digital whisper
Technologic Papers

פתרון אתגרי AppSec Villiage DefCon30

מאת מאור טל

הקדמה

באוגוסט 2022 השתתפתי בכנס DefCon 30 בלאס וגאס. למי שלא מכיר, מדובר באחד הכנסים המטורפים ביותר בתעשיית ההאקינג ואבטחת המידע (דמיינו קבוצה של האקרים מכל העולם מתקבצים במקום אחד, חולקים רעיונות, פורצים לכל מיני רכיבים וכו').

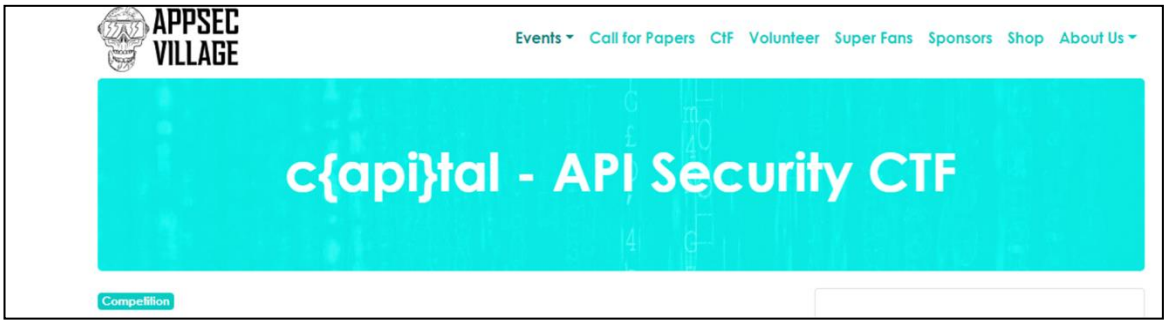
מעבר להרצאות, למסיבות, להופעות ולסדנאות, עיקר העקשן בכנס מתרכז סביב ה-Villiages הפזורים במתחם הכנס. מדובר באיזורים מוגדרים שבהם מתעסקים בנושאים ספציפיים מחוץ לתוכניה של הכנס, כגון: פריצה פיזית, הנדסה חברתית, מחקר אפליקציות, ביו-האקינג, מחקר חומרה ועוד. וכמו בכל שנה, גם השנה, במהלך הכנס התקיימו מספר תחרויות בכל אותם ה-Villiages (חברה ישראלית זכתה אפילו במקום הראשון בתחרות פריצה הרכבים השנה!).

במאמר זה אני אתאר לכם את הפתרון שלי לאתגר ה-AppSec Villiage. האתגר נכתב על-ידי חברת Checkmarx אשר התמקדה בתחום ה-API Security.

הינה אני:

1...2...3... מתכוננים

כך נראה העמוד הראשי של ה-CTF:



בתוך הווילג' עצמו (שהיה צריך לעמוד בתור שעה כדי להכנס... ראו ערך linecon) הופיע אותו השלט עם הכתובת לאתר התחרות www.capital-ctf.com (שעדיין זמין נכון לשורות אלו למשחק מחדש). המשחק עצמו מחולק למספר קטגוריות המבוססות על OWASP API TOP10 שבניגוד ל-CTF-ים אחרים, אנו נדרשים להוריד את ה-Docker שמכיל את צד-הלקוח והשרת של המשחק מקומית אלינו. ניתן היה לקבל את ההוראות להורדת ה-Dockers הרלבנטיים לטובת המשחק מהאתר עצמו:

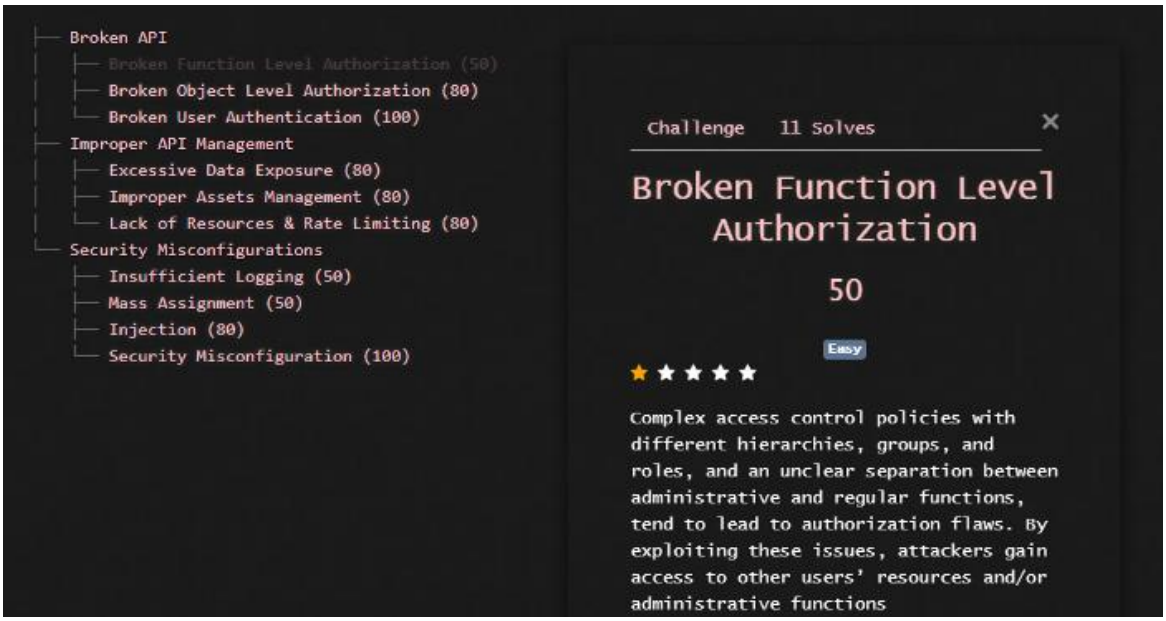
כמוכן, באותו האתר ניתן היה לקבל רשימת כלים מומלצים לטובת הפעילות לאלו שמתנסים ב-API Security בפעם הראשונה כגון: Postman, Fiddler, Burp.

כעת לאחר שהורדנו את ה-Docker מקומית והפעלנו אותו הגיע הזמן להפעלת המשחק.

כל שלב אנו נדרשים לפתור אתגר מסויים (הסדר אומנם לא חשוב) הנושא בחולשה או בעיה מסויימת ב-API של מערכת תקשורת הפוקימונים בכדי להביס את ~~צוות רוקט~~ האתגר!

שלב ראשון: Broken Function Level Authrization

בשלב הראשון של האתגר אנו מקבלים את התיאור הבא:



מה שזה אומר לנו שאנחנו צריכים למצוא מקום באפליקציה שאין שם בקרת הרשאות או גישה כנדרש, תחשבו על סיטואציה שבה נתיב מערכת השמור לממשק מניהול (לדוגמא: */admin/manageUsers*) ניתן לגישה גם על ידי משתמשים בעלי הרשאה נמוכה לאור העובדה כי לא מתקיימת בקרה או ויידוא שאכן הגורם שנגש אליה הוא מורשה לביצוע הפעולה.

אז לאחר שיטוט קצר באפליקציה, הבנתי שמדובר באפליקציה מסוג AngularJS או במילים אחרות Single Page Application אשר מכילה את כל הלוגיקה והנתיבים השונים בקוד צד-הלקוח. מנקודה זו החלטתי שהגיע הזמן לבחון את קבצי ה-JavaScript השונים במטרה למצוא אולי נתיב או מקום ש"מוסתר לעיני כל".

לבסוף בקובץ *bundle.js* ניתן היה למצוא את כלל הנתיבים של האפליקציה ובינהם משהו אחד די בלט:

```
localhost:4100/static/js/bundle.js
        columnNumber: 44
      }, this)
    }, void 0, false, {
      fileName: _jsxFileName,
      lineNumber: 76,
      columnNumber: 13
    }, this), /*#__PURE__*/(0,react_jsx_dev_runtime__WEBPACK_IMPORTED_MODULE_7__.jsxDEV)(react_router_dom__WEBPACK_IMPORTED_MODULE_8_
      path: "/article/:slug",
      element: /*#__PURE__*/(0,react_jsx_dev_runtime__WEBPACK_IMPORTED_MODULE_7__.jsxDEV)(Article, {}, void 0, false, {
        fileName: _jsxFileName,
        lineNumber: 77,
        columnNumber: 51
      }, this)
    }, void 0, false, {
      fileName: _jsxFileName,
      lineNumber: 77,
      columnNumber: 13
    }, this), /*#__PURE__*/(0,react_jsx_dev_runtime__WEBPACK_IMPORTED_MODULE_7__.jsxDEV)(react_router_dom__WEBPACK_IMPORTED_MODULE_8_
      path: "/admin",
      element: /*#__PURE__*/(0,react_jsx_dev_runtime__WEBPACK_IMPORTED_MODULE_7__.jsxDEV)(AdminPage, {}, void 0, false, {
        fileName: _jsxFileName,
        lineNumber: 78,
        columnNumber: 43
      }, this)
    }, void 0, false, {
      fileName: _jsxFileName,
      lineNumber: 78,
      columnNumber: 13
    }, this), /*#__PURE__*/(0,react_jsx_dev_runtime__WEBPACK_IMPORTED_MODULE_7__.jsxDEV)(react_router_dom__WEBPACK_IMPORTED_MODULE_8_
      path: "/logging",
      element: /*#__PURE__*/(0,react_jsx_dev_runtime__WEBPACK_IMPORTED_MODULE_7__.jsxDEV)(Logging, {}, void 0, false, {
        fileName: _jsxFileName,
        lineNumber: 79,
        columnNumber: 45
      }, this)
    }, void 0, false, {
      fileName: _jsxFileName,
      lineNumber: 79,
      columnNumber: 13
    }, this), /*#__PURE__*/(0,react_jsx_dev_runtime__WEBPACK_IMPORTED_MODULE_7__.jsxDEV)(react_router_dom__WEBPACK_IMPORTED_MODULE_8_
      path: "/settings",
      element: /*#__PURE__*/(0,react_jsx_dev_runtime__WEBPACK_IMPORTED_MODULE_7__.jsxDEV)(SettingsScreen, {}, void 0, false, {
        fileName: _jsxFileName,
        lineNumber: 80,
```

נשמע פשוט? בוא ננסה לגשת אליו כעת ישירות באמצעות BURP:

```
Request
Pretty  Raw  Hex
1 GET /api/admin HTTP/1.1
2 Host: localhost:8000
3 sec-ch-ua: "Chromium";v="103", ".Not/A)Brand";v="99"
4 authorization: Token
  eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzIiNiJ9.eyJleCVybmFtZSI6ImRoMXxvT2F0aWFuIiwiZXhwIjoxNjYx
  MDUyMzMwLCJzdWIiOiJhY2NlcMifQ.ukFVTLLZZly0pwzlwampRBFGAlZQxBz7cv3_iIHWGC0
5 sec-ch-ua-mobile: ?0
6 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like
  Gecko) Chrome/103.0.5060.134 Safari/537.36
7 sec-ch-ua-platform: "Windows"
8 Accept: */*
9 Origin: http://localhost:4100
10 Sec-Fetch-Site: same-site
11 Sec-Fetch-Mode: cors
12 Sec-Fetch-Dest: empty
13 Referer: http://localhost:4100/
14 Accept-Encoding: gzip, deflate
15 Accept-Language: en-GB,en-US;q=0.9,en;q=0.8
16 Connection: close
17
18

Response
Pretty  Raw  Hex  Render
1 HTTP/1.1 200 OK
2 date: Sun, 14 Aug 2022 03:45:08 GMT
3 server: uvicorn
4 content-length: 357
5 content-type: application/json
6 access-control-allow-origin: *
7 access-control-allow-credentials: true
8 Connection: close
9
10 {
    "flag":"flag{RF54_I_aM_Th3_aDmin_H3r3!}",
    "description":
    "Complex access control policies with different hierarchies, groups, and roles, and
     an unclear separation between administrative and regular functions, tend to lead t
    o authorization flaws. By exploiting these issues, attackers gain access to other u
    serâ€™ resources and/or administrative functions."
  }
```

[מתנצל על האיכות הגרועה, של תמונה זו ושל שאר התמונות במאמר שבאיכות ירודה. צילום המסך נלקח בלייב ואין לי אפשרות לצלם שוב]

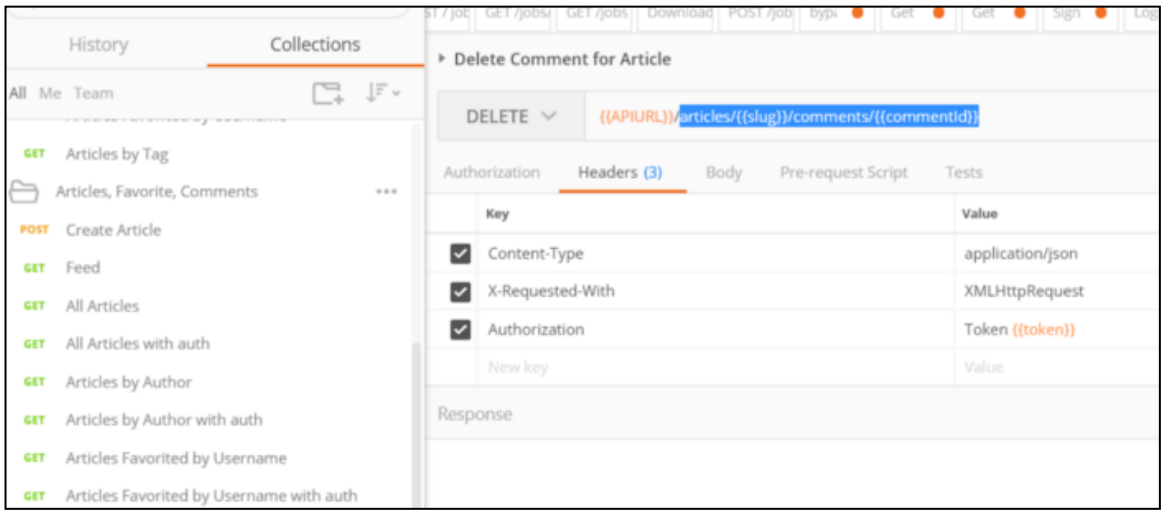
וכך השגנו את הדגל הראשון שלנו ☺

שלב שני: Broken Function Level Authrization

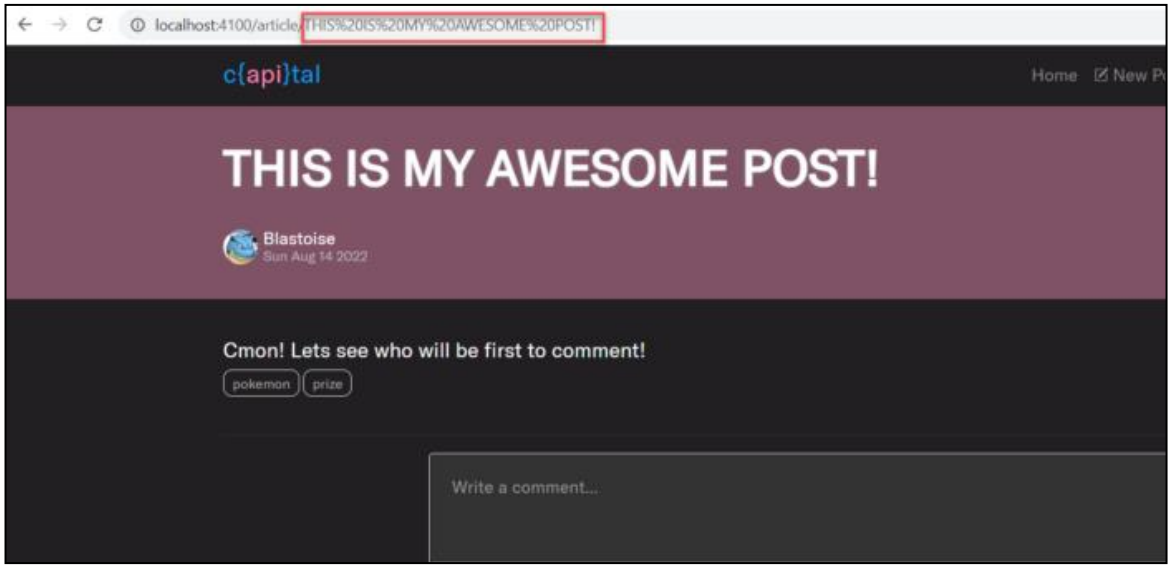
בחולשות מסוג BOLA (Broken Function Level Authrization) שבניגוד ל-Broken Function Level Authrization כמו מקודם, כאן אנו צריכים לחפש "פונקציה" (נקרא גם API Call) מסויים במערכת שניתן לבצע בו מעקף הרשאות ללא מנגנוני בקרה שונים. לרוב כשנרצה לחפש חולשות מסוג זה, נרצה להתמקד ב-API Calls שמעבירים מידע מזהה בשורת הכתובת (לדוגמא: GET /users/<id>/info) בניסיון לשנות את הפרמטרים הנשלחים בכדי לצפות במידע שמחוץ לתחום ההרשאה של המשתמש הנוכחי שלנו.

אחד מהדרכים היעילות לבדוק בהם בארגונים גדולים חשיפה לחולשה הזו הוא לעבור על הדוקומנטציה (כגון Postman או Swagger) שלהם ובכך להתמקד ב-API מסוימים לטובת הבדיקות, כמובן שלא כולם מסייעים ואז צריכים לבצע אינומרציה ידנית... אך אנחנו ב-CTF לימודי אז זה קצת שונה ☺

מאחר וקיים ברשותנו ה-Postman אז החלטתי לפתוח את התיעוד ולחפש שם... עד שמצאתי את הממשק API שנקרא Delete Comment for Article שנשמע מעניין:



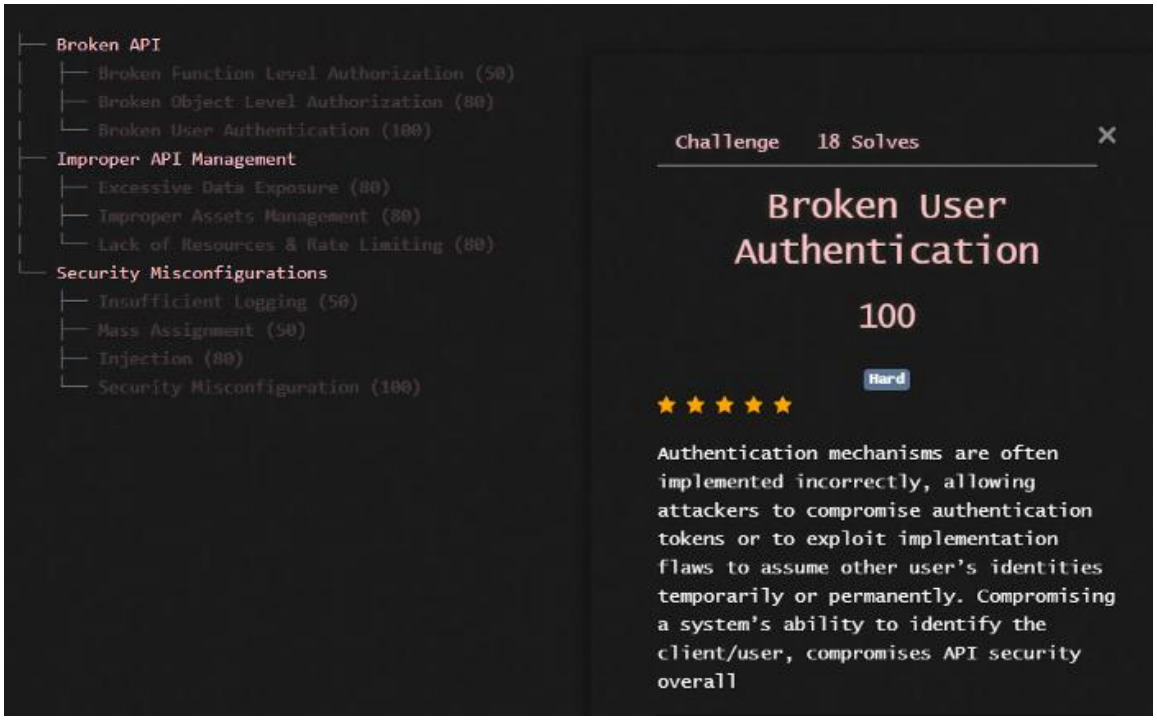
על בסיס התיעוד נראה שהערך הראשון שמועבר בשורת הכתובת הוא Article Slug אז החלטתי לשוטט קצת באתר ולחפש רעיונות לערך המסויים הזה. לבסוף ניתן היה לראות שבכל כניסה לעמוד מאמר מסויים במערכת מועבר בשורת הכתובת שם לפוסט, אולי זה ה-Article Slug שאנו מחפשים?



כעת ננסה לגשת לממשק ולמחוק את הפוסט כמשתמש רגיל עם הערך של מה שמצאנו בשורת הכתובת:

ונקבל את הדגל השני!

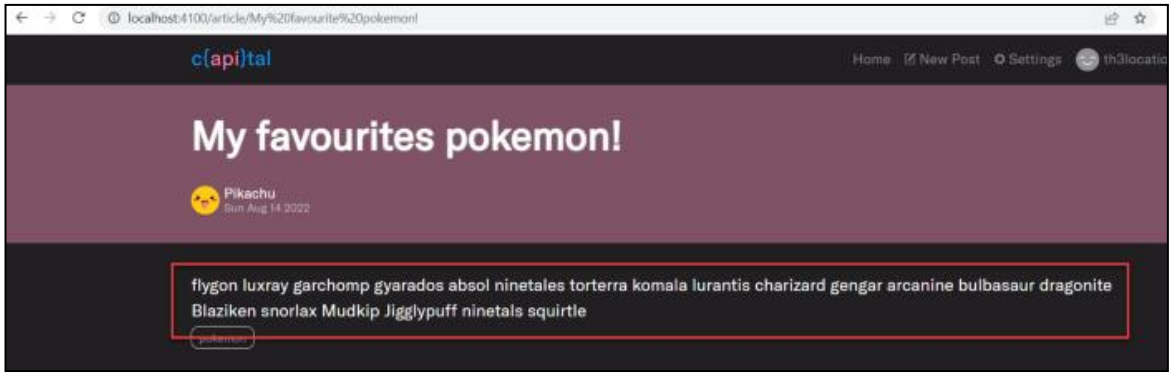
שלב שלישי: Broken User Authentication



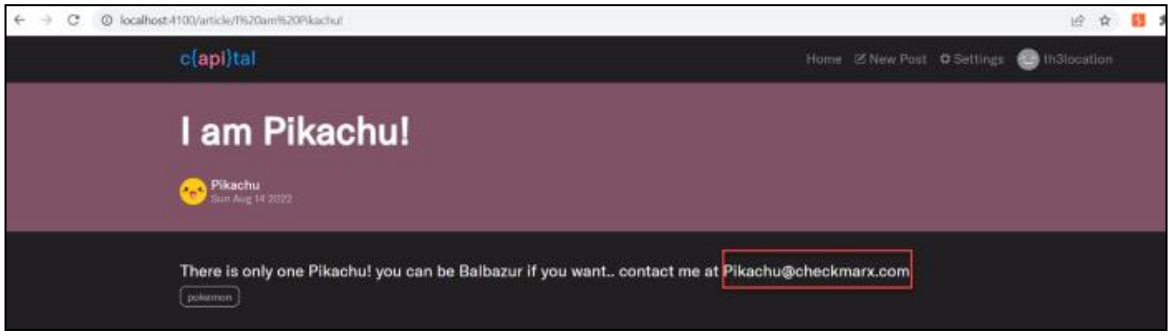
במתקפות מסוג Broken User Authetication אנו מתייחסים בעיקר לתהליכים הקשורים למנגנון ההזדהות העלול לסייע לנו להשיג גישה לחשבונו, כגון שימוש בטכניקות של Brute Force או Password Spray וזאת במטרה להשיג אחיזה ראשונית על משתמש קיים במערכת.

אך בכדי לבצע את המתקפה בצורה יעילה, נצטרך לבצע OSINT בסיסי ("איסוף מודיעין") על מנת על המשתמשים במערכת בכדי לראות האם קיימים נתונים שנוכל להעזר בהם לטובת מתקפות של ניחוש סיסמאות.

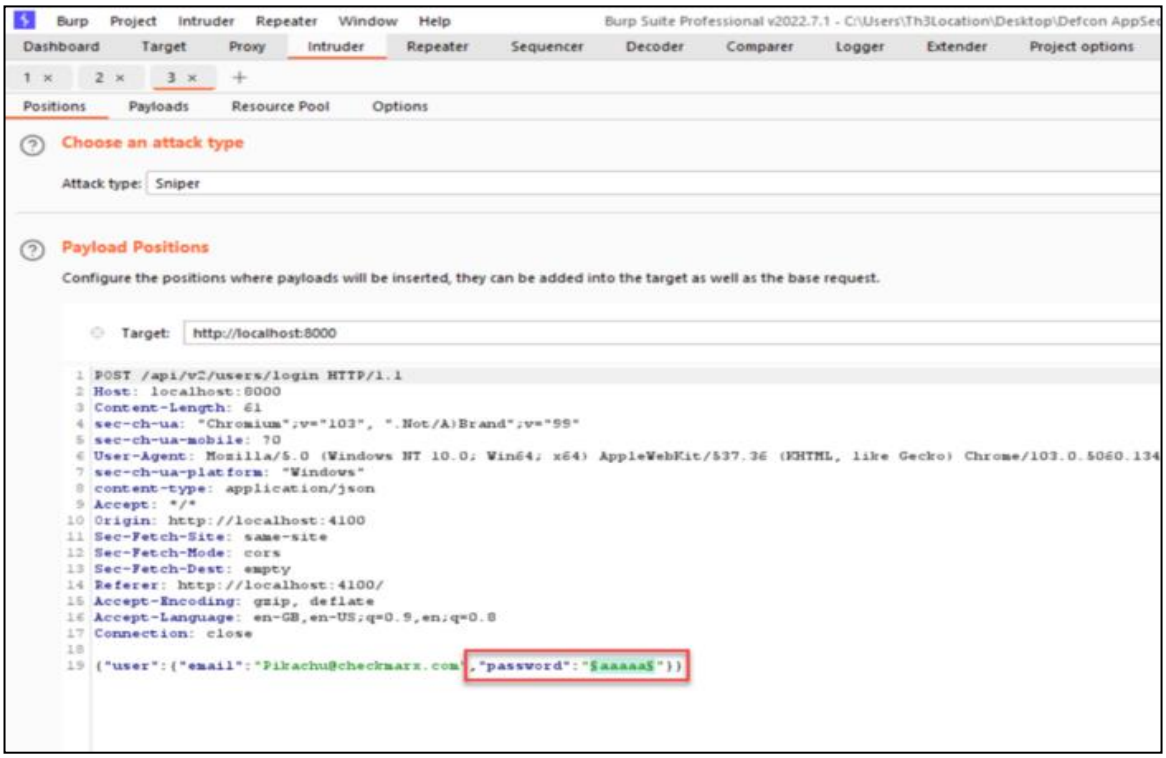
מאחר וכל האפליקציה בסגנון "פוקימונים" החלטתי ללבוש את המצחייה האדומה ולנסות ללכוד כמה בעצמי... לאחר שיטוט קצרצר מצאתי שפיקאצ'ו השובב מתרברב לו ברשימת החברים שלו בינהם ניתן למצוא סלבים כמו גיריוס, דרגונייט (זה פוקימון?) וסקווירטל!



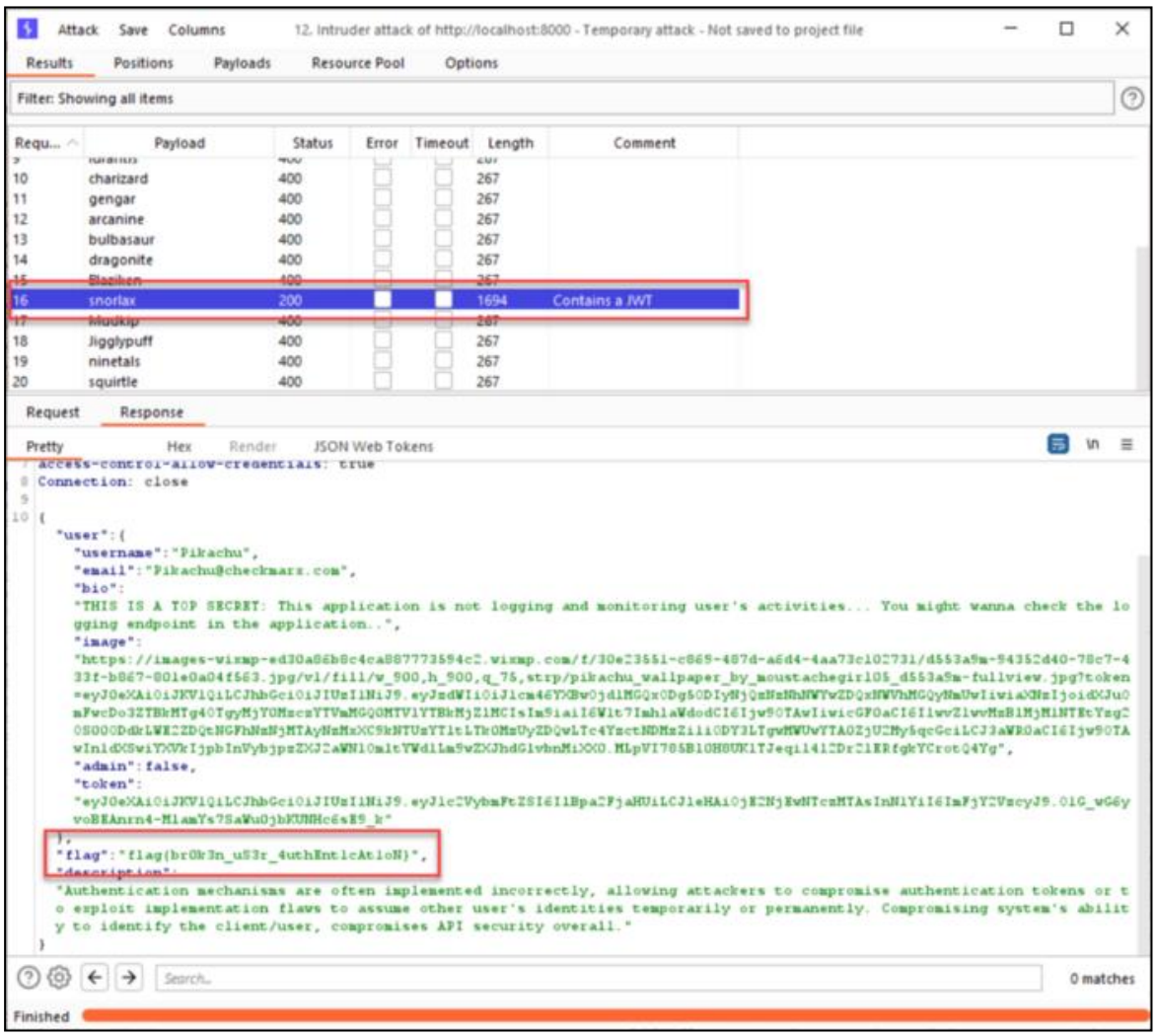
נראה שאנחנו הולכים בנתיב הנכון... אם נמשיך לחפש נוכל לראות שפיקאצ'ו אפילו הועיל בטובו לפרסם את כתובת המייל שלו למעריצים וגרופים אחרים:



מנקודה זו, יש ברשותנו רשימה של פוקימונים וכתובת המייל של פיקאצ'ו. בוא נטען אותם ל-Burp Intruder על ה-API של ההתחברות, כאשר שדה שם המשתמש יהיה המייל והסיסמא יהיה הערך המשתנה של אחד משם הפוקימונים:

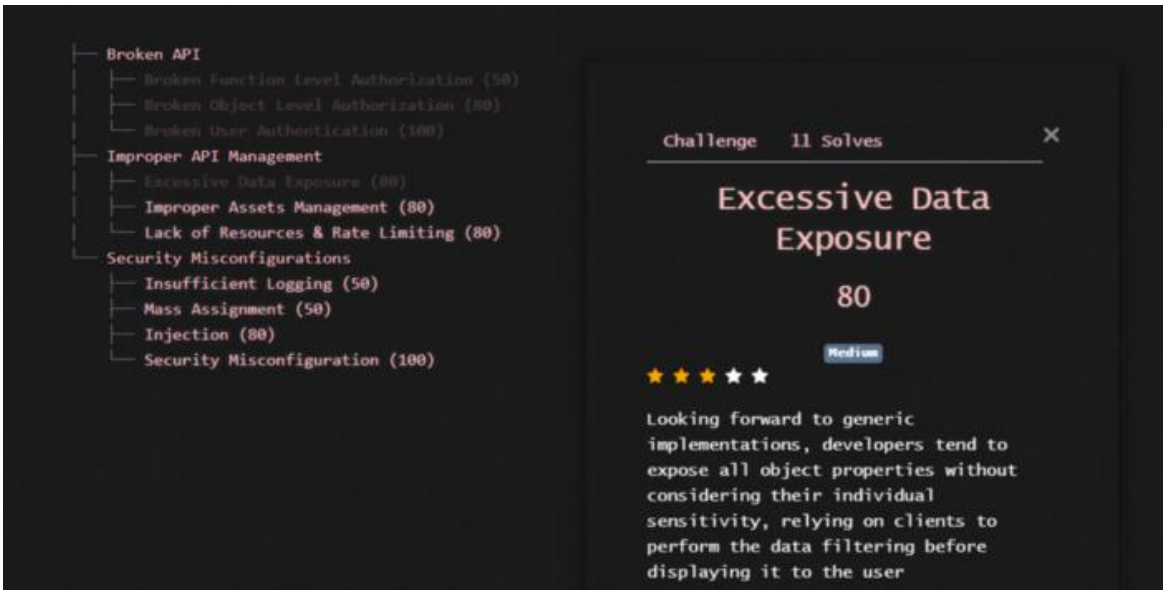


לבסוף, נקבל את הסיסמא שהיא סנורלקס יחד עם הדגל השלישי שלנו:



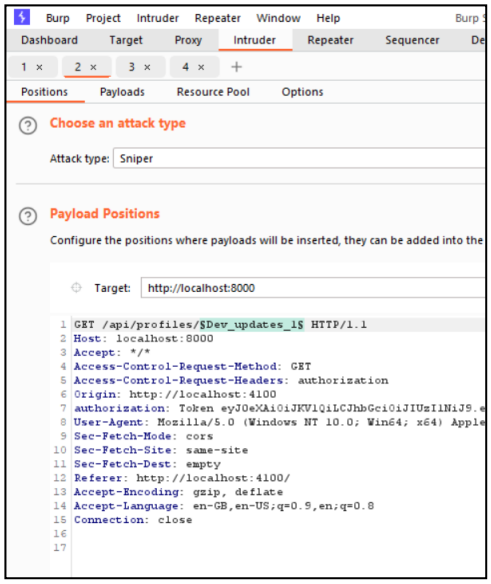
שלב רביעי: Excessive Data Exposure

כעת יש לנו אחיזה על החשבון של פיקאצ'ו ואנחנו ממשיכים לשלב הבא:

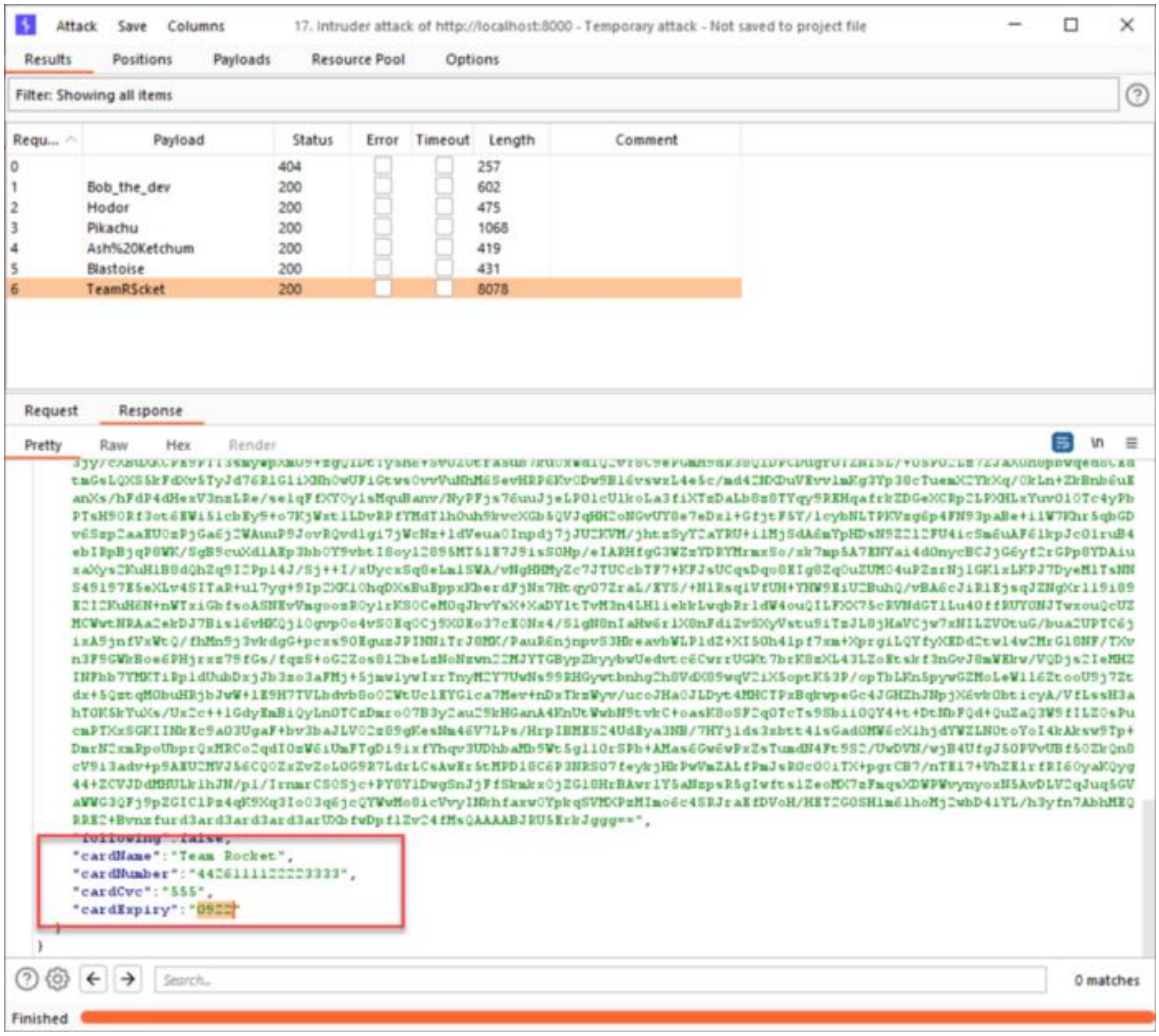


ב-Excessive Data Exposure אנו לרוב מחפשים קריאות שמאפשרות לנו לצפות במידע של משתמשים אחרים, בתקווה למידע רגיש או שלא אמור להיות פומבי (לדוגמא: ת.ז, כרטיס אשראי וכ'ו...). מנסיון לא מעט, לרוב נתקלים אצל לקוחות במקרה שבו שכבת התצוגה (כגון VueJS, AngularJS, ReactJS וכ'ו...) לא מציגה את כל המידע שמתקבל מה-API ובכך נוצרת "הגנה" שאמורה ל"מנוע גישה" לאנשים שלא מורשים לצפות בנתונים אלו... זה בערך כמו לתת לפקיד לצפות בשם פרטי ומשפחה של הלקוח אבל ברגע שהוא יגש ל-API ישירות הוא יראה שמוחזר גם תאריך לידה, ת.ז ומספר מלא של כרטיס אשראי...

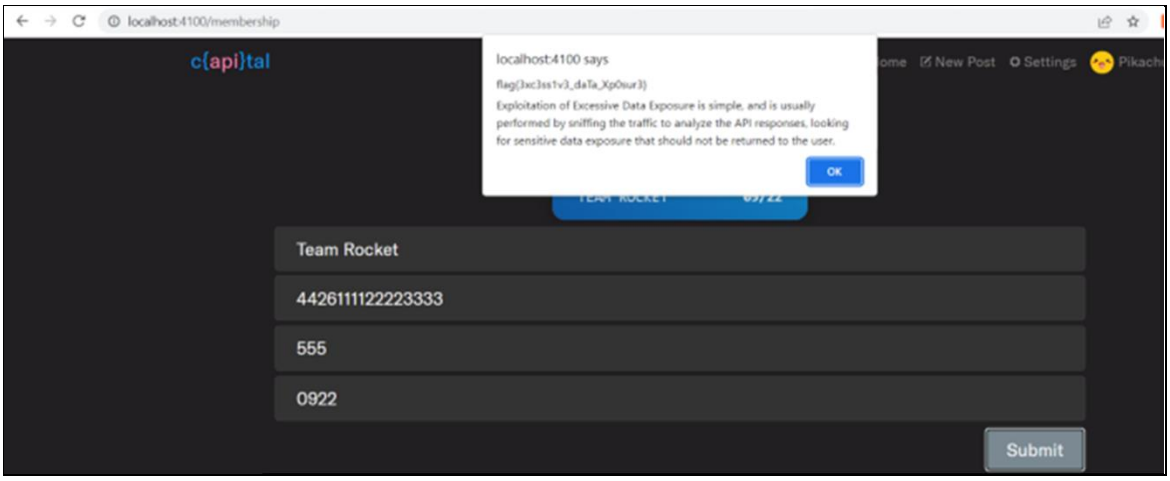
אחד מהדברים הראשונים שנוכל לבצע הוא לבצע מיפוי של כל הפרופילים במערכת ולראות האם ה-API הזה מחזיר נתונים שהוא לא אמור להחזיר לגבי פרופיל מסויים:



לבסוף נחשו את מי מצאנו? צוות רוקט (מייאו בדיוק)! המספר כרטיס אשראי שלהם מעודכן במערכת והוא חוזר כחלק מהתשובה של הבקשה:

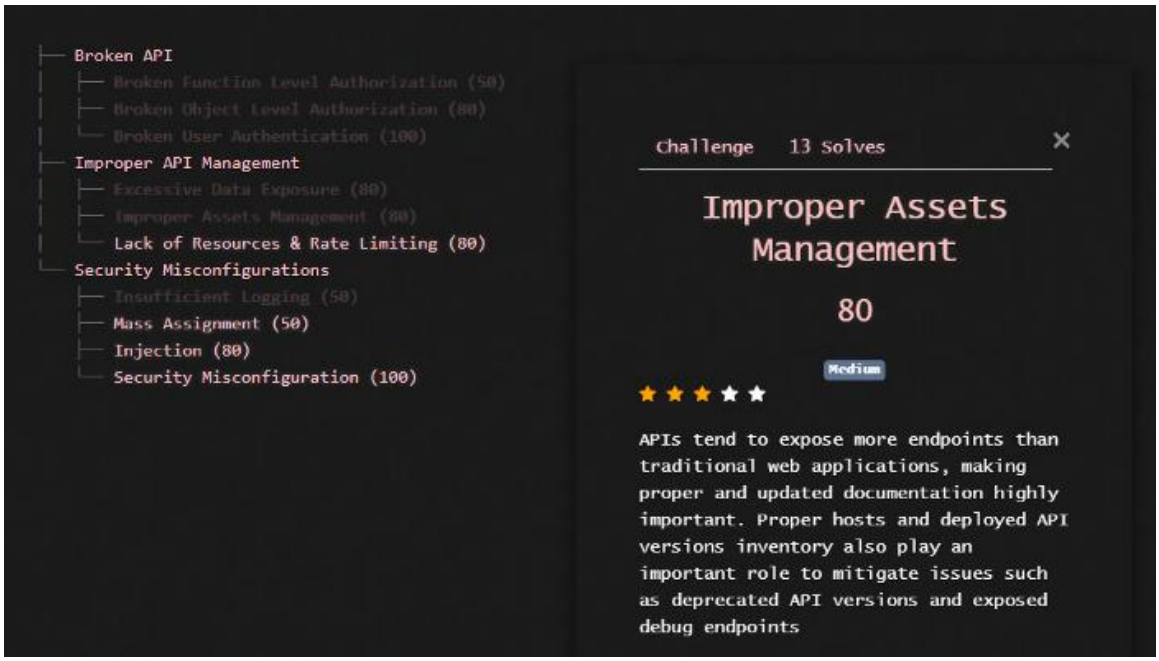


הגיע רגע הנקמה שפיקאצ'ו חיכה כבר 20 עונות וזה לנסות להשתמש באשראי שלהם שבתקווה יחזיר לנו את הדגל שאנחנו צריכים:



והצלחנו ☺

שלב חמישי: Improper Assets Mangement



אז מה זה בעצם Improper Assets Mangement? מפתחים רבים נוטים לעלות גרסאות שונות של ה-API לייצור תוך כדי שהם עובדים על שינויים או פיטצ'רים חדשים ובכך להפריד אותם בסימוני גרסאות שונים (כגון .../v2/.../v1/) ובאופן כך מאפשרים לנו כתוקפים למפות חולשות שהיו קיימות בגרסאות ישנות ובכך להרחיב את משטח התקיפה עבור פיתוחים שככל הנראה תוקנו בגרסאות קודמות של ה-API מולו אנחנו עובדים.

לכן אם נשים לב טוב לגרסא של ה-API באתגר שלנו נוכל להבחין שמדובר על "/api/v2/":

Request

Pretty Raw Hex

```
1  POST /api/v2/users/login HTTP/1.1
2  Host: localhost:8000
3  Content-Length: 63
4  sec-ch-ua: "Chromium";v="103", ".Not/A)Brand";v="99"
5  sec-ch-ua-mobile: ?0
6  User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537
   Gecko) Chrome/103.0.5060.134 Safari/537.36
7  sec-ch-ua-platform: "Windows"
8  content-type: application/json
9  Accept: */*
10 Origin: http://localhost:4100
11 Sec-Fetch-Site: same-site
12 Sec-Fetch-Mode: cors
13 Sec-Fetch-Dest: empty
14 Referer: http://localhost:4100/
15 Accept-Encoding: gzip, deflate
16 Accept-Language: en-GB,en-US;q=0.9,en;q=0.8
17 Connection: close
18
19 {
     "user":{
       "email":"Pikachu@checkmarx.com",
       "password":"snorlax"
     }
   }
```

כעת ננסה לשנות את הערך מ-""/v2/ ל-"/v1/" ונראה מה נקבל:

Request

Pretty Raw Hex

```
1  POST /api/v1/users/login HTTP/1.1
2  Host: localhost:8000
3  Content-Length: 63
4  sec-ch-ua: "Chromium";v="103", ".Not/A)Brand";v="99"
5  sec-ch-ua-mobile: ?0
6  User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like
   Gecko) Chrome/103.0.5060.134 Safari/537.36
7  sec-ch-ua-platform: "Windows"
8  content-type: application/json
9  Accept: */*
10 Origin: http://localhost:4100
11 Sec-Fetch-Site: same-site
12 Sec-Fetch-Mode: cors
13 Sec-Fetch-Dest: empty
14 Referer: http://localhost:4100/
15 Accept-Encoding: gzip, deflate
16 Accept-Language: en-GB,en-US;q=0.9,en;q=0.8
17 Connection: close
18
19 {
     "user":{
       "email":"Pikachu@checkmarx.com",
       "password":"snorlax"
     }
   }
```

Response

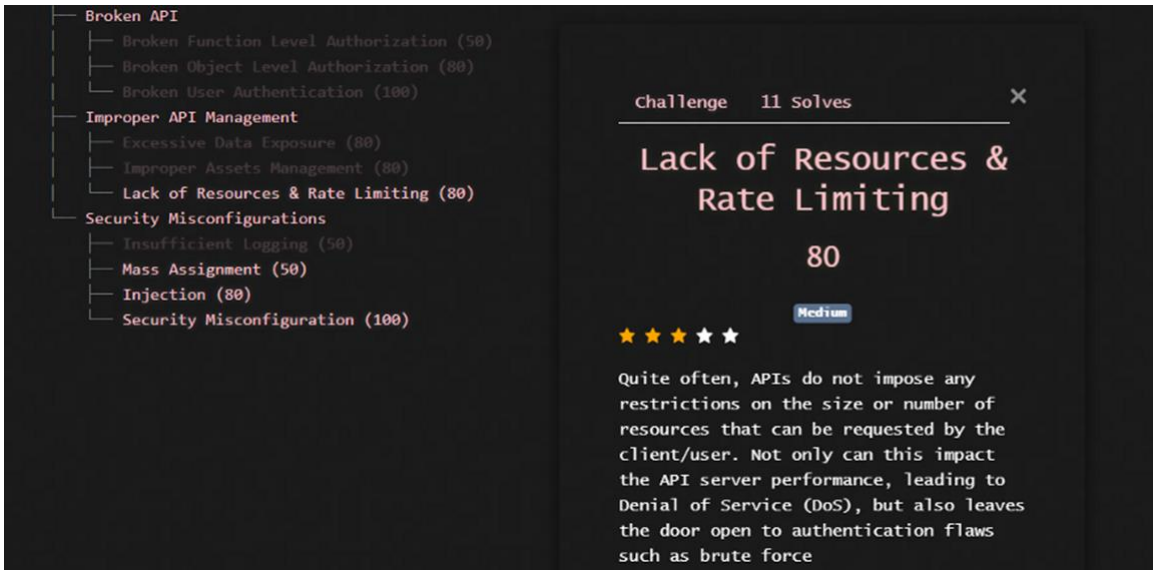
Pretty Raw Hex Render

```
1  HTTP/1.1 200 OK
2  date: Sun, 14 Aug 2022 05:24:06 GMT
3  server: uvicorn
4  content-length: 1084
5  content-type: application/json
6  access-control-allow-origin: *
7  access-control-allow-credentials: true
8  Connection: close
9
10 {
     "user":{
       "username":"Pikachu",
       "email":"Pikachu@checkmarx.com",
       "bio":"",
       "image":
       "https://images-wixmp-ed30a86b8c4ca887773594c2.wixmp.com/f/30a2b4b1-c869-487d-a6d
       4-4aa73c107731/d553afm-943b2d40-70c7-433f-b867-001e0a04f463.jpg/v1/fill/w_800,h_5
       00,q_75,strp/pikachu_wallpaper_by_mousetachegirl05_d553afm-fullview.jpg?token=eyJ0
       eXAiOiJKV1QiLCJhbGciOiJIUzIiNiJ9.eyJzdWIiOiJlcmk6TXBwOjdlMGQzODg4ODIyMjZzNzMzNWFVw
       ZDQzNWVhMGQyNmUwIiwiaXNzIjoidXJuOmFwcDo3ZTBkMTg40TgyMjYGMmcsYTVmMGQONTVlYTRkMjZlN
       CIsIm9iaiI6WiteTIabIaWdodCI6IjwO0TAwIiwicGFGaCI6IlwvZlwvMzNlMjMiNTRtYzg2OGQ0DdrlW
       EIZDqtHGFmNmNjNTAyNmNmNCMzNTUwYTItlTbUNmOyZDQwLTc4YmctMDMzZiliODY3LTgwNWUwYTA0ZjU
       IMy5qcGciLCJ3aWR0aCI6IjwO0TAwInldOSwiYXVkIjpbInVybjpzZDZIaWN10mlnYWdlLmfwZGZhdGlv
       bmNlOOO.MlgVI786B1GHOUKi7Jeqii4iZDr2iRBfgRYCrot24Yg",
       "admin":false,
       "token":"Keep it up!"
     },
     "flag":"flag{ImproperAssets_ManAgMent}",
     "description":
     "Old API versions are usually unpatched and are an easy way to compromise systems w
     ithout having to fight state-of-the-art security mechanisms, which might be in plac
     e to protect the most recent API versions. "
   }
```

והנה בעת השינוי של גרסאת ה-API בממשק ההתחברות נקבל גם את הדגל לשלב שלנו!

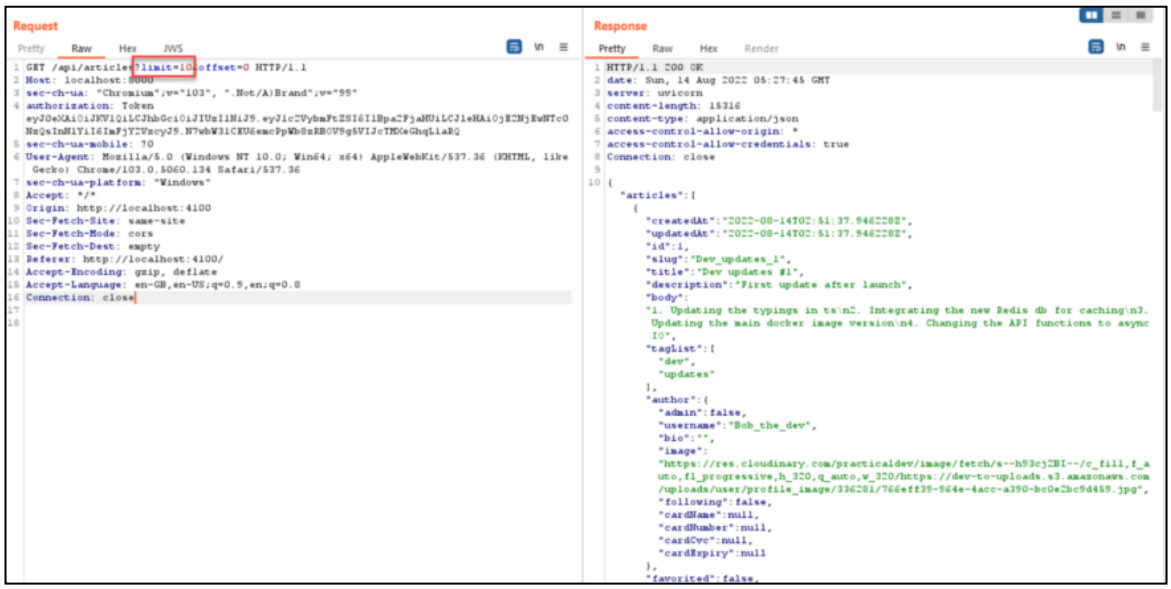
שלב שישי: Lack of Resources & Rate Limiting

כעת הגענו למשימה האחרונה בנושא Improper API Mangement:

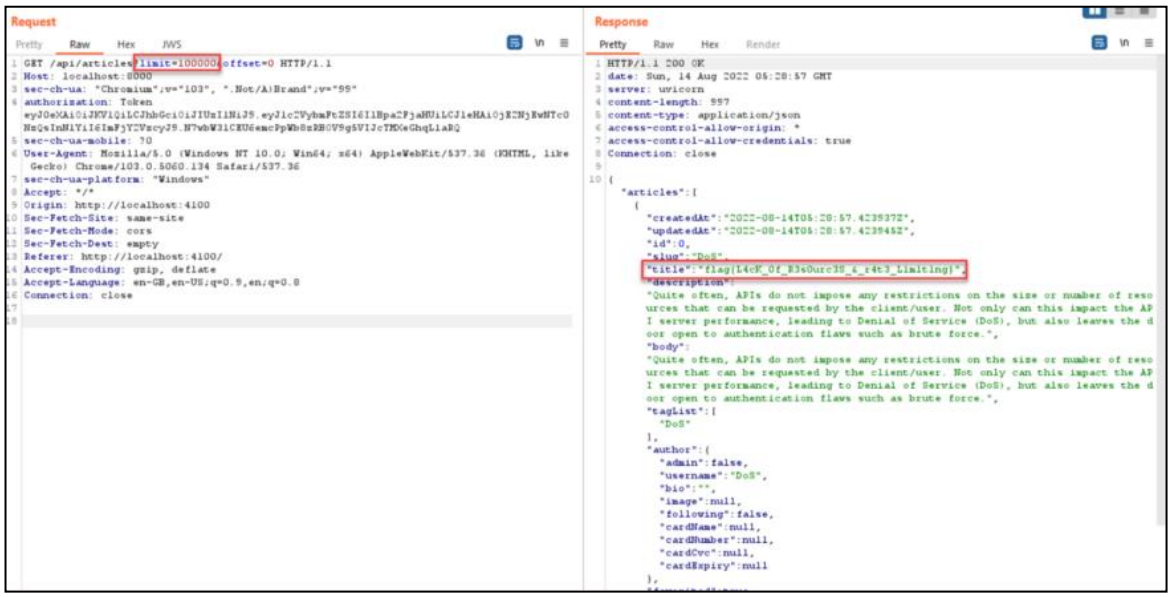


ב-Lack of Resource & Rate Limiting אנו מתייחסים לחולשות אשר נגרמות בעקבות ריבוי בקשות או ביצוע עומס של עיבוד נתונים בצד-השרת אשר עלול לגרום למניעת שירות (DoS). דוגמאות קלאסיות ניתן למנות כמו Regex, ביצוע בקשות מרובות לממשקים עיבוד נתונים וקבצים בצד-השרת או אפילו חשיפות קלאסיות כגון Internger Buffer Overflow.

לכן כשמחשפים נקודת ניצול ראשונית חשוב למפות כמה שיותר פרמטרים וערכים שונים בבקשות אשר יכולים להיות חשופים לאחד או יותר מהתרחישים האלו. לאחר סיור באפליקציה ומיפוי כמה בקשות אפשריות באמצעות Postman נמצא API שמציג את כל המאמרים בבלוג. אך מה שמעניין באמת לגביו הוא העובדה שהוא מגביל את כמות הנתונים שנשלפים ל-10 רשומות בלבד בכל בקשה:

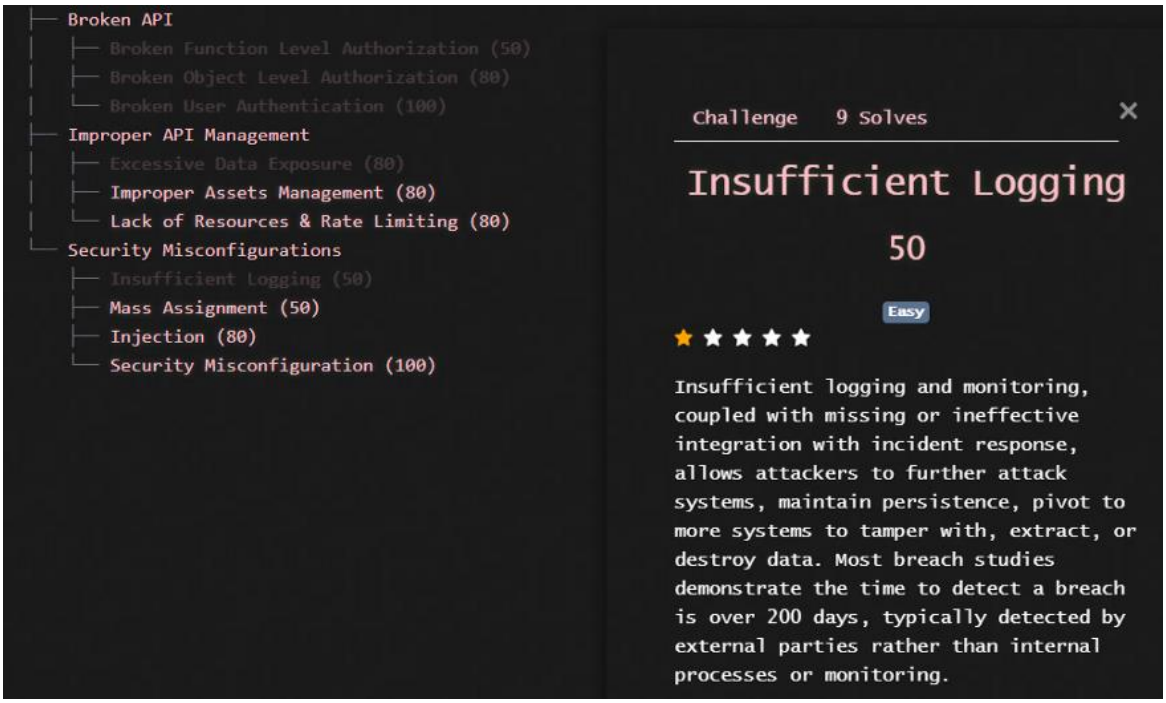


האסטרטגיה עכשיו תיהיה לנסות לגרום לצד-השרת לבצע שאילתא כבדה מאחורי הקלעים בכך שתנסה לשלוף יותר מ-100 אלף רשומות, מה שבתקווה עלול לנצל משאבים רבים אשר עלולים לגרום ל-DOS כלשהוא. לאחר שנעלה את הערך של הפרמטר limit נקבל את הדגל:



מה שלבסוף יחזיר לנו את הדגל שלנו ☺

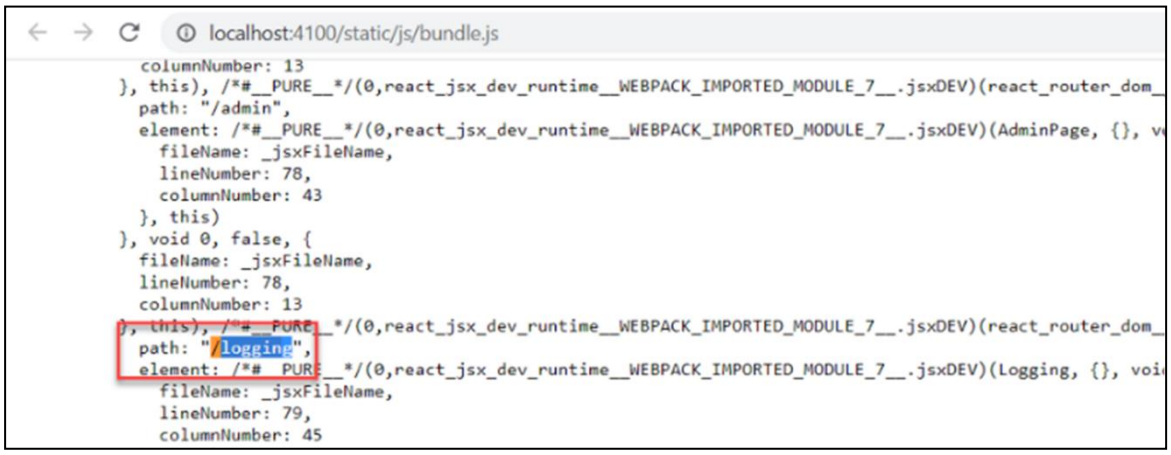
שלב שביעי: Insuffianct Logging



ב-Insuffienact Logging אנו לרוב נחפש ממשקי API אשר אחראים לתיעוד פעולות במהלך מחזור חיי האפליקציה. הסיבה היא שבמידה ואנחנו יכולים להשפיע על אופן תיעוד הלוגים באפליקציה אנחנו בעצם יכולים ל"הסתיר" את הפעולות שלנו כתוקפים ובכך להקשות על פעולות התחקור של הצוות פיתוח בצד השני. ☺

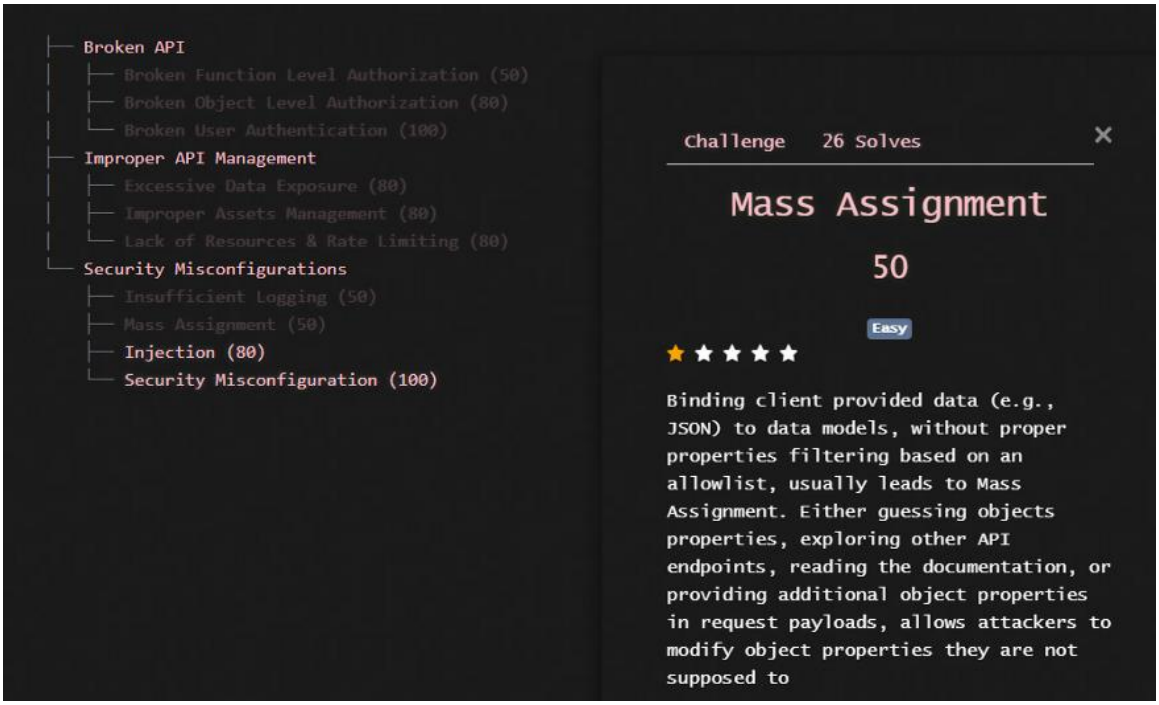
במקרה שלנו כשננסה לחפש ממשק API כלשהוא שקשור ללוגים או לתיעוד לא נמצא לכך אזכור ב-Burp (למשל דרך Target->Scope שממפה עבורנו את כל ה-Endpoints שעברנו), כמוכן גם ברגע שנבדוק את ה-Postman או ה-Swagger שסופק לנו אנחנו לא נמצא לשם אזכור... איך מתקדמים מכאן?

זוכרים את מה ששוחנו עליו בחשיפה הראשונה? בסופו של יום האפליקציה שומרת את כל הנתיבים שלה בצד-הלקוח! אז ניתן מבט חטוף שוב ל-bundle.js שלנו:



מעולה! אז כעת מצאנו שקיים API בשם logging/ אז בוא ננסה לפנות אליו ישירות, ננסה באמצעות POST ו-GET עד שנגלה שמדובר ב-GET שיחזיר לנו את הדגל שלנו:

שלב שמיני: Mass Assigment



מתקפת Mass Assigment (הנקראת גם AutoBinding עבור Spring MVC, .NET MVC ו-PHP Object Injection) היא מתקפה אשר מתייחסת למצב בו המפתחים מקבלים את הערכים מצד-הלקוח בבקשה כאובייקט ללא ביצוע בקרת קלט על הערכים או אימות של סוגי הנתונים אשר השרת מצפה לקבל ובכך מאפשרת לתוקף לדרוס ערכים מסויימים בצד-השרת באופן שהאפליקציה לא מצפה.

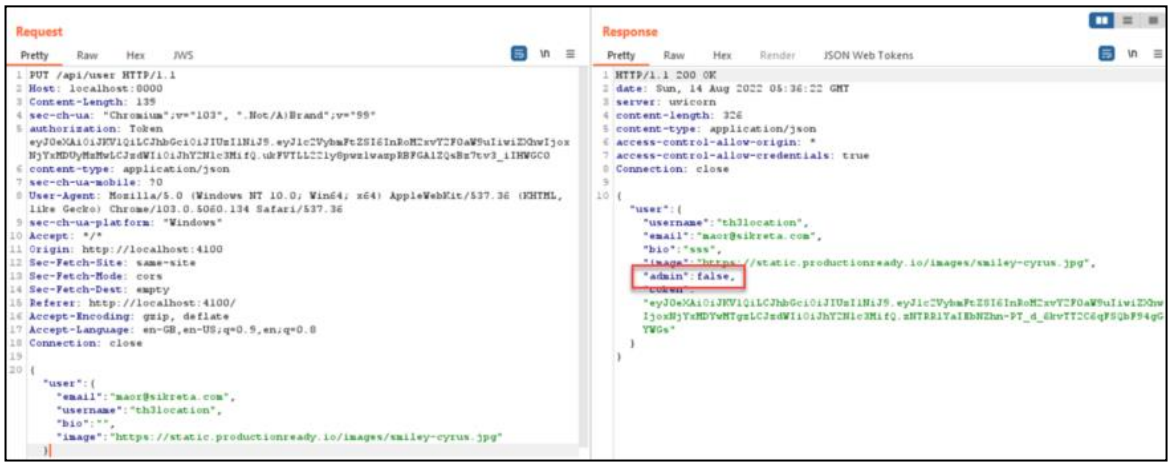
לדוגמא, נניח ובצד-לקוח קיים טופס התחברות עם הקוד הבא:

```
<form>
        <input name="username" type="text">
        <input name="password" type="text">
        <input type="submit">
</form>
```

וקוד צד השרת בנוי מאובייקט אשיר מכיל את הערכים: username, password, isAdmin.

אבל רגע! אין לנו בטופס isAdmin נכון? במילים אחרות, אם נוכל לקבל מושג / לנחש שהערך isAdmin אכן קיים בצד-השרת נוכל להוסיף אותו כפרמטר נוסף לבקשה שלנו (למרות שבפועל הבקשה מיועדת להעביר רק שם משתמש וסיסמא בלבד) מה שיגרום לצד-השרת לעבד את הערך הזה גם ולעבד אותו!

בחזרה לאתגר, אם נסתכל על הבקשת התחברות של האפליקציה נוכל לראות שבצד-שרת קיים הערך admin המציין אם מדובר במשתמש מנהל או לא:

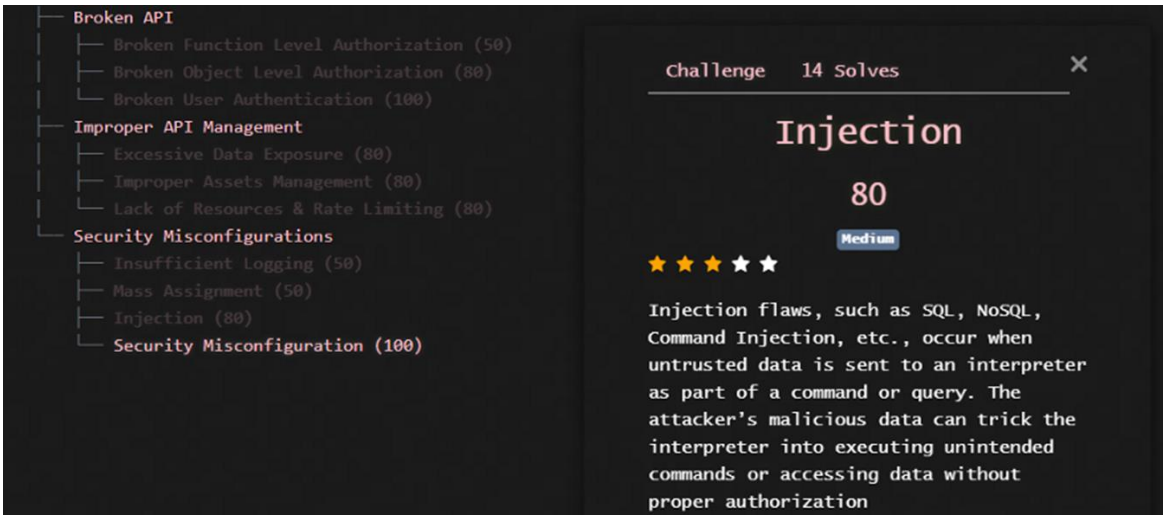


האם נוכל לדרוס את הערך admin? לצורך כך ננסה כעת לשלוח את הפרמטר admin: true כחלק מה-JSON של הבקשה המקורית:

```
{
    "user": {
            "email": "maor@sikreta.com"
            "username": "th3location",
            "bio":"",
            "image": "https://static.productionready.io/images/smily-cyrus.jpg",
            "admin": true,
        }
}
```

ולבסוף נקבל את הדגל שלנו ☺

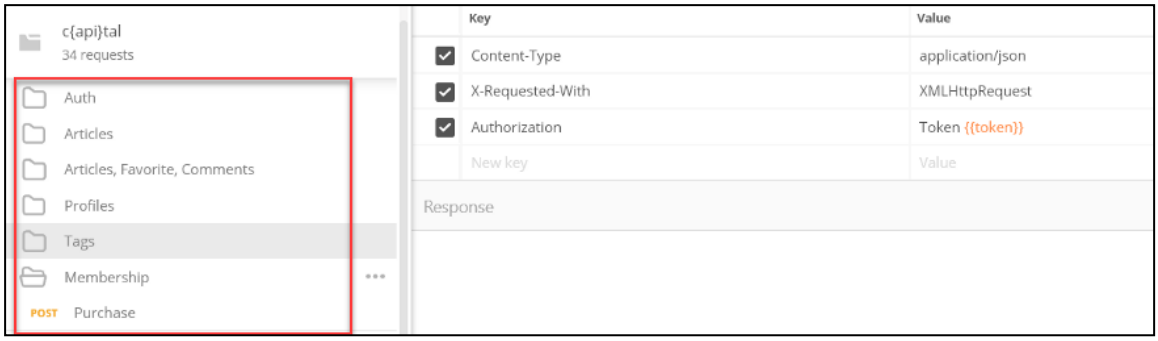
שלב תשיעי: Injection



במתקפת Injection עלינו למצוא מקום שבו אחד או יותר מהפרמטרים הנשלחים בבקשה לשרת ה-API חשוף למתקפות מסוגי הזרקות כגון GraphSQL, SQL Injeciton, Cross-Site Scripting, NoSQL וכו'.

במהלך האתגר, כדי למפות את כל נקודת הכניסה האפשריות נעבור על כל התיעוד הקיים לנו (הן ב-Postman והן ב-Swagger). כאשר נבחן את שניהם נוכל לגלות שקיימים לנו "Shadow Methods" או במילים אחרות קריאות API שלא מצאנו במהלך השיטוט הרגיל באפליקציה.

שימו לב לגבי התיעוד הרגיל ב-Postman:



ועכשיו לגבי ה-Swagger שחושף לנו מידע אודות קריאת API בשם debug:

ממעבר על התיעוד של הפונקציה נראה כי היא מאפשרת הרצת קוד. ננסה להשמיש אותה:

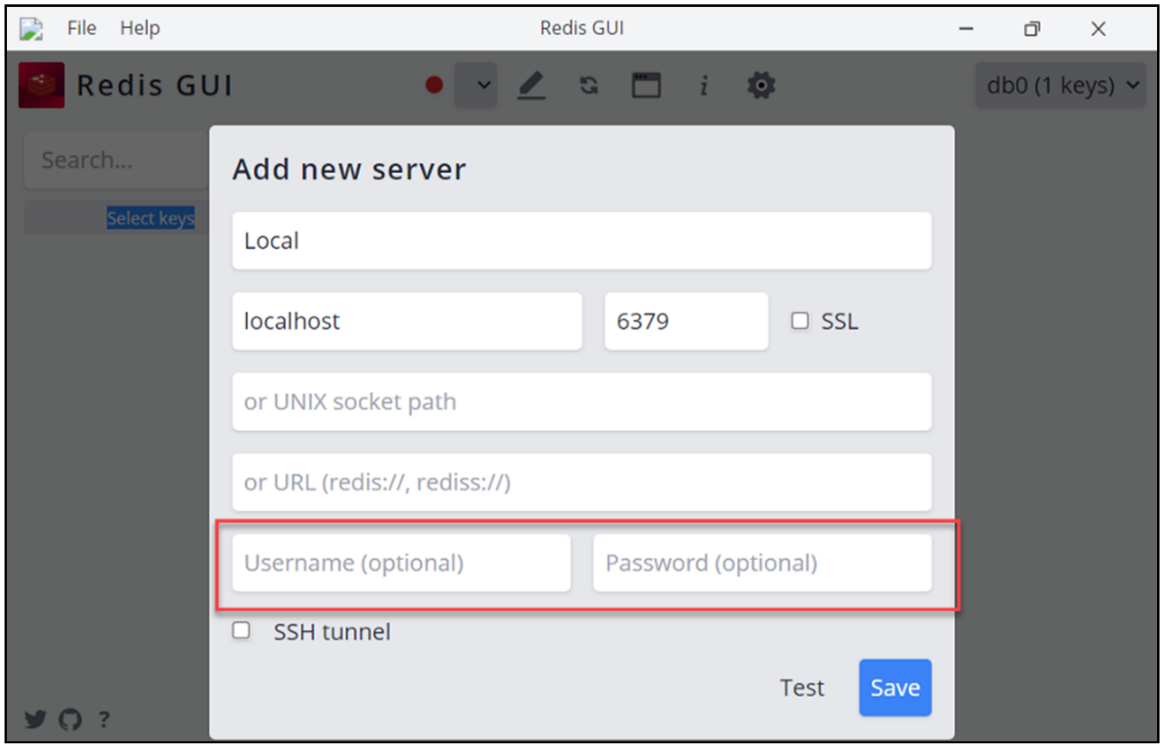
ניכר כי קיים בקרת קלט מסויימת והרצה מוגבלת לפקודה ספציפית בלבד - *uptime* אז ננסה אותה הפעם אבל עם ה-Payload הבא: *uptime && dir* בתור ניסיון ראשון:

הקוד רץ בצד-השרת בהצלחה ! כעת יש לנו את הדגל שלנו.

שלב עשירי ואחרון: Security Misconfigurtion

ב-Security Misconfigurtion אנו מתייחסים לכל מה שקשור להגדרות לקויות, חלקיות בשרת או באפליקציה אשר עלולות לחשוף לנו משטח תקיפה תשתיתי (כגון: שרות נוסף חשוף, פורט נוסף וכ'ו...). דוגמא קלאסית היא למשל שרת API שחושף את הטכנולוגיה שהוא מריץ דרך ה-Headers שלו כמו X-Powered-By ו-Server.

מאחר ותהליך התקנת ה-Dockers בוצע מהצד שלנו, נזכרתי שהמסד נתונים בצד-השרת היה מבוסס Redis. אז החלטתי לבדוק האם השרת מסד נתונים מגדיר את הפורט שלו חשוף, כדי לבצע את זה הורדתי את ה-Redis GUI (תוכנה חינמית) למחשב וניסיתי בפעם הראשונה ללא שם משתמש וסיסמא:



ו... התחברנו!

כעת מה שנותר זה לבדוק את תוכן המסד נתונים בכדי להשיג את הדגל האחרון:

לסיכום

אין ספק שבין כל האתגרים שהיו במהלך הכנס, זה היה אתגר לימודי, כיפי ומעניין שהיה פתוח למגוון רחב של משתתפים בכל הרמות. הנושא של API Security הוא חם מאוד בתקופה הזו ולכן אתגרים מסוג זה הם ממש יעילים כדי לקדם את המודעות וההכרות בתחום גם לאלו שמתחילים את צעדיהם הראשונים או כאלו שעדין חושבים שהמילה API Security זה מילה מסובכת ☺

קחו בחשבון שאם אתם מתכננים לבצע את המתקפות שהוזכרו כאן באתר הלקוח או כחלק מתהליך Bug bounty חשוב לבצע שימוש בבדיקות ידניות (כמו שהודגם כאן) וגם בכלי אוטמציה שונים לכיסוי מיטבי (למשל באמצעות Authrize ו-Burp Target), ללמוד את התיעוד של הלקוח ובעיקר להבין את התהליכים העסקיים השונים בכדי להצליח במשימה שלנו להיות מאסטרי API Security! בהצלחה!

על המחבר

מאור טל הוא בודק חוסן וחוקר סייבר (מוסמך OSCP ו-CCSK), עם ניסיון של 10 שנים בתחום הסייבר הכוללים פיתוח יישומים, הדרכות לצוותי PT, שירותי סייבר ומודיעין. כיום, מאור מספק שירותי ייעוץ לחברות גדולות בתחום הפיתוח המאובטח, מחשוב ענן ומבדקי חוסן שונים. מחבר הספר "Adavcned Web Hacking Handbook" שיצא לאור בשנת 2019. ניתן ליצור קשר עם מאור בלינקאדין.