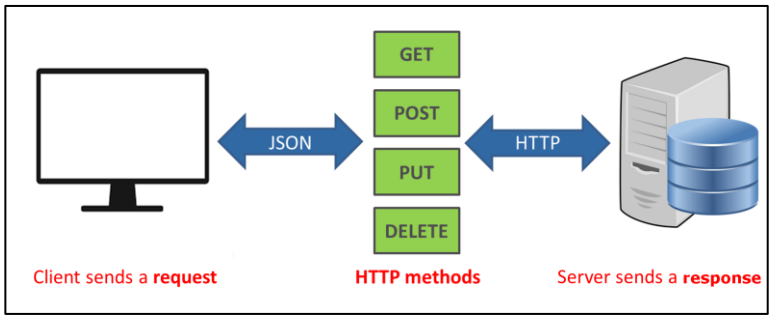
GraphQL וחולשות נפוצות

מאת אבישי גונן

הקדמה

לכל אתר יש מידע שהוא מספק ללקוח, ובשביל זה הוא משתמש ב-API (Application program interface).
במשך שנים רבות, המימוש הנפוץ של ה-API היה REST, ראשי תיבות של Representational State Transfer.
בקצרה מאוד, יש נקודת קצה, לדוגמא /users או /posts, והמשתמש שולח בקשת HTTP פשוטה ל-endpoint,
ומקבל תשובה שהכילה את המידע שהמשתמש ביקש.

באופן הזה היה ניתן לשלוח בקשות שונות, לדוגמא PUT לעדכון מידע או GET לקבלת מידע. אתרים
שמשתמשים בשיטה הזאת מספקים endpoint עבור כל ישות, לדוגמא /users/15 וכן /users/16.



ישנן שתי בעיות מרכזיות בשיטה הזאת.

1. **over-fetching**

המשתמש מקבל מידע מיותר, וזה עלול לסרבל וגם לפגוע בביצועים. לדוגמא, השם של המשתמש עם המספר
הסידורי 17, אתה תקבל בתור תשובה את האובייקט של המשתמש, וזה יכול להכיל עוד הרבה שדות שלא
היית צריך בשאילתא הזאת.

2. **under-fetching**

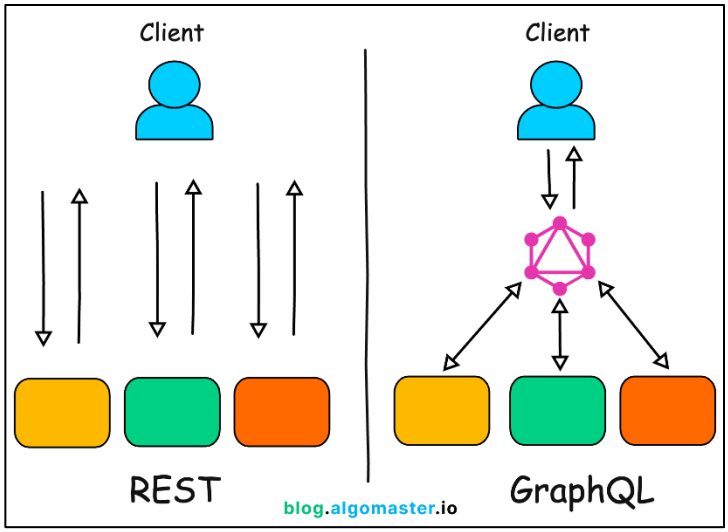
לפעמים כשתרצה לקבל מידע המורכב מקשרים בין אובייקטים, תצטרך לפנות להרבה endpoints שונים.
לדוגמא, אם תרצה למצוא את כל המשתמשים שגרים בפתח תקווה, תצטרך לפנות לכל נקודות הקצה של
המשתמשים, כלומר, ל-/users/5, /users/6, /users/7 וכו'.

בעקבות הבעיות שהצגנו לעיל, חברת Facebook פיתחו את **GraphQL**, קיצור של Graph Query Language. השפה פותחה על ידם ב-2012 והפכה לקוד פתוח ב-2015, אולם, למרות גילה הצעיר יחסית ל-REST, היא נכנסה לשימוש נרחב בשל היותה גמישה ופשוטה.

הרעיון ב-GraphQL הוא שהמשתמש יתקשר מול נקודת קצה אחת, שבה ינוהל סוג של גרף נתונים המכיל קשרים בין האובייקטים השונים. בנוסף, המשתמש יוכל לבקש שדות ספציפיים ולקבל רק את המידע שהוא צריך.

בצורה זו פתרנו את בעיית ה-**over fetching**, כיוון שעכשיו ניתן לקבל רק את שם המשתמש עם המספר הסידורי 17, ולא נצטרך לקבל את כל האובייקט.

בנוסף, פתרנו את בעיית ה-**under fetching**, כי כעת אנחנו מתקשרים מול endpoint יחיד.



במאמר הבא אנחנו נלמד איך עובדים עם graphQL, מה זה-graph QL scehma ומה סוגי השאילתות השונות. כמו כן, נראה איך למצוא endpoints של graphQL, ונלמד איך להשתמש ב-burp suite בהקשר של graphQL.

לאחר שנלמד על הטכנולוגיה, נסקור חולשה אמיתית שמצאתי לאחרונה באתר ידוע, נראה כיצד ניתן למצוא חולשות כאלו בעצמנו ואיך לנצל אותן, וכמובן, נציג את ה-mitigations האפשריים.

GraphQL schema

ב-GraphQL כל המידע חשוף דרך **schema** אחת. ה-schema מתארת אילו types של אובייקטים קיימים, אילו שדות יש להם וכמובן מה ה-type של כל שדה.

בנוסף, הסכימה מכילה את הפעולות האפשריות שניתן לבצע, שאילתות המיועדות לשליפת מידע המכונות **queries**, שאילתות המיועדות לעדכון מידע המכונות **mutations** וכן שאילתות המיועדות לעדכונים בזמן אמת המכונות **subscriptions** - לא נפרט עליהן במאמר זה, כיוון שהן אופציונליות. בהמשך המאמר נתמקד בסוגי השאילתות השונות ונבין איך לעבוד איתן.

```
type User {
    id: ID!
    name: String!
    email: String!
    posts: [Post!]!
}
type Post {
    id: ID!
    title: String!
    content: String
    author: User!
}
```

בעצם, כל אובייקט הוא גם type בפני עצמו שיכול להיות שדה באובייקט אחר. בדוגמא הזו, אנחנו יכולים לראות של-User יש מערך של post.

ה-**!** מסמן שזה שדה שלא יכול להיות null. בדוגמא הזאתי אנחנו יכולים לראות שכל משתמש מכיל מערך של post, כשהמערך עצמו חייב להכיל לפחות post אחד. בנוסף, כל post מכיל שדה author מסוג User, שבעצם זה הפנייה ליוצר ה-post.

Queries

query אלו שאילתות קבלת המידע, נראה כיצד מגדירים אותן בסכמה, ולאחר מכן כיצד ניתן להשתמש בהן.

הגדרת ה-Query

```
type Query {
    getUser(id: ID!): User
    listPosts: [Post!]!
}
```

כפי שניתן לראות, הגדרנו פה שתי שאילתות שונות מסוג **query.** שאילתא יכולה לקבל פרמטרים, שיצויינו בתוך הסוגריים המעוגלות אם קיימות, וכן מציינת את סוג הערך המוחזר, ע"י ה-type שמופיעה לאחרי הנקודותיים. במקרה שלנו השאילתא getUser מקבלת פרמטר מסוג id. אותו פרמטר לא יכול להיות null כיוון שיש **!** לאחר ה-type שלו. השאילתא מחזירה אובייקט מסוג User. לעומת זאת, השאילתא listPosts לא מקבלת שום פרמטרים, ומחזירה מערך של post.

שימוש ב-Query

לאחר שראינו כיצד מגדירים **query**, נראה כעת איך משתמשים בשאילתא כזו.

```
query myGetUserQuery {
    getUser(id: "1") {
        name
        email
    }
}
```

בתור התחלה נציין את סוג הפעולה, במקרה הזה query. זה אופציונלי אך מקובל לכתוב אותו. בהמשך, ניתן שם לשאילתא, במקרה הזה myGetUserQuery, גם זה אופציונלי אך זה נוח למטרות דיבוג ולוגים.

לאחר מכן, ניתן את שם הפעולה בפועל כפי שמוגדרת ב-schema, וכן ניתן את הארגומנטים כפי שמצויין בהגדרת הסכמה.

לבסוף, ניתן את השדות אותם אנחנו רוצים לשלוף. פה מתחיל הטריק, כפי שניתן לראות, בשאילתא הזאת ביקשתי רק את ה-name וה-email של המשתמש עם המספר הסידורי 1, פה טמונה הגמישות של **graphQL**.

אם נרצה לבקש את ה-posts, נשים את זה בתוך אובייקט מקונן, כפי שניתן לראות בדוגמא הבאה.

```
query {
    getUser(id: "1") {
        name
        email
        posts {
            title
            content
        }
    }
}
```

נוכל לקחת את זה שלב קדימה ולבקש את שדה ה-author, כפי שניתן לראות בדוגמא הבאה.

```
query {
    getUser(id: "1") {
        name
        email
        posts {
            title
            content
            author {
                name
            }
        }
    }
}
```

Mutations

אלו שאילתות קבלת המידע, נראה כיצד מגדירים אותן בסכמה, ולאחר מכן כיצד ניתן להשתמש בהן.

הגדרת ה-Mutation

```
type Mutation {
    createUser(name: String!, email: String!): User
    createPost(title: String!, content: String, authorId: ID!): Post
}
```

השאילתא מקבלת ארגומנטים, ומחזירה בסופו של דבר את האובייקט לאחר העדכון. במידה והאובייקט לא קיים, השאילתא יוצרת אובייקט חדש ובסוף מחזירה אותו למשתמש, כדי שלא נצטרך להשתמש בשאילתת query שתביא לנו את האובייקט שיצרנו.

שימוש ב-Mutation

לאחר שראינו כיצד מגדירים **mutation**, נראה כעת איך משתמשים בשאילתא כזו.

```
mutation myUpdateUser {
    createUser(name: "Bob", email: "bob@theBuilder.com") {
        id
        name
        email
    }
}
```

בתור התחלה נציין את סוג הפעולה, במקרה הזה mutation. זה לא אופציונלי, כי אם נשמיט את זה הוא ילך לברירת המחדל, שזה query. בהמשך, ניתן שם לשאילתא, במקרה הזה myUpdateUser, זה אופציונלי אך זה נוח למטרות דיבוג ולוגים.

לאחר מכן, ניתן את שם הפעולה בפועל כפי שמוגדרת ב-schema, וכן ניתן את הארגומנטים כפי שמצויין בהגדרת הסכמה. לבסוף, ניתן את השדות אותם אנחנו רוצים לשלוף, באותה הצורה כפי שראינו ב-query. בסוף mutation זה גם עדכון מידע, וגם בסוף עושה פעולה של query.

נוכל להשתמש גם בקינון בשביל להשיג את המידע מתוך אובייקטים שהם לא מסוג type פשוט, אלא מסוג אובייקט שהגדרנו אותו בסכמה עם שדות.

```
mutation myUpdateUser {
    createUser(name: "Bob", email: "bob@theBuilder.com") {
        id
        name
        email
    }
}
```

Introspection

introspection היא תכונה מובנית ב-GraphQL שנותנת למשתמש את היכולת לשלוח בקשות שייתנו לו את המבנה של כל הסכמה, כדי שהוא יוכל לדעת אילו פעולות הוא יכול לבצע, ואיזה מידע הוא יכול לקבל.

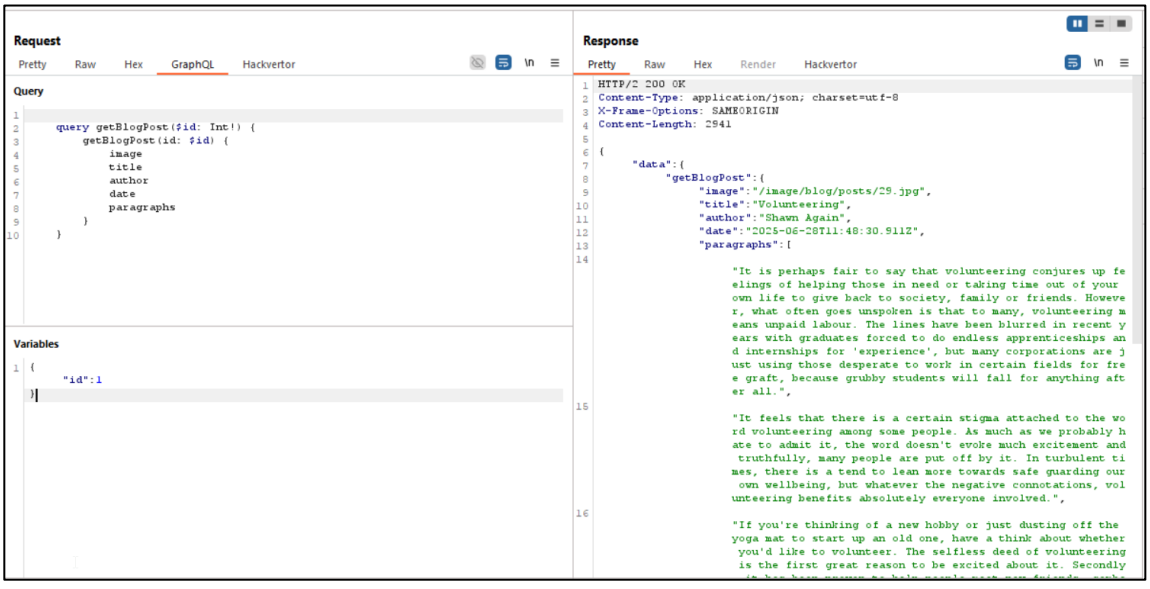
התכונה הזאת משמשת כלים לעבודה עם GraphQL, ובאמצעותה הם יכולים להשלים שאילתות באופן אוטומטי. במילים פשוטות, השאילתא הזאת "מדליפה" את כל המבנה הפנימי של הסכמה, אולם, זה לאו דווקא מהווה סיכון.

אתרים כמו http://nathanrandal.com/graphql-visualizer מספקים לך את שאילתת ה-introspection, וכן נותנים ציור וויזואלי של הסכמה, שהרי בסוף היא גרף, אם תיתן להם את התשובה (כמובן בלי ה "data" שעוטף את המידע, אלא רק את המידע עצמו).

ישנן עוד תכונות ופיצ'רים רבים שהטכנולוגיה הזאת מציעה, אנחנו לא נתמקד בהם במאמר כיוון שבאנו ללמוד קצת על אבטחת מידע, אבל בכל זאת, אתם מוזמנים להסתכל במקורות הנוספים שמצורפים בסוף המאמר.

שימוש ב-Burp Suite ב-GraphQL Injection

Burp suite זה כלי חינמי שמיוצר ע"י החברה port swigger. הכלי הזה מהווה פרוקסי שבאמצעותו ניתן לבחון אתרים ורשתות. בגדול זה הכלי מספר אחד בכל התחום הזה, גם בגרסת ה-community החינמית שלו, הוא נחשב לכלי שאין לו מתחרים ברמה בשוק. הכלי יכול לזהות לבד כשיש GraphQL, ונוכל לראות שבחלון ה-Repeater שלו תפתח חלונית שבה נוכל לערוך את שאילתות ה-GraphQL שלנו.



GraphQL Injection בעולם האמיתי

לפני כמה שבועות ישבתי וגלשתי באתר גדול ומוכר בארץ, וראיתי שיש בקשות לנקודת הקצה של ה-GraphQL שרצות ברקע. בגלל שבאותו שבוע בדיוק עשיתי CTF פשוט מאתר ה-webhacking.kr CTF's על GraphQL Injection, החלטתי לבדוק איך זה נראה בחיים האמיתיים.

ישנן endpoints ידועים ל-GraphQL כגון: /api/gql, /api/graphql, /gql, /graphql, /api, ועוד.

במקרה שלי אני זיהיתי את ה-endpoint מהשם של נקודת הקצה, שהוא gql. כמו כן, Burp Suite זיהה אוטומטית שמדובר בבקשת HTTP שמכילה שאילתת GraphQL ופתח חלונית לעבודה נוחה עם השאילתא, כפי שניתן לראות בתמונה למעלה.

זה די הגניב אותי שלא היה צריך להתקין שום תוסף בשביל זה. בכל מקרה, ישנם תוספים שימושיים ל-Burp Suite שיכולים לעזור מאוד, גם בגרסה החינמית, אני ממליץ בחום להתקין את התוספים כי זה מקל מאוד על העבודה.

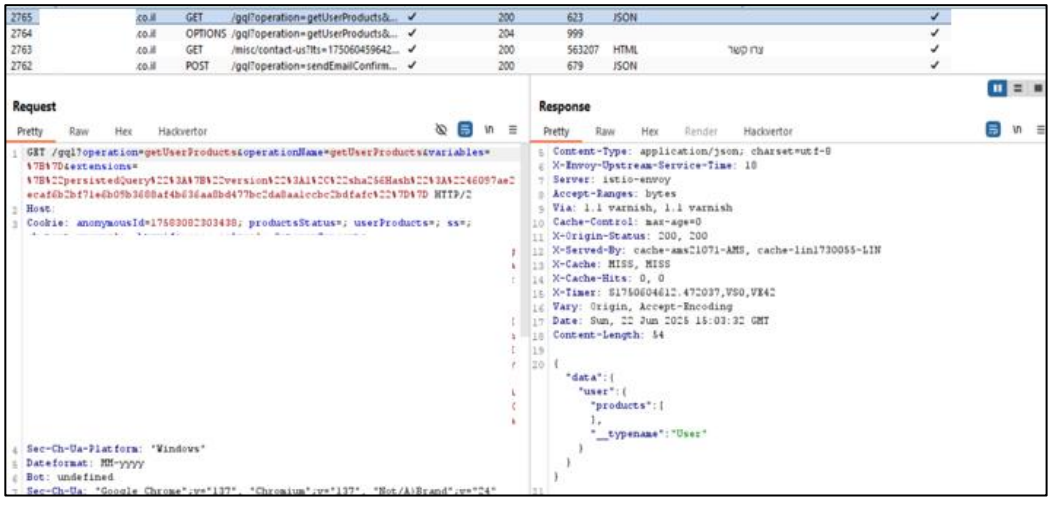
בכל מקרה, נחזור לנושא שלנו.

הדלפת ה-schema

שלחתי את שאילתת ה-**introspection** בשביל לקבל את המבנה השלם של הסכימה, ובאופן די מפתיע, קיבלתי אותה.

בסך הכל הסכימה הייתה כ-100,000 שורות לאחר שהדבקתי את התשובה ב-vscode.

בתמונה הבאה ניתן לראות את השלב בו גיליתי שהאתר משתמש ב-Graph QL, כפי שניתן לראות, נקודת הקצה היא gql ובתור תשובה לשאילתא שנשלחה, קיבלתי json שהביא לי אובייקט מסוג User.



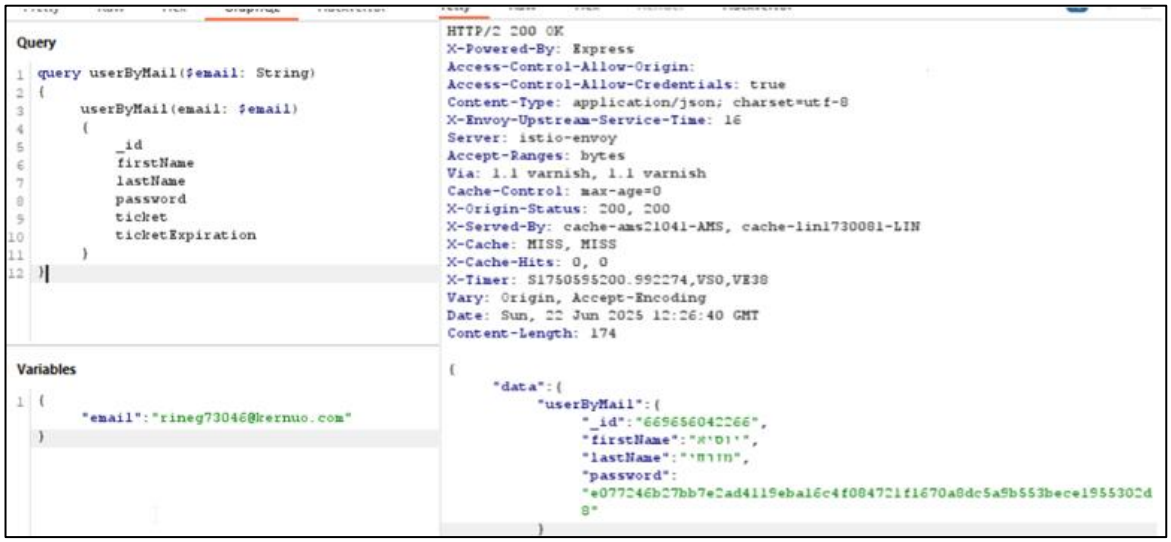
והנה הסכימה שלנו, מחכה שנחקור אותה :)

Unsalted password

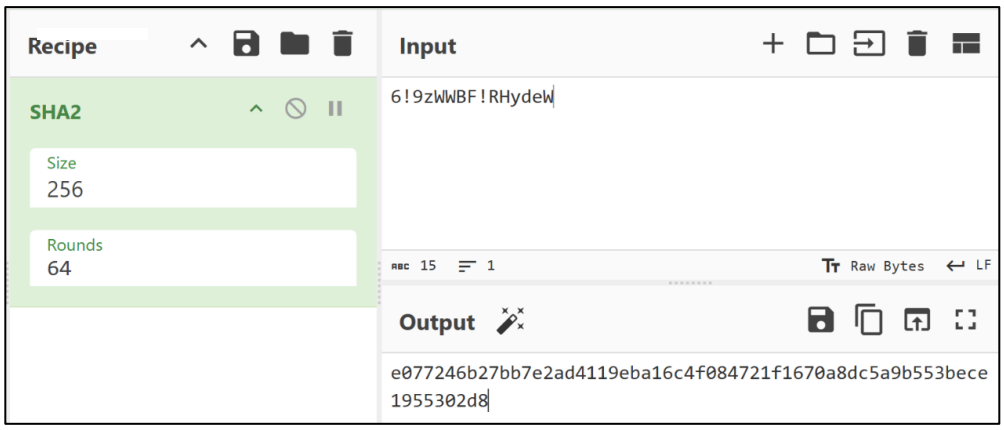
חיפשתי שאילתות שמחזירות לי אובייקט מסוג User, כיוון שראיתי שה-User מכיל גם שדה מסוג password, ורציתי לראות איך הסיסמא נשמרת במאגר.

מצאתי את השאילתא userByMail באמצעות חיפוש פשוט במבנה הסכימה שהשגתי (חיפשתי אחרי פונקציות שמחזירות User באמצעות ctrl+F פשוט ב-vscode).

נתתי את ה-mail שלי, בשביל לראות מה אני אקבל בתור תשובה.



אוקי, הסיסמא מגובבת ע"י **SHA256**, רק חבל שהיא בלי **salt**... (הסיסמא שלי היא 6!9zWWBF!RHydeW)



בכל מקרה, באנו לפה בשביל GraphQL, לא בשביל הוספת מלח בסיסמאות, אז יאללה, בוא נקפוץ אל ה-Graph QL Injection.

GraphQL Injection

ציטוט מ-chatgpt:

GraphQL Injection היא מתקפה שבה תוקף מנצל את הגמישות של GraphQL כדי להחדיר שאילתות זדוניות, במטרה לגשת למידע רגיש, לעקוף מנגנוני הרשאה או לחשוף את מבנה הסכמה של ה-API.

זה קורה כאשר השרת לא בודק או מסנן כמו שצריך את הקלט שמגיע מהמשתמש, מה שמאפשר לתוקף לשלוח שאילתות מותאמות אישית שלא היו אמורות להתבצע.

בדומה ל-SQL Injection, גם כאן אפשר לשלוף מידע שלא אמור להיות חשוף - לפעמים אפילו את כל בסיס הנתונים.

אוקי, ניסיתי להשתמש בשאילתא שראינו קודם ולשלוח מייל שלא שייך לי (עשיתי inspect לאתר ולקחתי מייל של אחד מהאנשים שעובדים שם, היה כפתור פשוט של sendEmail באתר), אבל לא קיבלתי את אובייקט ה-User שלו.

המשכתי בחיפוש בתוך הסכימה אחר שאילתות שמחזירות אובייקט מסוג User, תוך שימוש ב-ctrl+F פשוט. אולי נצליח באיזשהו אופן להשיג אובייקטים של משתמשים שהם לא אני, בלי הרשאות.

השגת מידע באמצעות מייל בלבד

מצאתי את השאילתה GetUserLogin, שמחזירה אובייקט מסוג UserLogin לפי כתובת מייל בלבד. השתמשתי בכתובת מייל שלא שייכת לי, לקחתי אותה מתוך קוד האתר. קיבלתי את פרטי המשתמש, כולל ה-ID שלו. בבדיקות נוספות גיליתי שה-ID הזה קבוע ולא משתנה, והוא גם זה שנשמר בעוגייה של הדפדפן, מה שמאפשר לאתר לזהות את המשתמש. העוגייה הזו לא מבוססת על session אלא על זיהוי קבוע, מה שנחשב לפרקטיקה בעייתית באבטחת מידע.

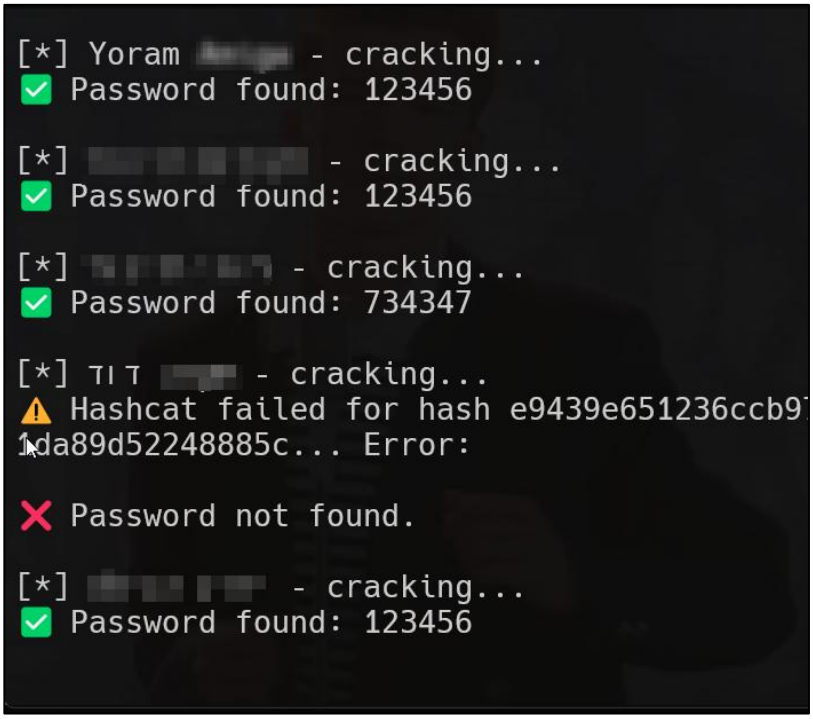
בנוסף, הצלחתי להשיג את מספר הטלפון שהיה מטושטש רק ע"י 4 כוכביות, לדוגמא, 052****8765.

למרות כל המידע הרגיש שהצלחתי לקבל, האובייקט שמכיל סיסמאות הוא User ולא UserLogin, לכן צריך להמשיך לחפש שאילתות שידליפו לי את ה-User.

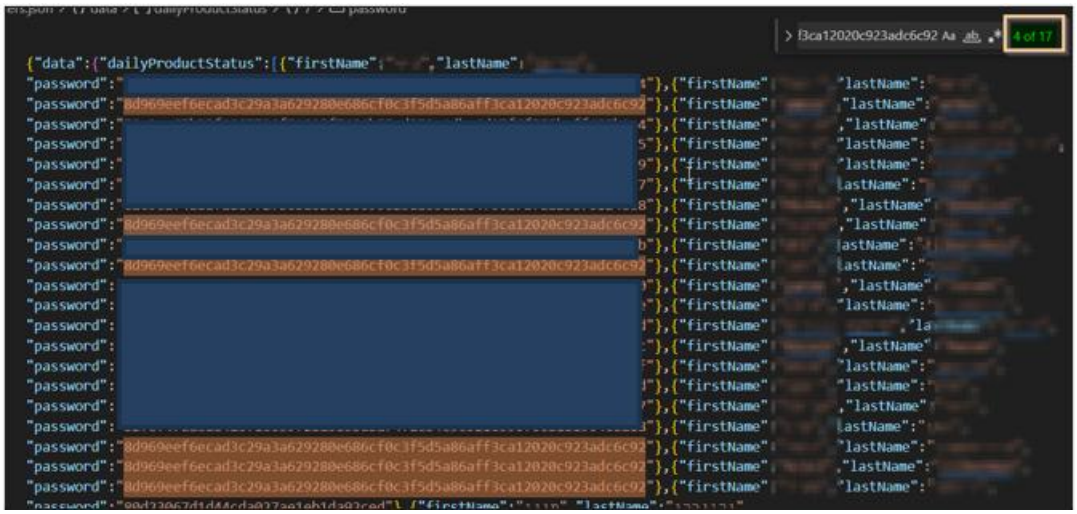
הדלפת אובייקטים מסוג User ללא הרשאות

מצאתי את השאילתא dailyProductStatus שמחזירה גם כן אובייקטים מסוג User, ניסיתי את מזלי ובום! קיבלתי רשימה של משתמשים עם כל המידע, כולל הסיסמאות מגובבות.

החלטתי לבדוק את החוזק של הסיסמאות, כדי להבין האם האתר אוכף מדיניות של סיסמאות חזקות. כתבתי סקריפט קצר עם python ו-hashcat לפריצת הסיסמאות, ואלו התוצאות...



מתוך 70 משתמשים שאספתי, הסיסמא 123456 הופיעה "רק" 17 פעמים...



לבסוף, לקחתי את אחד המשתמשים עם הסיסמא שפיענחתי (123456) וניסיתי להתחבר לאתר, וזה עבד!

שימוש ב-Mutations

בשלב הזה החלטתי לעצור ולא לעבור לשאילתות מסוג **mutation** כיוון שלא רציתי לגרום נזק.

אולם, נוכל להשתמש בתור דוגמא בשאילתא המכונה sendInvoice, שמקבלת בתור ארגומנט אובייקט מסוג SendInvoiceInput, שמכיל שלושה שדות: email, fromDate, toDate.

השאילתא מחזירה בתור תשובה אובייקט מסוג SendInvoiceResponse, שמכיל msg ו-success.

כיוון שאני יכול לשלוח את הבקשה הזאתי בלי שום id, אני מניח שזה פשוט שולח חשבונית למייל, בלי שום אותנטיקציה... באופן הזה, יכולתי להספים מישהו במיילים.

אוסיף ואומר שהיו עוד פעולות בסכימה שלא חקרתי אותן לעומק.

Mitigations

אציג כעת את ה-mitigtaions השונים שהיו יכולים למנוע את דלף המידע הקריטי שהצגתי לכם כעת.

- **לחסום את ה-Introspection**

התכונה הזאת נועדה לעזור למפתחים בשלב הפיתוח בלבד. עדיף שלא לתת למשתמש את האפשרות לקבל את כל מבנה הסכמה. ידועה כבר האמרה ש-**security by obfuscation** זה רעיון רע, אבל מצד שני, לא צריך להגיש על מגש של כסף לתוקף את כל מבנה המערכת.

- **ביצוע פעולות רק באמצעות הזדהות**

לדאוג שבכל שאילתא שפתוחה בפני המשתמש, יהיה שדה חובה שמייצג את המשתמש. לדוגמא, שתמיד יהיה את המספר הסידורי הייחודי שבאמצעותו מזדהה המשתמש. באופן זה המשתמש לא יוכל לגשת למידע שלא שייך לו.

- **לחסום פעולות שלא אמורות להיות חשופות למשתמשים**

פעולה שנותנת לי רשימה של כל המשתמשים מסוג מסויים לא אמורה להיות חשופה, אני לא מצליח לחשוב על מצב בו אני כן עשוי להשתמש בה. צריך להבין שלא כל מה שהמפתח משתמש בו אמור להיות פתוח ל-User.

- **להשתמש ב Persisted Queries**

לתת למשתמש רק סט מסויים של שאילתות שאותן הוא יוכל להריץ. נשמור hash של כל השאילתות המאושרות בשיטת **white-list** בצד שרת, ואז כשהמשתמש ירצה לשלוח שאילתא שהמערכת התירה, הוא פשוט ישלח את ה-hash והפרמטרים שהוא רוצה להעביר, והשרת ישווה את ה-hash לשאילתות השמורות אצלו.

לעניות דעתי זאת השיטה הכי פשוטה ויעילה לחסום את החולשה, הרי בסופו של דבר המשתמש צריך להריץ רק שורת שאילתות מצומצמת, וכן white-list זו פרקטיקה טובה לשמירה על אבטחת מידע.

בונוס למי שנשאר עד הסוף :)

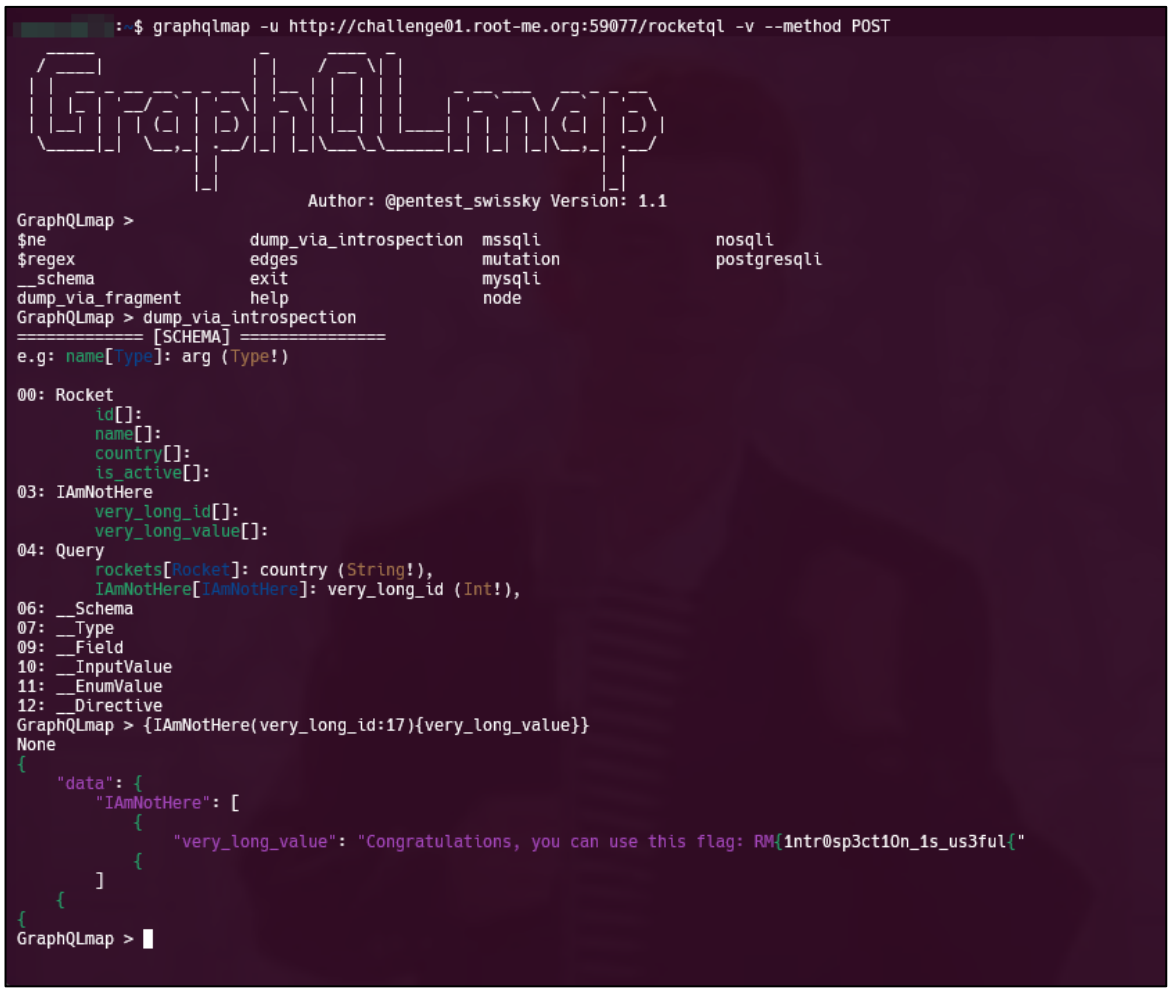
לאחר שסיימתי לכתוב את המאמר, התחלתי לשחק קצת באתר root-me, אתר CTF's, ובמקרה הגעתי ל-CTF הזה: https://www.root-me.org/en/Challenges/Web-Server/GraphQL-Introspection.

התחלתי לעבוד עליו, כשאז מצאתי את הכלי הזה: https://github.com/swisskyrepo/GraphQLmap, וחשבתי שכדאי להוסיף לסוף המאמר דוגמא לאיך פותרים CTF כזה ביחד עם הכלי החזק הזה.
(שימו לב מאיפה הרפו הזה, מ-**swisskyrepo** , אותו אחד שמתפעל גם **PayloadsAllTheThings** שמי שלא מכיר, כדאי מאוד להכיר, מאגר מאוד טוב ושימושי, עם כמעט 70 אלף כוכבים...)
אני רק מריץ את השורה:

```
graphqlmap -u http://challenge01.root-me.org:59077/rocketql -v --method POST
```

ואחרי זה נפתחות בפני האפשרויות, בחרתי ב-dump_via_introspection , ואוטומטית הכלי פורס את כל הסכמה.

היופי בכלי הזה הוא ההשלמה האוטומטית שהוא מספק. הוא זוכר כל אובייקט ושדה בהם הוא נתקל, ויודע להשלים בלחיצה פשוטה על טאב, זה מאוד נוח.



(עשיתי brute-force לכל ה-very_long_id מ-1 והלאה בעזרת burp intruder, אני הצגתי לכם רק את ה-very_long_id של הדגל).

סיכום

במאמר הזה הצגתי את הטכנולוגיה GraphQL, מה היחס בינה לבין REST וכיצד ניתן להשתמש בה.

ראינו מה זה query ו-mutaition, וכן ראינו כיצד ניתן להשתמש ב-introspection בשביל לקבל את מבנה ה-schema.

לקראת סוף המאמר עברנו לדבר על GraphQL Injection, הצגתי חולשה קריטית של דלף מידע שמצאתי לאחרונה, וראינו איך לנצל את החולשה בקלות באמצעות Burp Suite.

יש לשים לב שהשתמשתי במהלך ניצול החולשה רק בשאילתות מסוג query, ולא נגעתי בכלל ב-mutations, השאילתות שנועדו לשינוי מידע. כמו כן, לא ניסיתי להדליף מידע שקשור לכרטיסי אשראי, חיפשתי בסכמה ומצאתי אובייקטים שקשורים לתשלום.

על המחבר

אהלן, קוראים לי אבישי גונן, בן 20, תלמיד ישיבת הר עציון שמתעסק בזמנו הפנוי באבטחת מידע. אני אוהב לעשות CTFים ולהרחיב את הידע שלי בתחומים של אבטחת מידע.

מוזמנים לפנות אליי במייל avishaigonen@gmail.com בלינקדין, או בגיטהאב.

כמו כן, מוזמנים להציץ באתר שלי https://avishaigonen123.github.io/, או לקפוץ ישירות לדף ה-writeup's של ה-CTF-ים שפתרתי https://avishaigonen123.github.io/CTF_writeups.

מקורות מידע

- https://www.youtube.com/watch?v=xMCnDesBggM&list=PL4cUxeGkcC9gUxtblNUahcsg0WLxmrK_y - פלייליסט קצר ביוטיוב שמסביר על GraphQL בצורה נהדרת!
- https://www.howtographql.com - אתר להרחבה על הנושא של GraphQL, מביא הסברים מפורטים.
- https://portswigger.net/web-security/graphql/what-is-graphql - הסבר מ-burp suite academy על GraphQL, אחלה מקום באופן כללי ללמוד על web. הכל שם בחינם, מלמדים אותך איך להשתמש ב burp suite, הכלי שהם יצרו שהוא בגדול הכלי הבסיסי של עולם ה-web explotation.
ממליץ בחום ללמוד נושאים משם, בתור אדם שמתחיל את הדרך שלו בעולם, זה פותח את התיאבון וגם נותן בסיס מאוד טוב.
- https://portswigger.net/web-security/graphql - מסביר על חולשות ב-GraphQL, מספק גם מעבדות שדרכן אפשר ללמוד ולהתנסות.
- http://webhacking.kr:10012 - קישור ל-CTF שעשיתי שנקרא NoSQL שעוסק ב GraphQL Injection.
- http://nathanrandal.com/graphql-visualizer - אתר שמספק גרף וויזואלי לסכמה שאתה מקבל משאילתת ה-introspection.
- https://github.com/swisskyrepo/GraphQLmap - כלי לאוטומיזציה של כל העבודה עם GraphQL, מומלץ בחום.