

SQL Injection (SQLi)

Τι είναι;

- Πρόκειται για ευπάθεια στην οποία ο attacker μπορεί να τροποποιήσει τα queries που πραγματοποιεί μια εφαρμογή στη βάση δεδομένων με την οποία επικοινωνεί ο web server της
- Προκύπτει όταν είσοδος του χρήστη αποτελεί τμήμα του query (injection) που εκτελείται, χωρίς όμως να έχει προηγηθεί κατάλληλο sanitization
- Υπάρχουν διάφορες παραλλαγές με τις οποίες μπορεί να εμφανιστεί ένα SQLi vulnerability.



Μας ενοχλεί;

Μία ευπάθεια τύπου SQLi είναι ιδιαίτερα σημαντική καθώς μπορεί να οδηγήσει σε:

- Unauthorized access
- Information leak
- Server compromise (συνήθως για αυτό απαιτείται κάποιου είδους escalation του SQLi vulnerability)



Ένα τυπικό παράδειγμα

Ας φανταστούμε ένα login page όπου ο χρήστης δίνει ως είσοδο το username και το password του. Η εφαρμογή, προκειμένου να ελέγξει την εγκυρότητα εκτελεί το query:

```
SELECT * FROM users WHERE username = 'user' AND password = 'w3b_sucks'
```

Σε περίπτωση όμως όπου δε γίνεται κατάλληλος έλεγχος της εισόδου στα δύο πεδία (username, password), ο χρήστης μπορεί να πετύχει login χωρίς password...



Πώς;

Αρκεί να κάνει τη συνθήκη του username αληθή και να σβήσει (comment out) το υπόλοιπο query χρησιμοποιώντας το string `--`

- `SELECT * FROM users WHERE username = 'admin'--'AND password = 'kek'` (προϋποθέτει τη γνώση του χρήστη admin)
- `SELECT * FROM users WHERE username = '' OR 1=1 --'AND password = 'kek'`

ΠΡΟΣΟΧΗ στους χαρακτήρες `'` και στην τοποθέτησή τους!!



FYI

- Προφανώς, SQLi μπορούμε να έχουμε σε οποιοδήποτε σημείο ενός query στο οποίο μπορούμε να τοποθετήσουμε κώδικα που δεν ελέγχεται
- Επικρατέστερο σενάριο είναι τα `WHERE` statements των *SELECT* queries
- Ωστόσο, δεν αποκλείεται SQLi σε *INSERT* και *UPDATE* statements, στα επιμέρους πεδία τους



UNION attacks

Ένα άλλο σενάριο που εμφανίζει ενδιαφέρον είναι η χρήση του τελεστή *UNION* της SQL. Μέσω αυτού μπορούμε να πετύχουμε διαρροή δεδομένων (και όχι μόνο).

Αν ας πούμε μία εφαρμογή επέστρεφε τα άτομα μίας σχολής με ειδικότητα που ελέγχεται από τον χρήστη (πχ τους καθηγητές) θα εκτελούσε το query

```
SELECT name FROM members WHERE role = 'Professor'
```

Σε περίπτωση SQLi όμως, ένας κακόβουλος θα μπορούσε να ζητήσει:

```
SELECT first_name, last_name FROM members WHERE role =  
'Professor' UNION SELECT username, password FROM admins --
```

Έτσι, του επιστρέφονται όχι μόνο τα ονόματα των καθηγητών της σχολής αλλά και τα credentials των διαχειριστών της βάσης!!

Blind SQLi

Εμφανίζεται όταν παρά την ύπαρξη του SQLi, τα HTTP responses δεν περιέχουν τα αποτελέσματα του σχετικού query (η πιθανά errors). Απαιτεί πιο advanced τεχνικές exploitation σε σχέση με τις προηγούμενες.

Μία από αυτές σχετίζεται με την πρόκληση τεχνητών καθυστερήσεων (delays) από τη μεριά του attacker προκειμένου να μπορέσει να διαρρεύσει με βεβαιότητα στοιχεία της βάσης (Time based SQLi). Για παράδειγμα μπορεί να πραγματοποιήσει bruteforce στους χαρακτήρες του password ενός admin ή να μάθει στοιχεία για την έκδοση της βάσης.

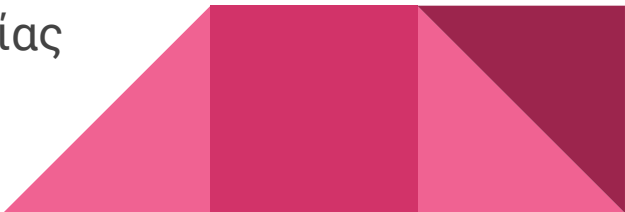


Time based SQLi

Για παράδειγμα αν μία εφαρμογή έχει endpoint όπου ο χρήστης διαλέγει το id ενός προϊόντος και αυτό είναι vulnerable σε SQLi, τότε ένας επιτιθέμενος προκειμένου να μάθει την έκδοση της MySql βάσης (εφόσον ξέρει ότι είναι MySql) θα μπορούσε να δώσει ως είσοδο:

```
SELECT * FROM products WHERE id = 1 AND IF(SUBSTRING(version(),  
1,1)=5, SLEEP(10), null)
```

Εάν η έκδοση της MySql της βάσης ξεκινάει με 5, τότε (και μόνον τότε) το παραπάνω query θα οδηγήσει σε delay 10 δευτερολέπτων το αντίστοιχο http response, με αποτέλεσμα να έχουμε διαρροή πληροφορίας χωρίς ωστόσο αυτή να είναι ορατή στον attacker.



Γενικά

- Όπως με κάθε vulnerability, έτσι και για τα SQLi, πρώτα πρέπει να μπορέσουμε να αναγνωρίσουμε την ύπαρξή τους
- Εάν έχουμε το source code, τότε ψάχνουμε για input χρήστη το οποίο καταλήγει να συνιστά μέρος κάποιου query προς εκτέλεση. Εάν δε γίνεται κατάλληλος έλεγχός του, τότε μάλλον υπάρχει SQLi.
- Αν δεν υπάρχει source code τότε τα πράγματα είναι δυσκολότερα...
- Σε κάθε περίπτωση πάντως, δε χάνουμε τίποτα να δοκιμάζουμε για SQLi σε κάθε login form που συναντάμε :)



Γενικά (1)

- Τα SQLi payloads συχνά διαφοροποιούνται μεταξύ τους ανάλογα με τη βάση δεδομένων στην οποία εκτελούνται. Για αυτό είναι ιδιαίτερα σημαντική η γνώση του συστήματος το οποίο προσπαθούμε να κάνουμε exploit.
- Πολλές φορές χρειάζεται υπομονή μέχρι να βρούμε κάποιο payload που να πετυχαίνει το SQLi που επιθυμούμε (θα έλεγε κανείς τι πιο σύνηθες στα web...)
- Η ύπαρξη SQLi δε συνεπάγεται αυτόματα την ικανότητα διαρροής πληροφοριών ή την εκμετάλλευση ενός server. Συνιστά ωστόσο, σημαντικότερη ευπάθεια.
- Για να αμυνθούμε απέναντι σε SQLi attacks πρέπει να φροντίζουμε να ελέγχουμε κάθε input χρήστη και να το φιλτράρουμε κατάλληλα.

SQLmap

Πρόκειται για εργαλείο που αυτοματοποιεί αρκετά από τα attacks που σχετίζονται με SQLi και είναι ιδιαίτερα χρήσιμο, καθώς παρέχει πάρα πολλές δυνατότητες σε έναν επιτιθέμενο.

Για τον σκοπό αυτό σας προτείνουμε να το κοιτάξετε και να ανακαλύψετε μόνοι σας τις φοβερές λειτουργικότητές του. Για όσους ενδιαφέρονται, υπάρχει και ένα εισαγωγικό room στο TryHackMe το οποίο επίσης προτείνουμε.

<https://tryhackme.com/room/sqlmap>



Resources

- <https://portswigger.net/web-security/sql-injection>
 - https://owasp.org/www-community/attacks/SQL_Injection
 - <https://tryhackme.com/room/sqlinjectionlm>
 - <https://tryhackme.com/room/sqlilab>
- 