חלק א:

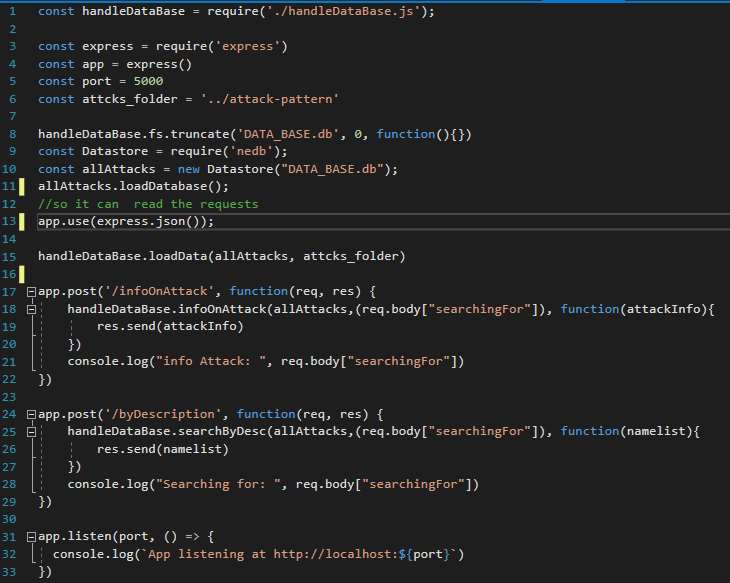
**:backend**

ה backend בנוי מקובץ app.js, handleDataBase.js ו-DATA\_BASE.db (קובץ בסיס הנתונים)

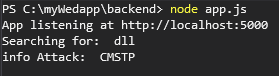
קובץ app הוא הmain של ה backend והוא מנהל את הבקשות והגישה לבסיס הנתונים דרך הקובץ handleDataBase.js.

מייבא את הפונקציות שמטפלות בבסיס הנתונים

מתחיל להקשיב לבקשות http בlocalhost על port מסוים(במקרה הזה 5000)

בריבועים הירוקים ה backend מטפל בבקשות post שנשלחו אליו לאחר שהוא התחיל להקשיב לבקשות(app.listen). בבקשות אלו הוא מקבל את ההודעה מהלקוח(frontend) וקורא לפונקציה המתאימה מהקובץ handleDataBase ושולח ללקוח תשובה מתאימה.

כאשר ה backend מקבל בקשה הוא עושה console.log לבקשה וכך כשאני בונה עוד לקוח שמדבר עם ה backend קל לראות אם ה backend קיבל את הבקשה ואיזה בקשות הוא קיבל לדוגמה:



בצהוב: מסומנים הפעולות שהוא עושה על הbd:

שורה 8 מוחקת את המידע שיש בקובץ הdatabase

לאחר מכן בשורות 9 10 11 ה backend פותח בסיס נתונים חדש(אם קיים אז הוא פותח את מה שקיים) של nedb.

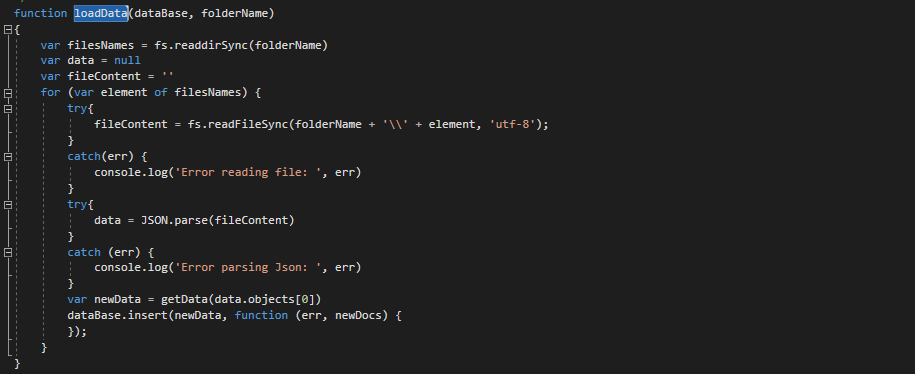
בשורה 15 הוא קורא לפונקציה(loadData) מהקובץ handleDataBase שקוראת את כל הקבצים בתקיה שהיא מקבלת ולוקחת מהם את המידע הרלוונטי ושמה בבסיס הנתונים.

\*\*ה backend משתמש בexpress בשביל לעבוד עם הבקשות http.

\*בחרתי למחוק את המידע הקודם מבסיס הנתונים ולטעון מחדש בכל הרצה של ה backend כי אם לדוגמה אני מוסיף קובץ מידע על מתקפה או משנה משהו או מוחק קובץ בסיס הנתונים של ה backend יתעדכן.

**handleDataBase.js:**

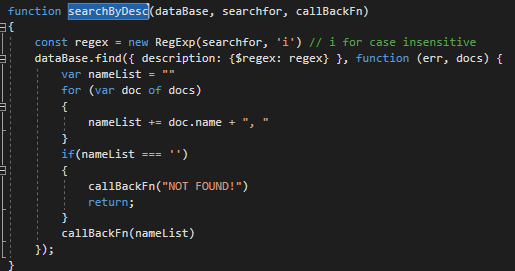
בקובץ הזה יש את הפונקציות שניגשות לבסיס הנתונים ומתעסקות איתו.

loadData – מקבלת בסיס נתונים ושם של התקייה עם הקבצים עם המידע וטוענת אותם לבסיס הנתונים.

הפונקציה עוברת על הקבצים בתיקייה וקוראת לפונקציה getData שמחזירה מידע בפורמט מתאים לdatabase של nedb ועושה למידע החדש insert לבסיס הנתונים.

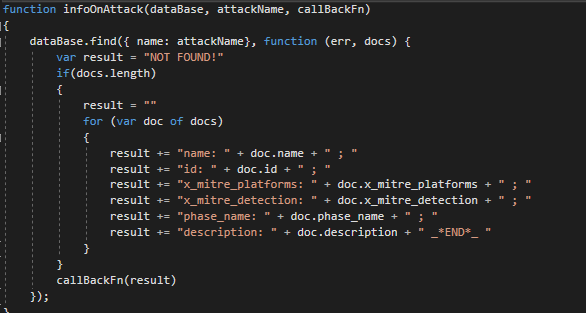
getData – מקבלת מידע מסוג object ומחזירה את המידע בפורמט מתאים לnedb (במידה וחסר מידע באחד מהשדות היא שמה שם NA)

searchByDesc - הפונקציה מקבלת מחרוזת חיפוש בסיס נתונים וcallbackfunction - ומחפשת בבסיס הנתונים את המחרוזת באופן לא רגיש (אותיות קטנות או גדולות או substring) ומחזירה רשימה של כל השמות שמכילות את המחרוזת בתיאור שלהן.



הפונקציה משתמשת בפונקציה .find של בסיס נתונים nebd. הפונקציה (במקרה פה) מחזירה את כל ה documents(לכל מתקפה יש document אחד שמכיל את כל המידע הרלוונטי) שמכילים בתיאור את מחרוזת החיפוש.

infoOnAttack – מחזירה מידע על מתקפה.

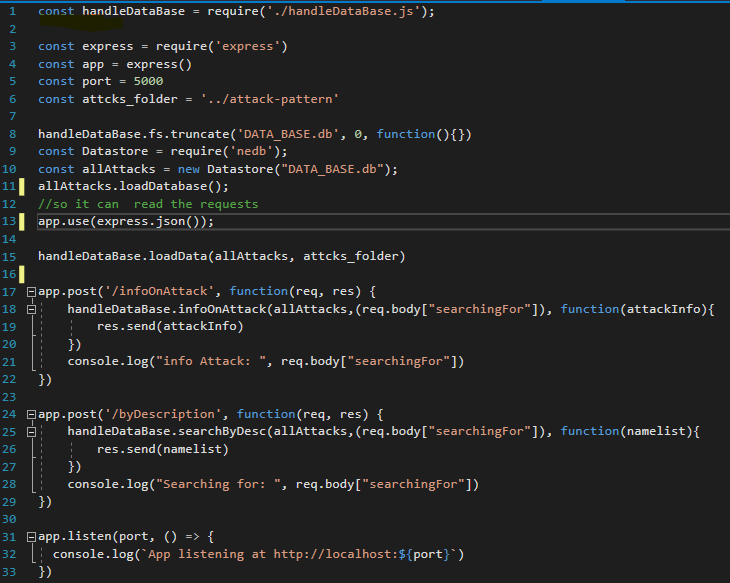
 עושה פעולה דומה ל searchByDescרק שהיא מחפשת בname באופן רגיש (השם ומחרוזת החיפוש צריכים להתאים באופן מלא).

הפונקציה מקבלת חזרה את כל הdocs שמתאימים לחיפוש והופכת אותם למחרוזת עם המידע על מתקפה מסויימת.

\*שתי פונקציות החיפוש קוראות בסוף לcallbackFunction ומעבירה להם את המידע המוחזר.

הסיבה שהשתמשתי בcallbackfunc זה בגלל שnode.js קורה באופן אסינכרוני. לכן כאשר אני קורא לinfoOnAttack זה לא משנה לו אם הפעולות שבתוך הפונקציה הושלמו והוא פשוט מנחש שכן וממשיך לשורה הבאה. בגלל זה אם לא הייתי משתמש בcallbackfunc שאילת החיפוש(find) והפונקציה לא הייתה מסתיימת כשזה המשיך לשליחה (res.send) והיה יוצא Undefined. כשאני משתמש בcallback מתוך השאילתה ורק אחרי הפעולה אני בעצם מוודא שהיא הסתיימה לפני שאני ממשיך.

דוגמה לשימוש ב infoOnAttack



\*את המידע כתבתי למסד הנתונים בצורה של documents. דוגמה של document:

{"name":"Extra Window Memory Injection","id":"attack-pattern--0042a9f5-f053-4769-b3ef-9ad018dfa298","x\_mitre\_platforms":["Windows"],"x\_mitre\_detection":"Monitor for API calls related to enumerating and manipulating EWM such as GetWindowLong (Citation: Microsoft GetWindowLong function) and SetWindowLong (Citation: Microsoft SetWindowLong function). Malware associated with this technique have also used SendNotifyMessage (Citation: Microsoft SendNotifyMessage function) to trigger the associated window procedure and eventual malicious injection. (Citation: Endgame Process Injection July 2017)","phase\_name":"NA","description":"something something (the original was too long for word)","\_id":"NWWAQDrfpFZJbKo9"}

דוגמה להוספת document חדש:

newData = { name: "name\_d" ,

id: "id\_d",

x\_mitre\_platforms: "x\_mitre\_platforms\_d",

x\_mitre\_detection: "x\_mitre\_detection\_d",

phase\_name: "phase\_name\_d",

description: "description\_d"}

dataBase.insert(newData, function (err, newDocs) {});

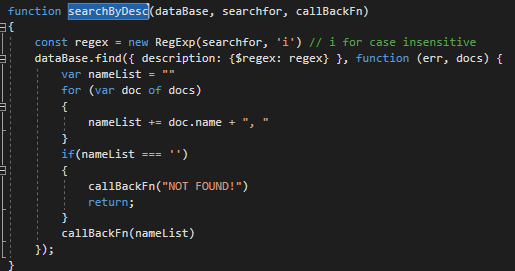
nedb database מוסיף אוטומטית שדה \_id לכל document חדש שמוסיפים.

שליפת מידע:

בשביל שליפת מדע השתמשתי בפונקציה .find של המסד נתונים(של nebd)

הפונקציה מחזירה מערך של documents המתאימים לחיפוש.

לדוגמה(שליפת המידע searchByDesc):



\*בחרתי להשתמש בnedb בגלל שקראתי עליו וראיתי שיש לו את הפונקציונאליות שאני מחפש ושהוא נשמר בקובץ לוקאלי על המחשב ואני יכול לראות בקלות איך נראה המידע שנשמר בו.

**Frontend:**

למשתמש יש אופציה של חיפוש לפי תיאור וקבלת רשימת שמות של כל המתקפות המתאימות(רק שם).

\*חיפוש כלום (לחיצה על אנטר או על הכפתור יחזיר רשימה של כל שמות המתקפות)

בנוסף יש אופציה לשים של של מתקפה ולקבל את המידע עליה. (יש להכניס שם מלא כולל אותיות קטנות וגדולות)

השתמשתי בreact בשביל הfrontend

בקובץ ה package.json שמקבלים כשמוסיפים את react הוספתי את השורה

"proxy": "http://localhost:5000",

כדי שכל בקשה שנשלחת ממנו תשלח ל backend אם לא צויין שם אתר. עשיתי זאת בגלל שידעתי שכל מה שהfrontend יעשה זה לתקשר עם הbackend שבניתי. לדוגמה:

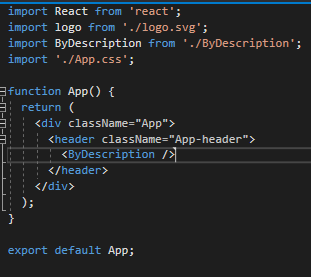
axios.post("/infoOnAttack", { "searchingFor" : this.state.searchingFor }) .then(response =>{

this.setState({searchResulte: response.data});

פה יש את הקבצים(שאני שיניתי או הוספתי בנוסף לקבצים שבאים אוטומטית עם react )

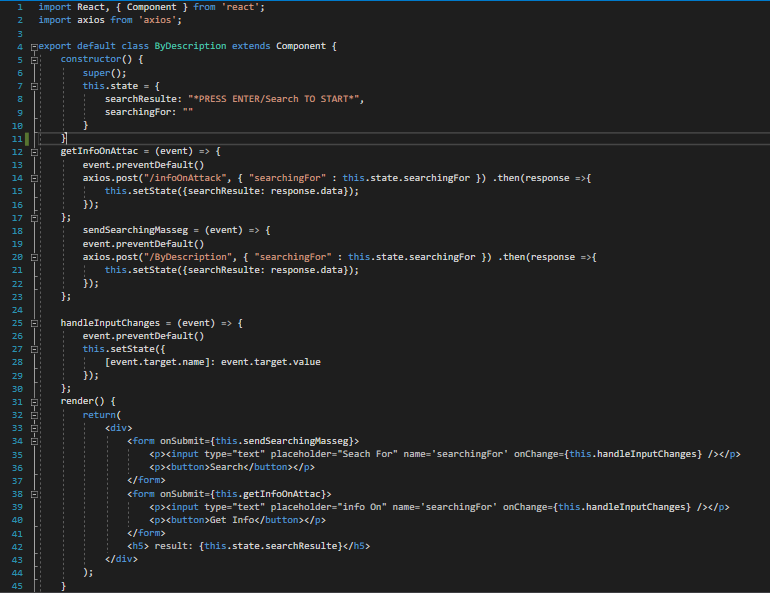
App.js ו - ByDescription.js

בapp.js יש את הבסיס של הדף Html של האפליקציה:

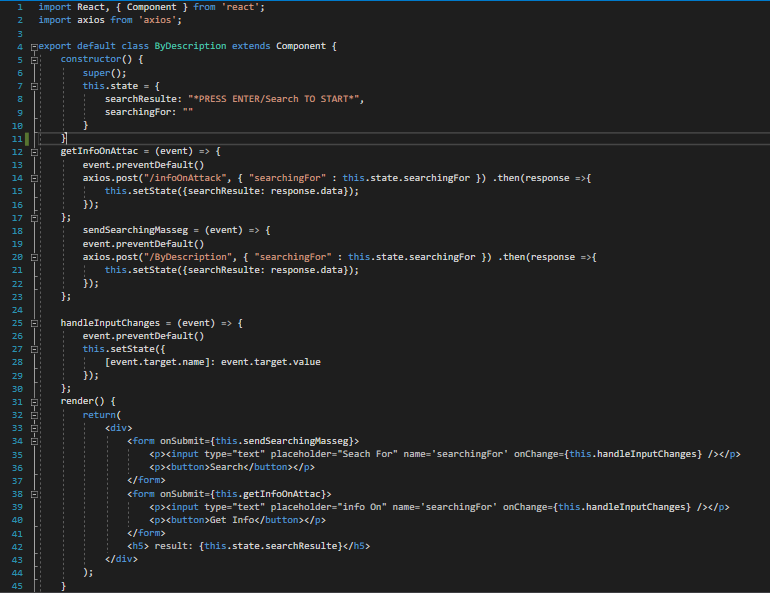
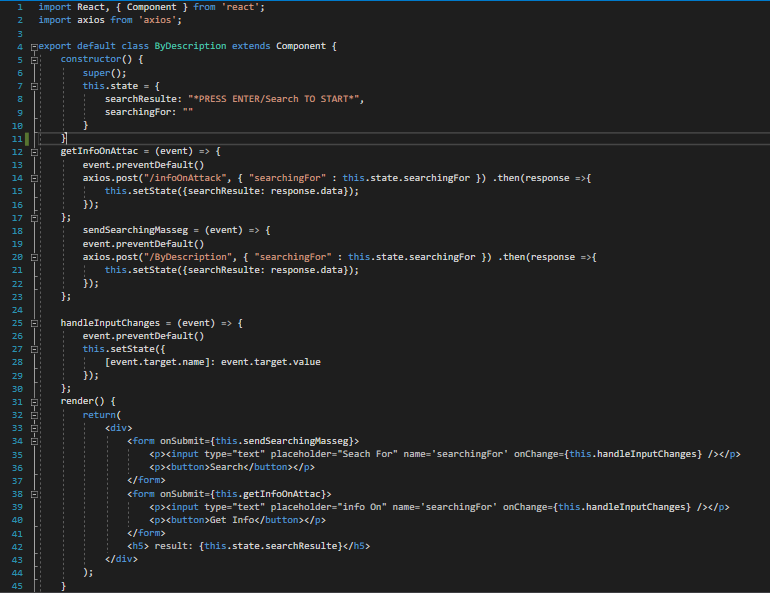


ByDescription היא מחלקה שבעצם מקבלת את הקלט מהמשתמש בדף ושולחת בקשות http לbackend

**ByDescription.js –**

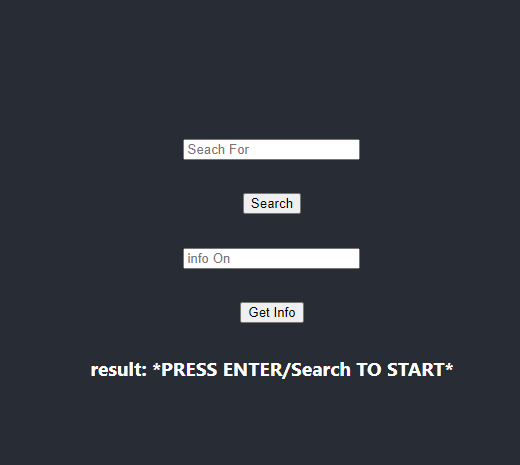
המחלקה אחראית לקבלת קלט מהמשתמש ושליחת מידע והצגתו:

החלק הכתום הוא בעצם עיקר הדף html יש בו כפתורים שלכל כפתור יש תיבת קלט וכפתור submit

בזמן השינוי (כתיבה לתיבת הinput) נקראת המתודה handleInputChanges שמכניסה את המחרוזת שבתיבה לשדה של המחלקה(לפי שם ה (input)

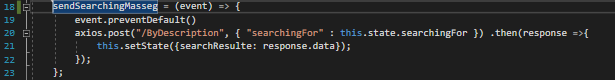
בעצם משנה את ערך השדה searchingFor לInput

(בתמונה מסך הפתיחה של האפליקציה – הfrontend - react)

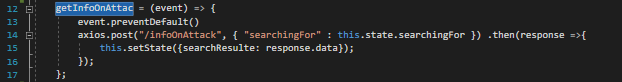


לכל אחד מהכפתורים (ותיבות הקלט שלהם (או בקיצור לכל form)) יש פונקציה של submit ששולחת את החיפוש (השדה searchingfor) לbackend מקבלת תשובה ושמה אותה בשדה searchResulte המוצג למשתמש (בהתחלה הוא \*PRESS ENTER/Search TO" "START\* כמו שניתן לראות בתמונה)

לראשון(search) יש את המתודה sendSearchingMasseg ששולחת הודעת חיפוש לפי תיאור לbackend.

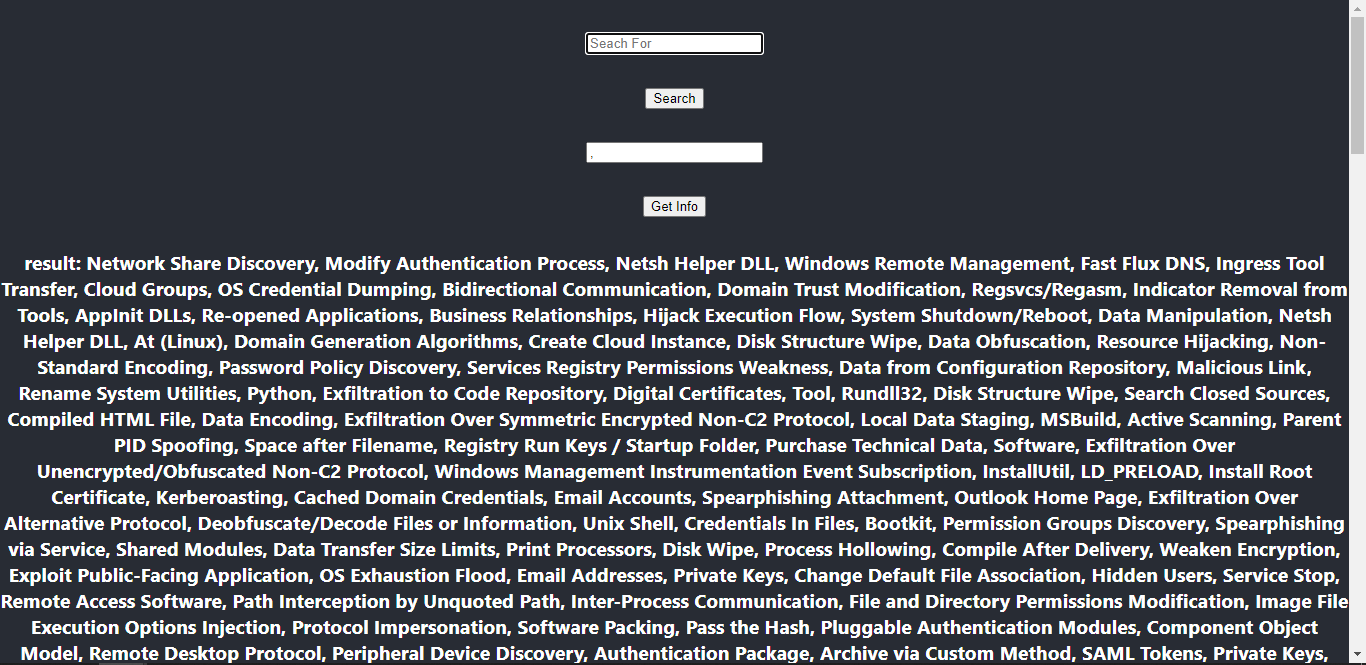


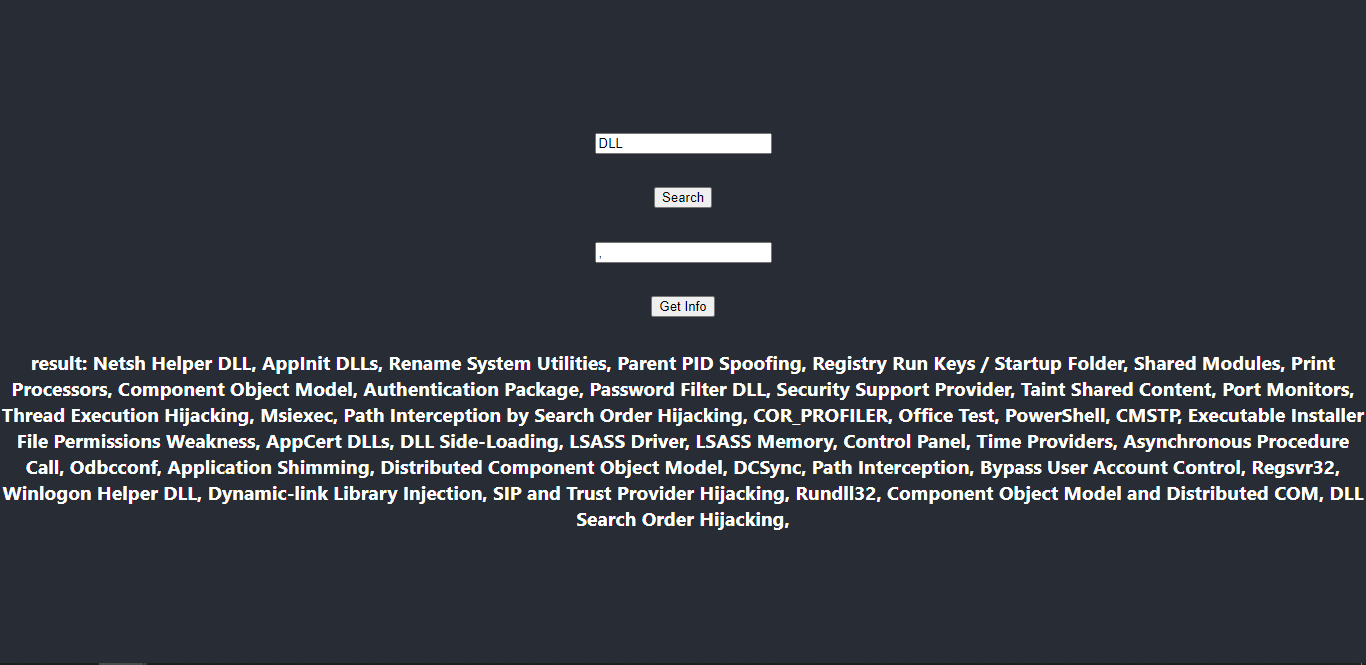
לשני(info On) יש את המתודה getInfoOnAttac ששולחת הודעת חיפוש לפי שם לbackend.

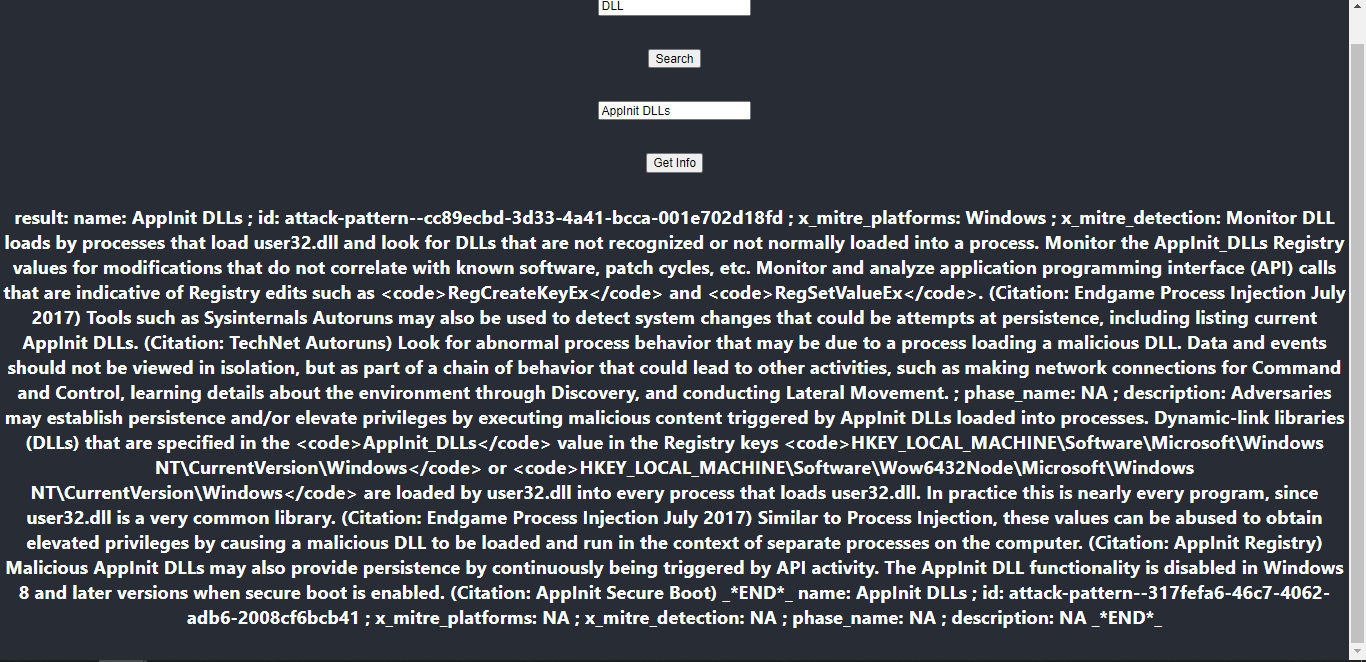


\*השתמשתי הaxios כי זה דרך קלה לשלוח בקשות http

דוגמאות:

קבלת השמות של כל המתקפות על ידי חיפוש כלום:

קבלת שמות כל המתקפות שמכילות DLL בתיאור:

חיפוש פרטים על מתקפה מסויימת ("END\*\_\*\_" מסמל סוף מידע על מתקפה – כי יש לפעמים יותר ממתקפה 1 עם אותו שם):

\*הסיבה שבחרתי לעשות זאת כך (המחרוזת עם \_\*END\*\_ זה בגלל שידעתי שיש גם את השלב של הבוט שגם הוא יתקשר עם הbackend – ובגלל שהוא צורת chat הוא חייב לפעול כך)

אם אין מתקפה בשם או שאין מתקפה עם המילה\מחרוזת בתיאור יוחזר:

