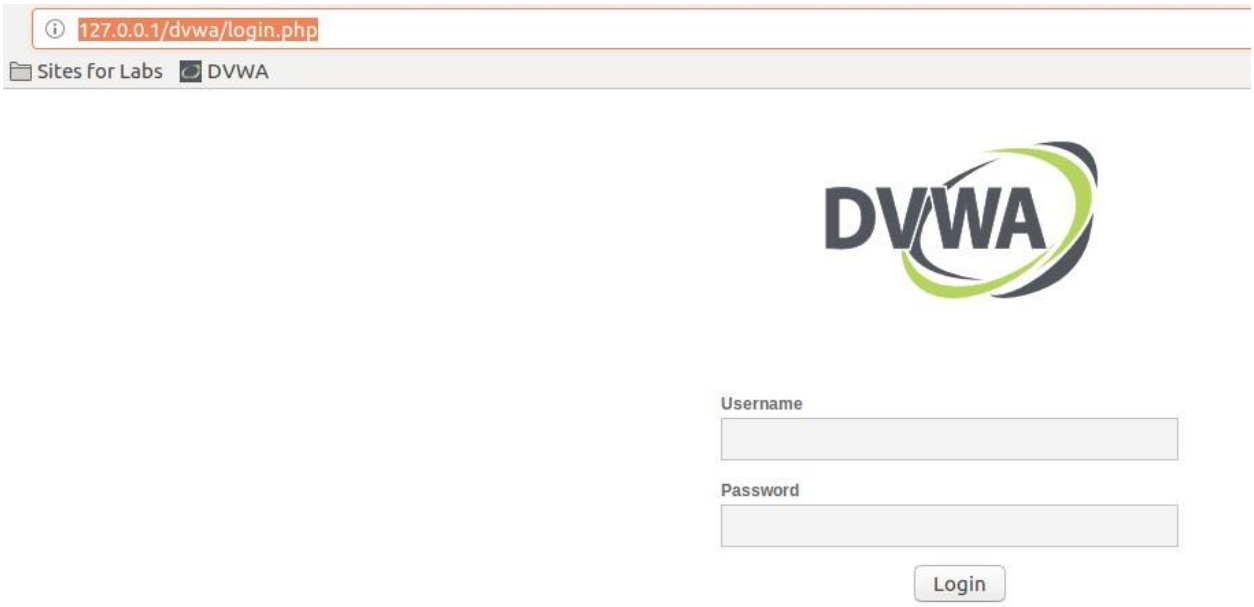
5.3 Εκτέλεση του εργαλείου ανίχνευσης σε ιστοσελίδες

Οι ενέργειες στις παρακάτω περιπτώσεις έγιναν με μόνο σκοπό την εξοικείωση και την ανάδειξη των δυνατοτήτων του εργαλείου ανίχνευσης κατά της επίθεσης SQL Injection. Οι 2 απο τις 3 ιστοσελίδες που θα παρουσιαστούν στην συνέχεια είναι ενεργές τοπικά μόνο και για λόγους εκπαιδευτικούς.

5.3.1 Περίπτωση 1η

Η πρώτη ιστοσελίδα με την οποία θα ασχοληθούμε είναι η DVWA (aka Damn Vulnerable Web Application) η οποία είναι γραμμένη σε PHP και χρησιμοποιεί MySQL για την βάση δεδομένων της. Αφού έχει στηθεί κατάλληλα η ιστοσελίδα, έχει δημιουργηθεί μία βάση δεδομένων και έχουν γίνει οι απαραίτητες ρυθμίσεις είναι έτοιμη να λειτουργήσει τοπικά σε περιβάλλον Ubuntu.

Το url είναι 127.0.0.1/dvwa/login.php και μας εμφανίζει να συμπληρώσουμε 2 πεδία, ένα για το username και ένα για το password:



Εικόνα 5.3.1.i (DVWA login form)

Για λόγους ευκολίας και συντομίας, το εκάστοτε url έχει τοποθετηθεί μέσα στον κώδικα του εργαλείου έτσι ώστε να χρειάζεται να τρέξουμε μόνο το πρόγραμμα χωρίς να πληκτρολογούμε το url στο terminal. Το αρχείο με τον κώδικα ονομάζεται test.py και παρακάτω εκτελείται:

```
Terminal
[04/14/21]seed@VM:~/.../test sqli$ python3 test.py
[!][URL INJ] Trying http://127.0.0.1/dvwa/login.php'

404 - Client Errors

[!][URL INJ] Trying http://127.0.0.1/dvwa/login.php"

404 - Client Errors

[!][URL INJ] Trying http://127.0.0.1/dvwa/login.php#

[+] Detected 1 forms on http://127.0.0.1/dvwa/login.php#
.
[+] Form:
{'action': 'login.php',
 'inputs': [{'name': 'username', 'type': 'text', 'value': ''},
            {'name': 'password', 'type': 'password', 'value': ''},
            {'name': 'Login', 'type': 'submit', 'value': 'Login'},
            {'name': 'user_token',
             'type': 'hidden',
             'value': '07bea14c1cea48f080303a1488e675eb'}],
 'method': 'post'}
```

Εικόνα 5.3.1.ii

```
[!][URL INJ] Trying http://127.0.0.1/dvwa/login.php?

[+] Detected 1 forms on http://127.0.0.1/dvwa/login.php?
.
[+] Form:
{'action': 'login.php',
 'inputs': [{'name': 'username', 'type': 'text', 'value': ''},
            {'name': 'password', 'type': 'password', 'value': ''},
            {'name': 'Login', 'type': 'submit', 'value': 'Login'},
            {'name': 'user_token',
             'type': 'hidden',
             'value': 'c221e218b85be3acd54eb3846257f6e5'}],
 'method': 'post'}
```

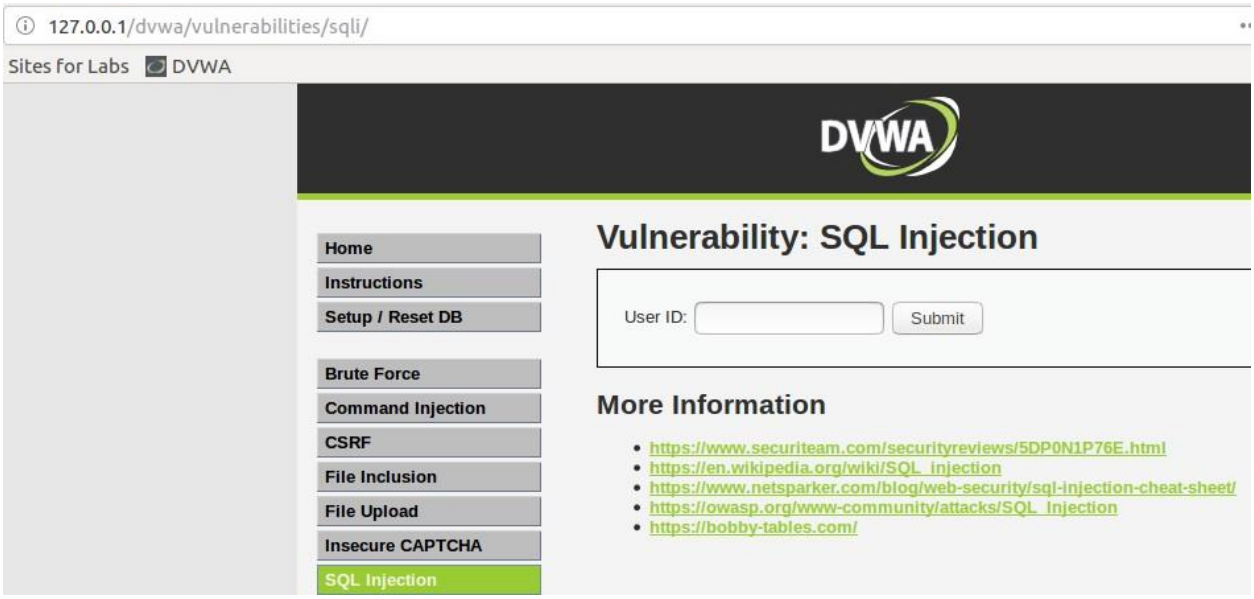
Εικόνα 5.3.1.iii

```
[!][URL INJ] Trying http://127.0.0.1/dvwa/login.php--

404 - Client Errors

[+] Detected 1 form(s) on the URL given : http://127.0.0.1/dvwa/login.php
[+] Form:
{'action': 'login.php',
 'inputs': [{'name': 'username', 'type': 'text', 'value': ''},
            {'name': 'password', 'type': 'password', 'value': ''},
            {'name': 'Login', 'type': 'submit', 'value': 'Login'},
            {'name': 'user_token',
             'type': 'hidden',
             'value': 'b32eb2eed63758b9662415200432cbce'}],
 'method': 'post'}
[04/14/21]seed@VM:~/.../test sqli$
```

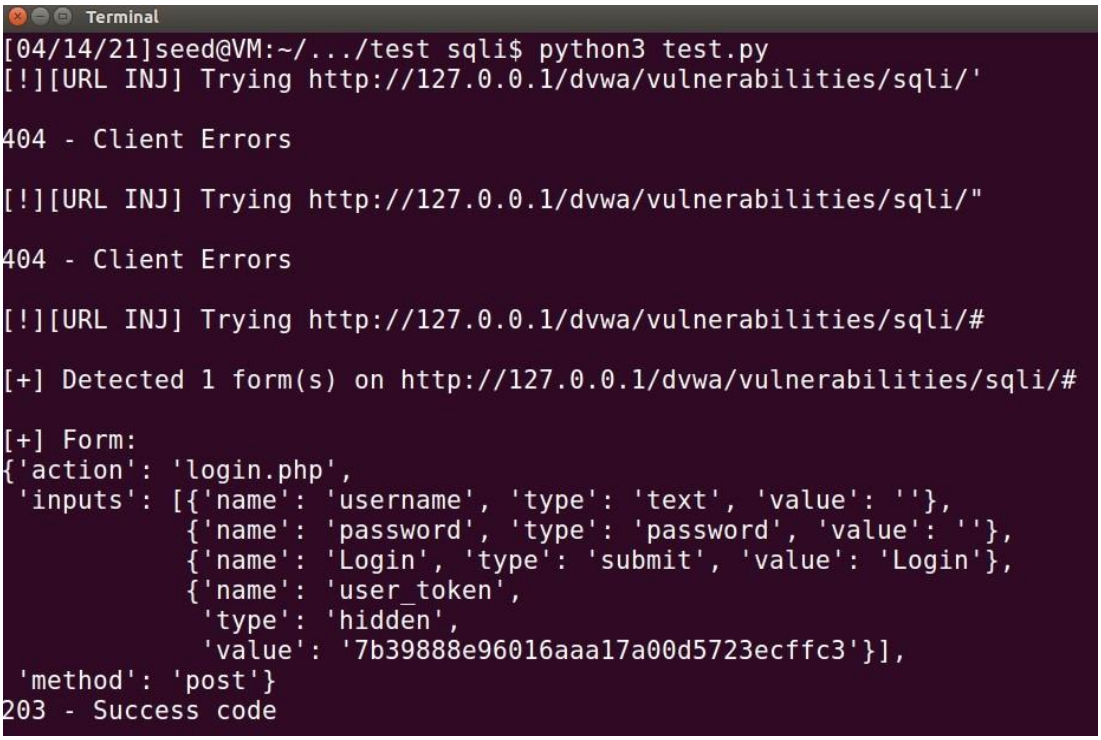
Εικόνα 5.3.1.iv

Αυτό που συμβαίνει περιληπτικά στην προκειμένη περίπτωση, όπως αναφέρθηκε και προηγουμένως στην ανάλυση του κώδικα, είναι ότι επιχειρεί να δημιουργήσει νέα url με βάση το text αρχείο που έχουμε φορτώσει στον κώδικα, να εκτυπώσει αν υπάρχουν νέες φόρμες συμπλήρωσης στα νέα url, να εκτυπώσει αναλυτικές πληροφορίες για την φόρμα του εκάστοτε url, να εκτυπώσει τα σφάλματα που προκύπτουν απο την ενέργεια αυτή και εν τέλει με βαση τα σφάλματα αυτά να αποφανθεί αν είναι ευάλωτη ή όχι. Το αποτέλεσμα ήταν ότι δεν βρήκε κάτι ευάλωτο στην φόρμα αυτή αλλά μας επέστρεψε πολύτιμες πληροφορίες για την φόρμα συμπλήρωσης. Σε αυτή την περίπωση όπως και σε όλες τις άλλες που θα ακολουθήσουν έχουν γραφτεί ελάχιστα σύμβολα/ UNION εντολές/ OR εντολές μέσα στο αρχείο text καθώς μεγαλύτερη σημασία έχει να αναδειχθεί ο τρόπος λειτουργίας της εφαρμογής παρά να βρεθεί ένα κενό ασφαλείας στις εκάστοτε ιστοσελίδες. Προφανώς γράφοντας περισσότερα σύμβολα και εντολές αυξάνουμε τις πιθανότητες στο να βρεθεί κάποια ευπάθεια.

Αφού έγινε η ανίχνευση στην φόρμα εισόδου του χρήστη, τώρα θα ακολουθήσει και η ανίχνευση σε κάποιο σημείο της ιστοσελίδας που έχει μία φόρμα αναζήτησης:



Εικόνα 5.3.1.v



Εικόνα 5.3.1.vi

```
[!][URL INJ] Trying http://127.0.0.1/dvwa/vulnerabilities/sqli/?

[+] Detected 1 form(s) on http://127.0.0.1/dvwa/vulnerabilities/sqli/?

[+] Form:
{'action': 'login.php',
 'inputs': [{'name': 'username', 'type': 'text', 'value': ''},
            {'name': 'password', 'type': 'password', 'value': ''},
            {'name': 'Login', 'type': 'submit', 'value': 'Login'},
            {'name': 'user_token',
             'type': 'hidden',
             'value': '9fc7fe36d3ce5d0e07867c6c0455858f'}],
```

```
[!][URL INJ] Trying http://127.0.0.1/dvwa/vulnerabilities/sqli/--

404 - Client Errors

[+] Detected 1 form(s) on the URL given : http://127.0.0.1/dvwa/vulnerabil
ities/sqli/
[+] Form:
{'action': 'login.php',
 'inputs': [{'name': 'username', 'type': 'text', 'value': ''},
            {'name': 'password', 'type': 'password', 'value': ''},
            {'name': 'Login', 'type': 'submit', 'value': 'Login'},
            {'name': 'user_token',
             'type': 'hidden',
             'value': '3b278599bc151a5847268be669ce16d8'}],
 'method': 'post'}
[04/14/21]seed@VM:~/.../test sqli$
```

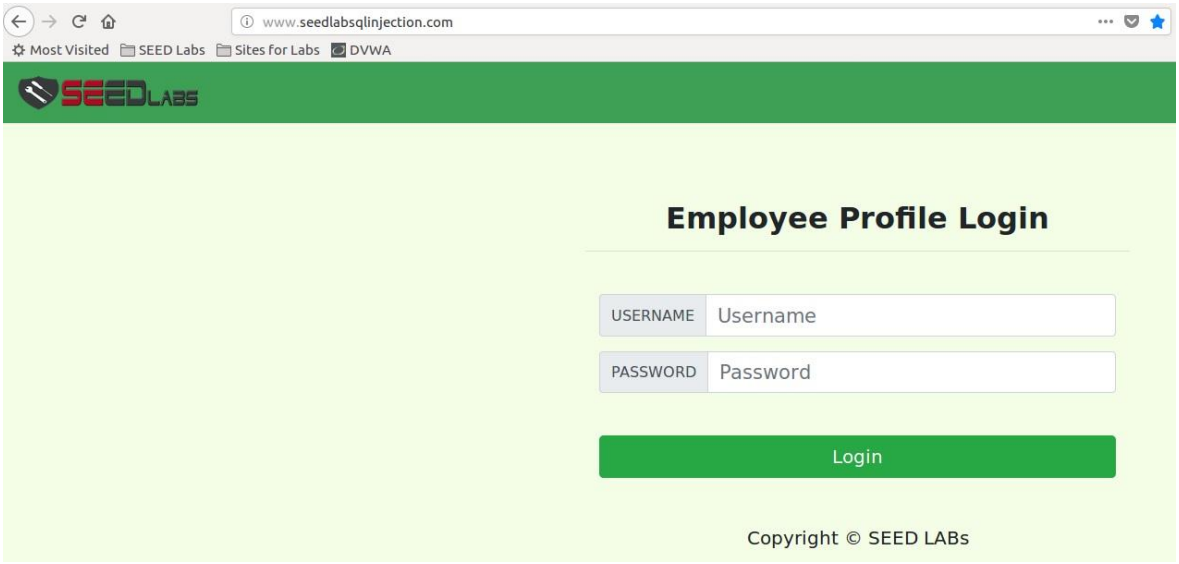
Εικόνα 5.3.1.vii & Εικόνα 5.3.1.vii

Στις παραπάνω εικόνες διαπιστώνουμε τις διαφορετικές προσπάθειες μέσω εναλλακτικών url συνοδευόμενες από τις προαναφέρουσες πληροφορίες και αφού εκτελέσει όλο το περιεχόμενο του text και δεν βρεί κάποια ευπάθεια εμφανίζει ξανά την φόρμα συμπλήρωσης για είσοδο των χρηστών.

5.3.2 Περίπτωση 2η

Για την 2η ιστοσελίδα που θα πειραματιστεί το εργαλείο ανίχνευσης είναι η Seedlabs, ένα open-source project το οποίο χρησιμοποιεί PHP και MySQL για βάση δεδομένων. Και αυτή η ιστοσελίδα είναι εγκατεστημένη και υπάρχει πρόσβαση μόνο τοπικά σε λειτουργικό Ubuntu. Στόχος του εργαλείου στην περίπτωση αυτή θα είναι το εξειδικευμένο για SQLI εργαστήριο του Seedlabs με url www.seedlabsinjection.com αφού έχει δημιουργηθεί κατάλληλα η πλατφόρμα και η βάση δεδομένων που χρησιμοποιεί η ιστοσελίδα αυτή.

Επαναλαμβάνουμε την ίδια διαδικασία και για αυτή την ιστοσελίδα και ακολουθούν τα αποτελέσματα:



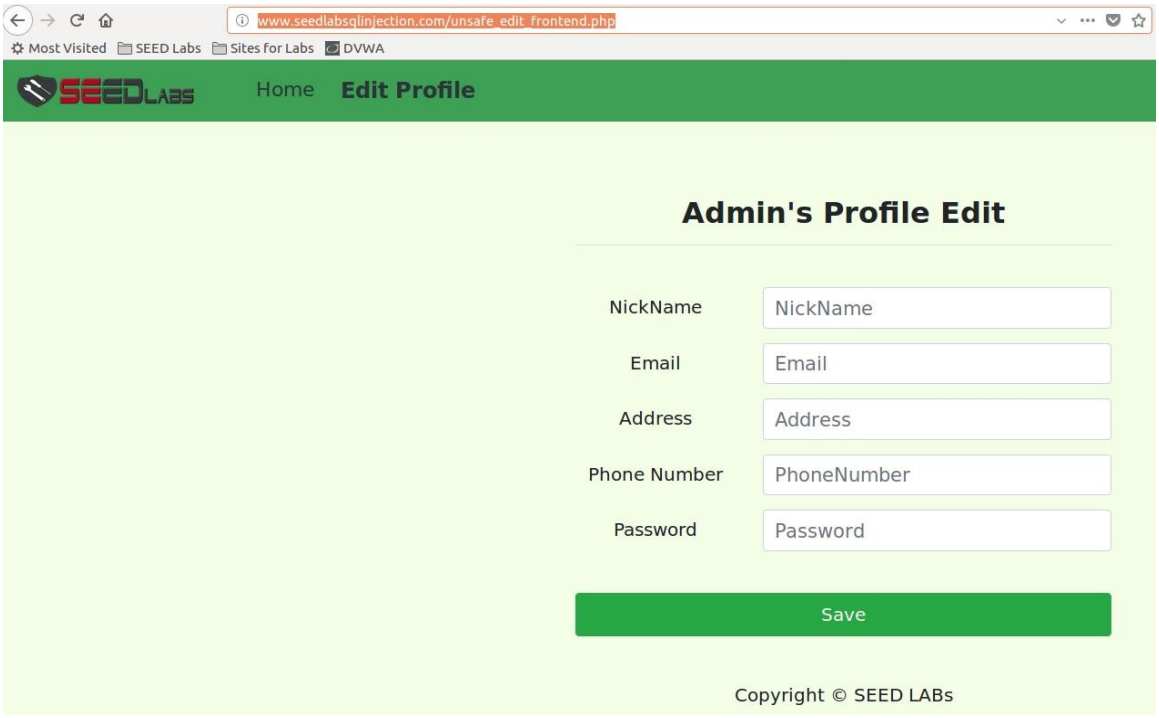

```
Terminal
[04/14/21]seed@VM:~/.../test sqli$ python3 test.py
[!][URL INJ] Trying http://www.seedlabsqlinjection.com/'

404 - Client Errors

[!][URL INJ] Trying http://www.seedlabsqlinjection.com/"

404 - Client Errors

[!][URL INJ] Trying http://www.seedlabsqlinjection.com/#

[+] Detected 1 forms on http://www.seedlabsqlinjection.com/#
.
[+] Form:
{'action': 'unsafe_home.php',
 'inputs': [{'name': 'username', 'type': 'text', 'value': ''},
            {'name': 'Password', 'type': 'password', 'value': ''}],
 'method': 'get'}
201 - Success code
200 - Success code

[!][URL INJ] Trying http://www.seedlabsqlinjection.com/?

[+] Detected 1 forms on http://www.seedlabsqlinjection.com/?
.
[+] Form:
{'action': 'unsafe_home.php',
 'inputs': [{'name': 'username', 'type': 'text', 'value': ''},
            {'name': 'Password', 'type': 'password', 'value': ''}],
 'method': 'get'}
201 - Success code
200 - Success code
```

```
[!][URL INJ] Trying http://www.seedlabsqlinjection.com/--

404 - Client Errors

[+] Detected 1 forms on http://www.seedlabsqlinjection.com/.
[+][FORM INJ] SQL Injection vulnerability detected, link: http://www.seedl
absqlinjection.com/unsafe_home.php
[+] Form:
{'action': 'unsafe_home.php',
 'inputs': [{'name': 'username', 'type': 'text', 'value': ''},
            {'name': 'Password', 'type': 'password', 'value': ''}],
 'method': 'get'}
[04/14/21]seed@VM:~/.../test sqli$
```

Εικόνα 5.3.2.i & Εικόνα 5.3.2.ii & Εικόνα 5.3.2.iii

Εδώ όπως φαίνεται και απο τα αποτελέσματα, στην αρχή εντοπίστηκε η φόρμα μέσω των διαφορετικών url αλλά στην συνεχεία διαπιστώθηκε ευπάθεια όταν έγιναν submit οι εντολές του αρχείου που είναι κατάλληλο για τις επιθέσεις μέσω των φορμών συμπλήρωσης στοιχείων και εκτυπώθηκαν περαιτέρω πληροφορίες για το action, τα inputs και το method.

Αφού γίνει σύνδεση και μπεί ο χρήστης στο προφίλ του για να αλλάξει ενδεχομένως κάποια προσωπικά στοιχεία οδηγείται σε αυτήν την ιστοσελίδα όπου θα γίνει ανίχνευση και το url της είναι www.seedlabsqlinjection.com/unsafe_edit_frontend.php (Εφόσον είναι τοπικά):



```
Terminal
[04/14/21]seed@VM:~/.../test sqli$ python3 test.py
[!][URL INJ] Trying http://www.seedlabsqlinjection.com/unsafe_edit_fronten
d.php'

404 - Client Errors

[!][URL INJ] Trying http://www.seedlabsqlinjection.com/unsafe_edit_fronten
d.php"

404 - Client Errors

[!][URL INJ] Trying http://www.seedlabsqlinjection.com/unsafe_edit_fronten
d.php#

[+] Detected 1 forms on http://www.seedlabsqlinjection.com/unsafe_edit_fro
ntend.php#
.
[+] Form:
{'action': 'unsafe_edit_backend.php',
 'inputs': [{'name': 'NickName', 'type': 'text', 'value': ''},
            {'name': 'Email', 'type': 'text', 'value': ''},
            {'name': 'Address', 'type': 'text', 'value': ''},
            {'name': 'PhoneNumber', 'type': 'text', 'value': ''},
            {'name': 'Password', 'type': 'password', 'value': ''}],
 'method': 'get'}
201 - Success code
200 - Success code
```

```
[!][URL INJ] Trying http://www.seedlabsqlinjection.com/unsafe_edit_fronten
d.php?

[+] Detected 1 forms on http://www.seedlabsqlinjection.com/unsafe_edit_fro
ntend.php?
.
[+] Form:
{'action': 'unsafe_edit_backend.php',
 'inputs': [{'name': 'NickName', 'type': 'text', 'value': ''},
            {'name': 'Email', 'type': 'text', 'value': ''},
            {'name': 'Address', 'type': 'text', 'value': ''},
            {'name': 'PhoneNumber', 'type': 'text', 'value': ''},
            {'name': 'Password', 'type': 'password', 'value': ''}],
 'method': 'get'}
201 - Success code
200 - Success code

[!][URL INJ] Trying http://www.seedlabsqlinjection.com/unsafe_edit_fronten
d.php--

404 - Client Errors

[+] Detected 1 forms on http://www.seedlabsqlinjection.com/unsafe_edit_fro
ntend.php.
[04/14/21]seed@VM:~/.../test sqli$
```

Εικόνα 5.3.2.iv & Εικόνα 5.3.2.v & Εικόνα 5.3.2.vi

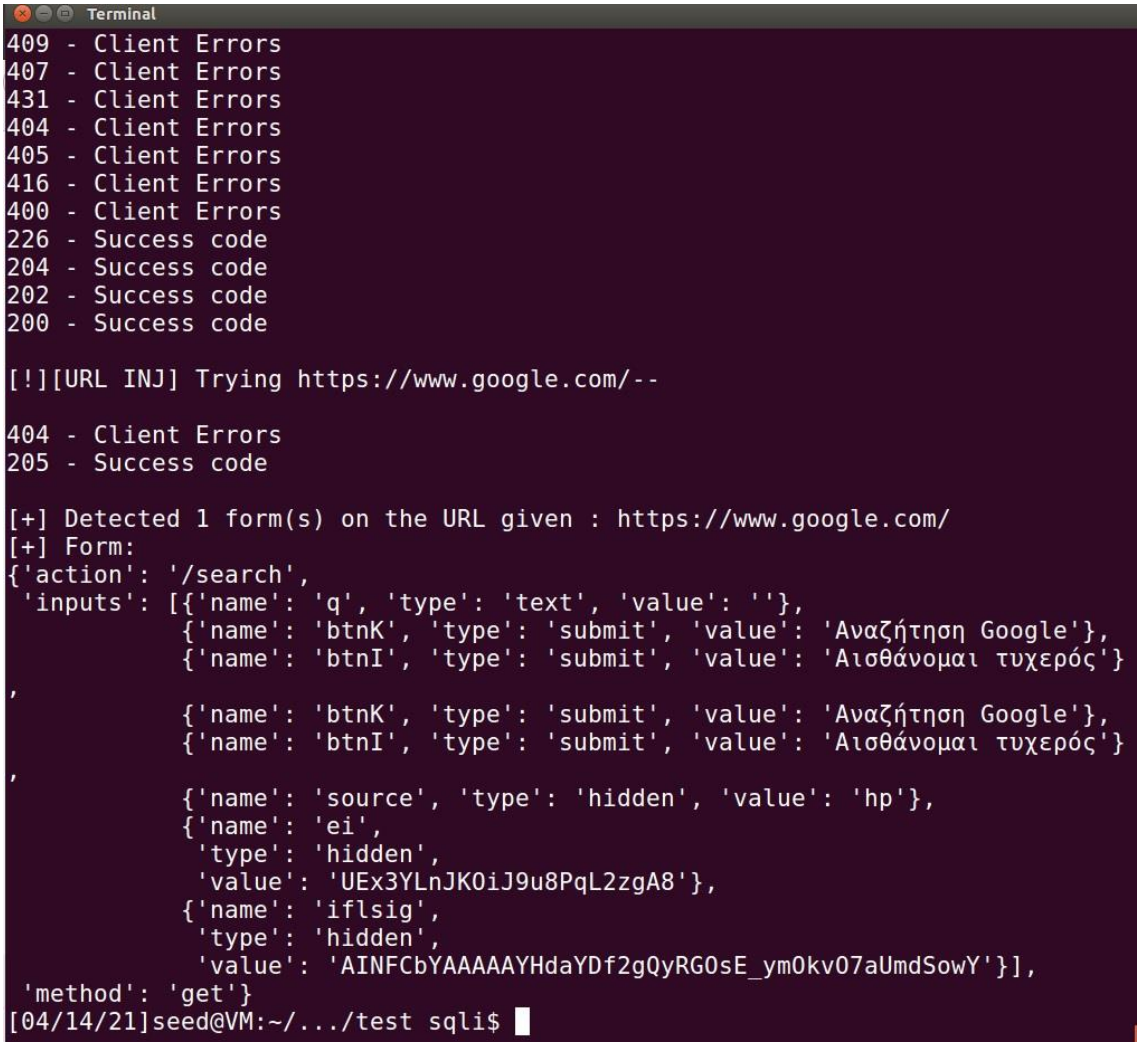
Παρατηρούμε ότι ανιχνεύει την φόρμα συμπλήρωσης στοιχείων με τα διάφορα πεδία όπως Email, Address κτλπ αλλά δεν επιβεβαιώνει την ευπάθεια απο επιθέσεις SQLI καθώς οι εντολές που χρησιμοποιήθηκαν μέσα στο text αρχείο δεν ήταν κατάλληλες για το συγκεκριμένο ιστότοπο ώστε να το επιβεβαιώσουν ως ευπαθές. Γεμίζοντας το αρχείο με περισσότερες ειδικές εντολές για SQLI είναι πολύ πιθανό να ανιχνεύσει και εδώ ευπάθεια.

5.3.3 Περίπτωση 3η

Στην 3η και τελευταία περίπτωση θα γίνει ανίχνευση σε μία ευρέως γνωστή και πολυχρησιμοποιημένη ιστοσελίδα την www.google.com :

```
Terminal
[04/14/21]seed@VM:~/.../test sqli$ python3 test.py
[!][URL INJ] Trying https://www.google.com/'

404 - Client Errors
205 - Success code

[!][URL INJ] Trying https://www.google.com/"

404 - Client Errors
205 - Success code

[!][URL INJ] Trying https://www.google.com/#

[+] Detected 1 form(s) on https://www.google.com/#

[+] Form:
{'action': '/search',
 'inputs': [{'name': 'q', 'type': 'text', 'value': ''},
            {'name': 'btnK', 'type': 'submit', 'value': 'Αναζήτηση Google'},
            {'name': 'btnI', 'type': 'submit', 'value': 'Αισθάνομαι τυχερός'}
,
            {'name': 'btnK', 'type': 'submit', 'value': 'Αναζήτηση Google'},
            {'name': 'btnI', 'type': 'submit', 'value': 'Αισθάνομαι τυχερός'}
,
            {'name': 'source', 'type': 'hidden', 'value': 'hp'},
            {'name': 'ei',
             'type': 'hidden',
             'value': 'T0x3YPy7HvCD9u8Pv8Gm6A0'},
            {'name': 'iflsig',
             'type': 'hidden',
             'value': 'AINFCbYAAAAAYHdaXxxfj8WErqXABQ7PfJKXKmEzkdly'}],
 'method': 'get'}
408 - Client Errors
451 - Client Errors
428 - Client Errors
406 - Client Errors
```

Εικόνα 5.3.3.i



Εικόνα 5.3.3.ii

Στην προκειμένη περίπτωση προφανώς και δεν θα βρισκόταν κάποια ευπάθεια με ένα τόσο απλό εργαλείο για μία τόσο μεγάλη εταιρεία, αλλά οι πληροφορίες σχετικά με την φόρμα αναζήτησης είναι αρκετά ενδιαφέρουσες όπως επίσης και τα διάφορα σφάλματα ή επιτυχή συμβάντα που επιστρέφει στον χρήστη μία αναζήτηση.

ΣΥΜΠΕΡΑΣΜΑΤΑ

Υπάρχει πληθώρα μεθόδων να αποκτήσει παράνομη πρόσβαση κάποιος χρήστης σε μία βάση δεδομένων και ιδιαίτερα με την επίθεση SQL Injection. Ενδεχομένως πολλές εταιρείες να μην γνωρίζουν αρκετά για την απειλή αυτή και να πέφτουν θύματα ενός χάκερ ακόμη πιο εύκολα.

Σε αυτή την διπλωματική όπου παρουσιάζονται και αναλύονται αρκετά πράγματα σχετικά με τις βάσεις δεδομένων και τους τρόπους εισχώρησης ενός εισβολέα, μπορεί να φανεί χρήσιμη για την καλύτερη κατανόηση του θέματος. Ειδικότερα αναλύονται οι τρόποι με τους οποίους μπορεί να θωρακιστεί μία βάση δεδομένων όπως για παράδειγμα να χρησιμοποιείται παραπάνω απο μία μεθόδους προστασίας (πχ. white lists σε συνδυασμό με παραμετροποιημένα ερωτήματα) για την καλύτερη εξασφάλιση της βάσης. Απο την άλλη πλευρά αναφέρονται διάφοροι τρόποι για να ανιχνευθούν τέτοιου είδους επιθέσεις αλλά και πραγματικά παραδείγματα επιθέσων.

Η SQL Injection που τα τελευταία χρόνια έχει αρκετά ανοδική τάση και αντί να περιορίζονται τα σφάλματα και οι τρόποι με τους οποίους μπορεί να εισέλθει κάποιος παράνομα σε μία βάση αυξάνονται και ο λόγος είναι επείδη υπάρχει τόση πολλή πληροφορία στον διαδικτυακό χώρο και ειδικότερα στις βάσεις δεδομένων που το καθιστά αδύνατο να υπάρχουν βάσεις δεδομένων που να μην είναι ευάλωτες σε τέτοιες επιθέσεις καθώς υπάρχουν πολλαπλοί τρόποι έγχυσης κώδικα. Δεν θα πρέπει σε καμία περίπτωση αυτή η επίθεση να θεωρείται ‘άκακη’ και να μην τηρούνται πολλαπλά επίπεδα ασφαλείας, τόσο απο τους διαχειριστές των βάσεων δεδομένων όσο και απο τους developers των ιστοσελίδων.

Ακόμη έχουν αναφερθεί πολλά λάθη απο μεριάς προγραμματιστών που καλό θα ήταν να αποφεύγονται. Η σωστή ανάπτυξη μιας ιστοσελίδας και η ασφαλής σύνδεση της με μια βάση δεδομένων σύμφωνα με τα πρότυπα που έχουν αναφερθεί στην διπλωματική θα εκμηδενίσουν τις επιτυχείς εισβολές τρίτων. Ακόμη, η συντήρηση των βάσεων δεδομένων και η σωστή ενημέρωση των χρηστών της είναι επίσης καλές πρακτικές για την ορθότερη λειτουργία της.

Μία άλλη πρακτική που καλό είναι να εφαρμόζεται είναι η χρησιμοποίηση έτοιμων εργαλείων που αποσκοπούν στην ανίχνευση των ευπαθειών μιας ιστοσελίδας ή μιας βάσης δεδομένων όπως και το εργαλείο που αναπτύχθηκε σε αυτή την διπλωματική. Οι διαφορές του με τα ήδη υπάρχοντα εργαλεία είναι ότι πρόκειται για ένα εργαλείο πολύ απλό στην χρήση του, αρκετά ‘ελαφρύ’ με βάση τους πόρους που χρειάζεται για να λειτουργήσει και πολύ αποτελεσματικό στον τομέα ανίχνευσης που ειδικεύεται δηλαδή στο error-based SQLI. Ακόμη είναι ευέλικτο καθώς μπορεί να προσαρμοστεί και να αποτρέψει και άλλες κατηγορίες επιθέσεων SQLI αφού οι εντολές και τα σύμβολα που ελέγχονται, αντλούνται από εξωτερικά αρχεία text. Είναι και αυτό ένα εργαλείο που συνδυαστικά με κάποιο άλλο εργαλείο ή με κάποιες άλλες μεθόδους προστασίας που έχουν προαναφερθεί, να φέρει σε πέρας ένα εξαιρετικό αποτέλεσμα και να προστατέψει μία βάση δεδομένων επιτυχώς.

Τέλος, η επίθεση SQLI όπως διαπιστώθηκε και παραπάνω μπορεί να ανιχνευτεί απο αυτούς τους μηχανισμούς αλλά δεν αρκεί μόνο αυτό. Όπως υπάρχουν πολλαπλοί τρόποι έγχυσης κώδικα έτσι θα πρέπει να υπάρχουν και πολλαπλοί τρόποι ανίχνευσης και προστασίας. Ένα εργαλείο μπορεί να ειδικεύεται σε ένα μόνο είδος επίθεσης οπότε όλοι οι άλλοι τρόποι μένουν ακάλυπτοι. Ακόμη

και αν χρησιμοποιηθούν 2 και 3 διαφορετικά εργαλεία ανίχνευσης πάντα θα πρέπει να συνδυάζονται με τις διάφορες μεθόδους αποτροπής έγχυσης κώδικα τόσο απο την μεριά του προγραμματιστή της ιστοσελίδας όσο και απο την μεριά του προγραμματιστή της βάσης δεδομένων για την μέγιστη ασφάλεια και των δύο. Μία επίθεση μπορεί να ανιχνευτεί και εύκολα και δύσκολα, εξαρτάται πάντα απο το πόσο πολύπλοκο είναι το εργαλείο δηλαδή πόσες μεθόδους χρησιμοποιεί για να εισχωρήσει σε μία βάση και βέβαια εξαρτάται και απο το πόσο θωρακισμένη είναι μία βάση.

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] Tucci, L 2014, Information Age, TechTarget, retrieved from https://searchcio.techtarget.com/definition/Information-Age

[2] Biscobing, J 2020, Relational Database, TechTarget, retrieved from https://searchdatamanagement.techtarget.com/definition/relational-database

[3] Becker, R 2021, Structured Query Language (SQL), Techopedia, retrieved from https://www.techopedia.com/definition/1245/structured-query-language-sql

[4] Cisco Systems, Inc., Types of Cyber Attacks, Cisco, accessed February 2021 https://www.cisco.com/c/en/us/products/security/common-cyberattacks.html

[5] Klimek, J, Relational Algebra, Charles University Prague, retrieved from https://www.yumpu.com/en/document/read/6262313/a-

[6] Chaitanya, S, DBMS Relational Calculus, BeginnersBook, accessed January 2021 https://beginnersbook.com/2019/02/dbms-relational-calculus/

[7] Gruhn, N 2020, Turing Complete, DEV, retrieved from https://dev.to/gruhn/what-makes-a-programming-language-turing-complete-58fl

[8] Marr, B 2015, Crunching Big Data, TechTarget, retrieved from https://www.datasciencecentral.com/profiles/blogs/ten-top-languages-for-crunching-big-data

[9] Stanford Encyclopedia of Philosophy, 2018, Turing Machines retrieved from https://plato.stanford.edu/entries/turing-machine/

[10] JavaTpoint, Relational Algebra, accessed December 2020 https://www.javatpoint.com/dbms-relational-algebra

[11] JavaTpoint, Relational Calculus, accessed December 2020 https://www.javatpoint.com/dbms-relational-calculus

[12] McGraw-Hill & Atzeni, 1999, Database Systems, Ceri, Paraboschi, Torlone https://www.yumpu.com/en/document/read/50726481/relational-algebra-and-calculus

[13] W3Schools, SQL Tutorial, accessed January 2021 https://www.w3schools.com/sqL/default.asp

[14] Code Academy, SQL Commands, CodeAcademy, accessed November 2020 https://www.codecademy.com/articles/sql-commands

[15] Carnes, B 2020, Basic SQL Commands, FreeCodeCamp, retrieved from https://www.freecodecamp.org/news/basic-sql-commands/

[16] Brirznieks, G 2021, How does a Web Server Work?, ServerWatch, retrieved from https://www.serverwatch.com/how-web-servers-work

[17] The Economic Times, Definition of 'Web Server', accessed January 2021 https://economictimes.indiatimes.com/definition/web-server

[18] Melnick, J 2018, Most common types of cyber attacks, Netwrix, retrieved from https://blog.netwrix.com/2018/05/15/top-10-most-common-types-of-cyber-attacks/

[19] Cisco Systems, Inc., How Cyber Attacks Work, Cisco, accessed February 2021 https://www.cisco.com/c/en/us/products/security/common-cyberattacks.html

[20] ItGovernance, 2019, Most Common Cyber Attacks, ItGovernance, retrieved from https://www.itgovernanceusa.com/blog/six-most-common-cyber-attacks

[21] Jefferson, B 2021, Most Common Types of Cyber Attacks, Lepide retrieved from https://www.lepide.com/blog/the-15-most-common-types-of-cyber-attacks/

[22] Acunetix, What is SQL Injection, Acunetix, accessed December 2020 https://www.acunetix.com/websitesecurity/sql-injection/?fbclid=IwAR0eFU_LlRfJSYBpotKmsZYUr_o3Nq7R8p7UTWQhk9mdPiXwN2iLZwo4BCU

[23] ItZone, 2020, What is SQL injection?, ItZone, retrieved from https://itzone.com.vn/en/article/what-is-sql-injection-how-many-types-of-sql-injection-attacks-are-there/

[24] Acunetix, Types of SQL Injection, Acunetix, accessed January 2021 https://www.acunetix.com/websitesecurity/sql-injection2/

[25] Zanni, A 2018, Types of SQL Injection, Rawsec, retrieved from https://rawsec.ml/en/types-of-sql-injection/

[26] Motoori, A, SQL Injection, QaFox, accessed February 2021 https://www.qafox.com/sql-injection-types/

[27] Imperva, SQL Injection, Imperva, accessed February 2021 https://www.imperva.com/learn/application-security/sql-injection-sqli/

[28] CyberCrowd, Impact of an SQL Injection, CyberCrowd, accessed February 2021 https://www.cybercrowd.co.uk/news/impact-of-a-sql-injection/

[29] PortSwigger, SQL injection, PortSwigger Cyber Security, accessed January 2021 https://portswigger.net/web-security/sql-injection

[30] Georgoulas, A 2020, Ασφάλεια στην τεχνολογία της πληροφορίας, SQL Injection, Πανεπιστήμιο Δυτικής Ατικής, Αθήνα

[31] OWASP, SQL Injection, OWASP Cyber Security, accessed February 2021 https://owasp.org/www-community/attacks/SQL_Injection

[32] Edgescan, Vulnerability Statistics report 2020, Edgescan, accessed March 2021 https://cdn2.hubspot.net/hubfs/4118561/BCC030%20Vulnerability%20Stats%20Report%20(2020)_WEB.pdf

[33] Factor, P 2018, How to Detect SQL Injection Attacks, RedGate, retrieved from https://www.red-gate.com/hub/product-learning/sql-monitor/detect-sql-injection-attacks-using-extended-events-sql-monitor

[34] Kumar, C 2020, How to find SQL Injection Vulnerabilities, Geekflare, retrieved from https://geekflare.com/find-sql-injection/

[35] Gamble, J & Szeto, P 2009, How to detect and stop SQL injection attacks, ComputerWeekly, retrieved from https://www.computerweekly.com/tip/How-to-detect-and-stop-SQL-injection-attacks

[36] Acunetix, Victiom of SQL Injection attack , Acunetix, accessed January 2021 https://www.acunetix.com/blog/articles/mysql-com-victim-of-sql-injection/

[37] OWASP, Blind SQL Injection, OWASP Cyber Security accessed March 2021 https://owasp.org/www-community/attacks/Blind_SQL_Injection

[38] Ahmed, A 2020, Freepik hacked via SQL Injection, Digital Information World, retrieved from https://www.digitalinformationworld.com/2020/08/hackers-stole-emails-and-password-hashes-for-8-million-freepik-users.html

[39] GbAdvisors, 2020, Ways to prevent SQL Injection, GbAdvisors, retrieved from https://www.gb-advisors.com/sql-injection-prevention-using-the-acunetix-tool/

[40] Franceschi, L 2015, One of the largest hacks, Vice, retrieved from https://www.vice.com/en/article/yp3z5v/one-of-the-largest-hacks-yet-exposes-data-on-hundreds-of-thousands-of-kids

[41] Stallings W & Brown L, (2016), Ασφάλεια Υπολογιστών – Αρχές και πρακτικές, (3η Αμερικανική έκδοση), ΕΚΔΟΣΕΙΣ ΚΛΕΙΔΑΡΙΘΜΟΣ, ΑΘΗΝΑ

[42] Acunetix, SQL Injection Scanner, Acunetix, accessed February 2021 https://www.acunetix.com/vulnerability-scanner/sql-injection-scanner/

[43] Subgraph, Vulnerability Scanner, Subgraph, accessed February 2021 https://subgraph.com/vega/

[44] Sqlmap, Automatic SQL injection and database takeover tool, Sqlmap, accessed February 2021 http://sqlmap.org/

[45] Berkeley Information Security Office, How to protect again SQL injection attacks, Berkeley Information Security Office, accessed March 2021 https://security.berkeley.edu/education-awareness/how-protect-against-sql-injection-attacks

[46] Positive Technologies, 2019, How to prevent SQL injection attacks, Positive Technologies, retrieved from https://www.ptsecurity.com/ww-en/analytics/knowledge-base/how-to-prevent-sql-injection-attacks/

[47] OWASP, SQL Injection prevention Cheat Sheet, OWASP Cyber Security, accessed February 2021
https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html

[48] W3Schools, SQL Stored Procedures, W3Schools, accessed March 2021 https://www.w3schools.com/sql/sql_stored_procedures.asp

[49] OWASP, Input Validation Cheat sheet, OWASP Cyber Security, accessed March 2021 https://cheatsheetseries.owasp.org/cheatsheets/Input_Validation_Cheat_Sheet.html

[50] W3Schools, Mysqli real escape string function, W3Schools, accessed March 2021 https://www.w3schools.com/php/func_mysqli_real_escape_string.asp

[51] Hypr, Cryptografic Hash Function, Hypr Security Encyclopedia, accessed February 2021 https://www.hypr.com/cryptographic-hash-function/

[52] Imperva, The state of vulnerabilities in 2019, Imperva, accessed February 2021 https://www.imperva.com/blog/the-state-of-vulnerabilities-in-2019/

[53] OWASP, Vulnerabilities, OWASP Cyber Security, accessed February 2021 https://owasp.org/www-community/vulnerabilities/

[54] PurpleSec, 2021 Cyber Security Statistics, PurpleSec, accessed February 2021 https://purplesec.us/resources/cyber-security-statistics/

[55] OWASP, Testing for SQL injection, OWASP Cyber Security, accessed February 2021 https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/07-Input_Validation_Testing/05-Testing_for_SQL_Injection

[56] Beagle, 2018, Time Based Blind SQL injection, Beagle Security, retrieved from https://beaglesecurity.com/blog/vulnerability/time-based-blind-sql-injection.html

[57] Pypl, Top DB Index, Pypl, accessed February 2021 https://pypl.github.io/DB.html

[58] Oracle, What is Oracle Database, Oracle, accessed March 2021 https://www.oracletutorial.com/getting-started/what-is-oracle-database/

[59] Vaughan, J, Multimodel Database, TechTarget, accessed March 2021 https://searchdatamanagement.techtarget.com/definition/multimodel-database

[60] Sqlsplus, Most popular DBMS, Sqlplus, accessed February 2021 https://www.sqlsplus.com/the-most-popular-database-management-systems-dbms-in-the-world-in-2020/

[61] Moore, L, MySQL, TechTarget, accessed March 2021 https://searchoracle.techtarget.com/definition/MySQL

[62] Hughes, A, Microsoft SQL Server, TechTarget, accessed March 2021 https://searchsqlserver.techtarget.com/definition/SQL-Server

[63] Shankdhar, P 2021, Open source SQL injection tools,InfoSecInstitute, retrieved from https://resources.infosecinstitute.com/topic/best-free-and-open-source-sql-injection-tools/

[64] Kost, S 2007, An introduction to SQL Injection Attacks for Oracle Developers, Integrigy Corporation, Chicago USA, retrieved from https://www.yumpu.com/en/document/read/23444825/an-introduction-to-sql-injection-attacks-for-oracle-integrigy

[65] Clarke, J 2009, SQL Injection Attacks and Defense, Singress Publishing, Inc., Elsevier, Inc., Burlington USA

[66] TrustRadius, Static Code Analysis, TrustRadius, accessed March 2021 https://www.trustradius.com/static-code-analysis

ΠΑΡΑΡΤΗΜΑ Α' (Κώδικας του εργαλείου)

```
import requests
from bs4 import BeautifulSoup as bs
from urllib.parse import urljoin
from pprint import pprint


s = requests.Session()
s.headers["User-
Agent"] = "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, l
ike Gecko) "\
                          "Chrome/83.0.4103.106 Safari/537.36 "


def get_all_forms(url):

    soup = bs(s.get(url).content, "html.parser")
    return soup.find_all("form")


def get_form_details(form):

    details = {}
    try:
        action = form.attrs.get("action").lower()
    except:
        action = None
    method = form.attrs.get("method", "get").lower()
    inputs = []
    for input_tag in form.find_all("input"):
        input_type = input_tag.attrs.get("type", "text")
        input_name = input_tag.attrs.get("name")
        input_value = input_tag.attrs.get("value", "")
        inputs.append({"type": input_type, "name": input_name, "value": input_val
ue})


    details["action"] = action
    details["method"] = method
    details["inputs"] = inputs
    return details
```

```
def is_vulnerable(res):

    errors = {
        # MySQL
        "MySQLSyntaxErrorException",
        "valid MySQL result",
        "you have an error in your sql syntax;",
        "warning: mysql",
        "Unknown column '[^ ]+' in 'field list'",
        "MySqlException",
        "SQLSTATE\[\d+\]: Syntax error or access violation",
        "com\.mysql\.jdbc",
        "MySqlClient\.",
        "check the manual that (corresponds to|fits) your MySQL server version",
        # PostgreSQL
        "PostgreSQL.*?ERROR",
        "valid PostgreSQL result",
        "PG::SyntaxError:",
        # SQL Server
        "Driver.*? SQL[\-\_\ ]*Server",
        "OLE DB.*? SQL Server",
        "Warning.*?\W(mssql|sqlsrv)_",
        "\[SQL Server\]",
        "unclosed quotation mark after the character string",
        # Oracle
        "\bORA-\d{5}",
        "Oracle error",
        "Oracle.*?Driver",
        "quoted string not properly terminated",
        "SQL command not properly ended",
        "macromedia\.jdbc\.oracle",
        "oracle\.jdbc",
        "OracleException",
        # Microsoft Access
        "Microsoft Access (\d+ )?Driver",
        "JET Database Engine",
        "Access Database Engine",
        "ODBC Microsoft Access",
        "Syntax error \(missing operator\) in query expression"
    }

    for error in errors:

        if error in res.content.decode().lower():
            #print (error)
```

```
            return True

    return False


def scan_sql_injection(url):


    with open('url_lines.txt') as f:
        url_lines = f.readlines()

    for c in url_lines:
        new_url = f"{url}{c}"
        print(" ")
        print("[!][URL INJ] Trying", new_url)
        res = s.get(new_url)

        if res.status_code == requests.codes.ok :
            print (res.status_code, "- Success Code")
        else:
            print (res.status_code, "- Client Error")

        if is_vulnerable(res):
            print("[+] SQL Injection vulnerability detected, link:", new_url)
            continue

    print('--------------------------')

    with open('html_submit.txt') as f:
        html_submit = f.readlines()

    forms2 = get_all_forms(url)
    for form2 in forms2:
        form_details2 = get_form_details(form2)

        if len(forms2) > 0:
            print(f"[+] Detected {len(forms2)} form(s) on the URL given : {url}")
            print("[+] Form:")
            pprint (form_details2)

        for c in html_submit:
            data = {}

            for input_tag in form_details2["inputs"]:
                if input_tag["value"] or input_tag["type"] == "hidden":
```

```
                    try:
                        data[input_tag["name"]] = input_tag["value"] + c
                    except:
                        pass
                elif input_tag["type"] != "submit":
                    data[input_tag["name"]] = f"Admin{c}"

            url = urljoin(url, form_details2["action"])

            if form_details2["method"] == "post":
                res = s.post(url, data=data)
            elif form_details2["method"] == "get":
                res = s.get(url, params=data)

            if is_vulnerable(res):
                print("[+][FORM INJ] SQL Injection vulnerability detected, link:"
, url)
                print("[+] Form:")
                pprint(form_details2)
                break


if __name__ == "__main__":
    import sys
    url = sys.argv[1]
    scan_sql_injection(url)
```

Ορισμένα function και κώδικας αντλήθηκαν απο την παρακάτω πηγή: https://github.com/x4nth055/pythoncode-tutorials/tree/master/web-scraping/extract-and-fill-forms