**מסמך ארכיטקטורה**

# **מבט על**

**דיאגרמה של רכיבים עיקריים:**

**A screenshot of a cell phone

Description automatically generated**

קישור: <https://drive.google.com/file/d/1z7RPieKUNebzPWrzo3ZNfKnrz9610XzQ/view?usp=sharing>

**הסבר על כל רכיב:**

1. UI: רכיב ממשק למשתמש. בעזרת ה- RESTful API נוכל לממש בקלות ממשקי אינטראקציה למשתמש. מכיוון שהפרויקט שלנו מתמקד בעיקר בזיהוי מתקפות והפיכה של התוכנה ליעילה ומודולרית כמה שיותר, לא נתמקד ב-UI וכנראה הוא יהיה ממשק משתמש בתצורת CLI. באותה קלות נוכל לפתח אתר אינטרנט, אפליקציה ל-Mobile וכדומה.
2. Network: המודול הזה יקשיב לבקשות התחברות. ברגע של קבלה - הוא ישלח את הבקשה ל-Core Logic לעיבוד ואת התשובה יעביר חזרה למבקש. (טכנולוגיה – Python)
3. RESTful API: נוכל לבנות ממשקי משתמש שונים בעזרת ה-API הזה: הוא יוכל לקבל בקשה, לדוגמה "רשימת הפעולות שניתן לעשות עם פלאגין X" - תוך כדי תקשורת ישירה עם ה-Core Logic הוא יקבל את המידע ויחזיר אותו. מכיוון שהמידע יחזור כפורמט קבוע אך עם ערכים שונים, הממשק משתמש ידע לפרסר אותו ולהציג אותו בצורה טובה. (טכנולוגיה - Python).
4. Core Logic: הלב של PiNAT. מקבל פאקטות ובעזרת פלאגינים שונים הוא ינתח וישנה את המידע על פי הצורך ויחזיר את הפאקטות חזרה. את החלק הזה נפתח בc++
5. Plugins: תהיה מערכת שתדע לטעון פלאגינים שנכתבו עם API ספציפי שנספק. הפלאגינים האלה ינתחו, אחד אחרי השני, את המידע. הם יוכלו לשנות אותו, לשמור מידע וסטטיסטיקה עליו (#8) או פשוט לא להעביר אותו הלאה. (טכנולוגיה - כנראה Python).
6. Sniffer: הרכיב שמסניף תעבורה על PiNAT, מעביר אותה ל-Core Logic ומחזיר אותה חזרה. (טכנולוגיה – Python / c++).
7. Database\_Manager: אחראי על שמירה ושליפה של מידע ממסד נתונים **כלשהו**. פלאגינים, וגם Core Logic יצטרכו באופן שוטף לשמור מידע ולכן חיוני שרכיב זה ידע לעבוד עם מספר בקשות במקביל, וגם יהיה מודולרי אם נרצה להחליף את המסד נתונים. (טכנולוגיה - Python).
8. DB: מסד הנתונים במקרה שלנו יהיה SQLite. (טכנולוגיה – SQLite).

# **עיצוב הנתונים ויישויות מידע**

* משתמש – שם משתמש, סיסמה והרשאות קריאה ועריכה. פרטים אלו נשמרים בבסיס הנתונים ונקראים כאשר משתמש מנסה להתחבר
* פאקטה – חבילה של מידע שנקלטת אצל הsniffer ומעובדת על ידי הcore logic. מידע רלוונטי ישמר בבסיס הנתונים
* אירוע חשוד – מכיל תאריך, שעה ותיאור של האירוע (לדוגמה, נסיון mitm, dns spoofing, וכו'). ישמר בבסיס הנתונים.
* Blacklist – רשימה של כתובת mac וip חסומות שישמרו בבסיס הנתונים

# **טכנולוגיות עיקריות**

* Linux – הפרויקט שלנו ירוץ על לינוקס בעיקר משום שזו מערכת נוחה יותר לעבודה בנושא רשתות. יש אפשרות לקנפג את כרטיס הרשת ולשנות את טבלת הניתוב בצורה יחסית פשוטה.
* Python – בשפה זו בחרנו לכתוב את ניהול הפלאגינים, בסיס הנתונים והתקשורת עם הclient. זו שפה נוחה וגמישה שמכילה כבר הרבה ספריות לעבודה עם בסיס נתונים ורשתות. כמו כן, נוח יותר להריץ לפאגינים בפייתון מכיוון שהיא לא מקופלת, כך שניתן ישר לטעון אותם.
* C++ - בשפה זו בחרנו לכתוב את הלוגיקה המרכזית של הפרויקט. זו שפה קלה (lightweight) וחזקה שרצה מהר בהשוואה לשפות עיליות, ולכן היא מתאימה למימוש הלוגיקה המרכזית, שדורשת יותר כוח חישוב ומשאבים. כמו כן, קיימת אינטגרציה בין c++ לpython (לפחות הגירסה שממומשת בc).
* Sqlite – בסיס נתונים מהיר שניגשים אליו דרך קובץ הרצה יחיד, ואין צורך להריץ שרת נפרד.
* Raspberry pi – במידה והraspberry pi יאפשר זאת מבחינת חומרה, נריץ את הפרויקט עליו, מכיוון שהוא נייד וקומפקטי, והוא זול יותר ממחשב שלם.

# **התאמה לאפיון**

|  |  |
| --- | --- |
| **פיצ'ר** | **רכיבים רלוונטים** |
| שימוש בRESTful API | לקוח - ממשק משתמש -> תקשורת  שרת - תקשורת -> ממשק API -> לוגיקה מרכזית |
| הסנפת תעבורה וביצוע פעולות עליה | שרת - סניפר -> לוגיקה מרכזית -> פלאגינים -> בסיס נתונים (במידת הצורך) |
| זיהוי מתקפות ברשת | שרת - סניפר -> לוגיקה מרכזית -> פלאגינים -> בסיס נתונים (במידת הצורך) |
| הצגת מצב רשת נוכחי | לקוח – ממשק משתמש -> תקשורת  שרת – תקשרות -> ממשק API - > לוגיקה מרכזית -> סניפר |
| מנגנון אותנתיקציה | לקוח - ממשק משתמש -> תקשורת  שרת - תקשורת -> ממשק API -> לוגיקה מרכזית -> בסיס נתונים |
| טעינה של פלאגינים | לקוח - ממשק משתמש -> תקשורת  שרת - תקשורת -> ממשק API -> לוגיקה מרכזית -> פלאגינים |
| ניתוק משתמשים מהרשת | לקוח - ממשק משתמש -> תקשורת  שרת - תקשורת -> ממשק API -> לוגיקה מרכזית |
| חסימה של כתובות IP וMAC | לקוח - ממשק משתמש -> תקשורת  שרת - תקשורת -> ממשק API -> לוגיקה מרכזית -> בסיס נתונים |