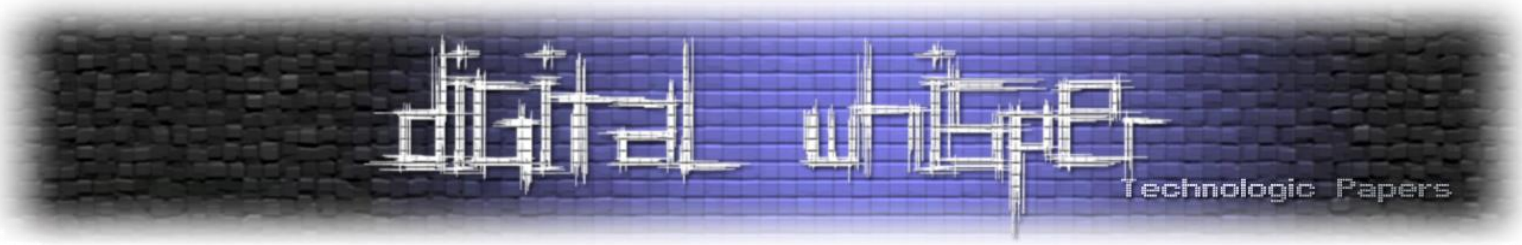


מדריך מעשי למציאת IDOR בעולם האמיתי

מאת אבישי גונן

הקדמה

כשאנשים חושבים על פריצה לאפליקציות, הם לרוב מדמיינים תרחישים מורכבים - ניצול של חולשות 0day, כלים מתוחכמים וטכניקות מורכבות, חודשים של עבודה. המציאות, לפחות בעולם של Broken Access Control, נפוצה ומדאיגה הרבה יותר: לפעמים השרת פשוט לא בודק אם המשאב ששלחת בקשה עליו - שייך לך.

Broken Access Control הוא אחד מהממצאים הנפוצים ביותר בדוחות אבטחה, ועלה לראש רשימת ה-OWASP Top 10 משנת 2021. הסיבה פשוטה: קל לממש Authentication - לוודא שהמשתמש הוא מי שהוא אומר שהוא. קשה יותר לממש Authorization - לוודא שהמשתמש המאומת מורשה לגשת דווקא למשאב הספציפי שהוא מבקש.

במאמר הזה אעבור צעד-צעד על תהליך בדיקת אפליקציית מובייל ישראלית לשירותי משלוחים, ואראה כיצד נקודת עיוורון אחת בצד השרת הובילה לשרשרת ממצאים: גניבת כתובות, גניבת טוקן כרטיס אשראי, ודלף מידע אישי מלא. אבל הסיפור האמיתי מתגלה רק בסוף - כי לא מדובר באפליקציה אחת.

המאמר הזה מיועד גם כמדריך מעשי: אתחיל מהקמת סביבת עבודה (Mobile Pentesting Lab) מאפס, כך שגם קוראים שמעולם לא חקרו אפליקציות מובייל יוכלו לעקוב ולנסות בעצמם.

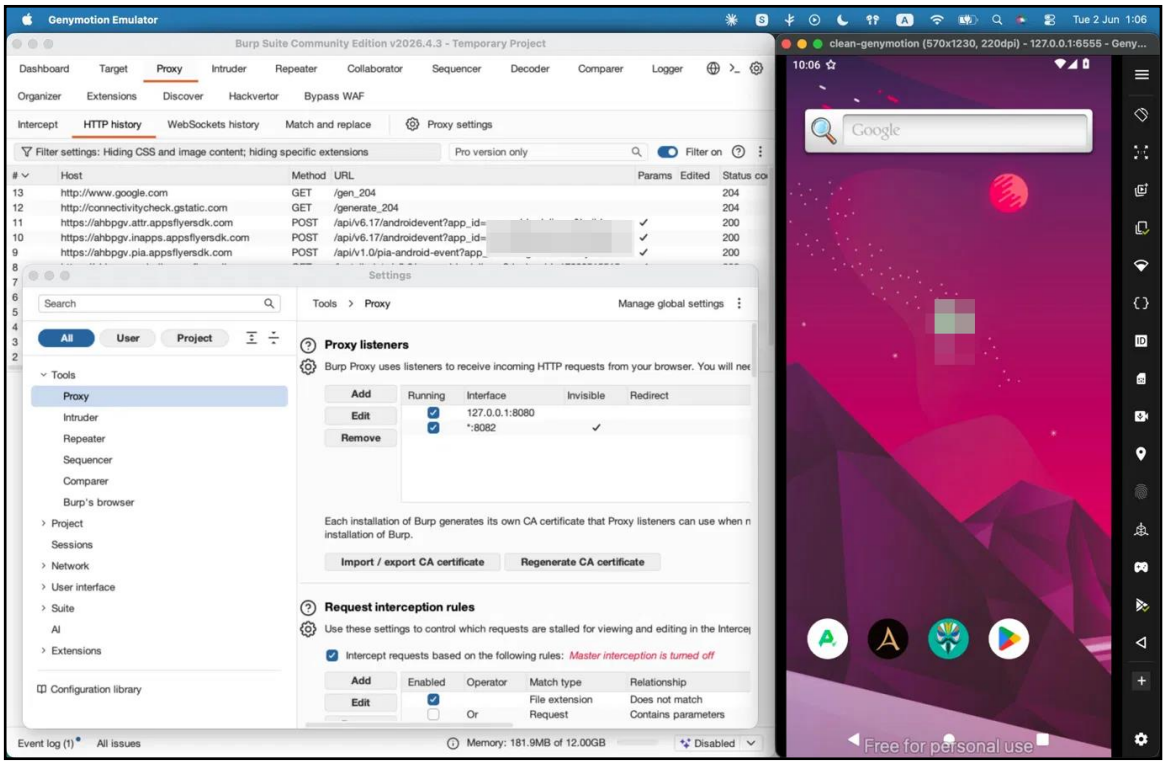
הכלים - הקמת סביבת Mobile Pentesting

לפני שנצלול לממצאים, נסקור בקצרה את הכלים שבהם השתמשתי ואת אופן ההגדרה שלהם. מי שכבר מכיר את הסביבה הזו - מוזמן לדלג לפרק הבא.

Genymotion - אמולטור Android

Genymotion הוא אמולטור Android המיועד לפיתוח ולבדיקות. בניגוד לאמולטור המובנה של Android Studio, Genymotion מציע ביצועים טובים יותר, ממשק נוח להגדרת פרוקסי, וגמישות בבחירת גרסת ה-

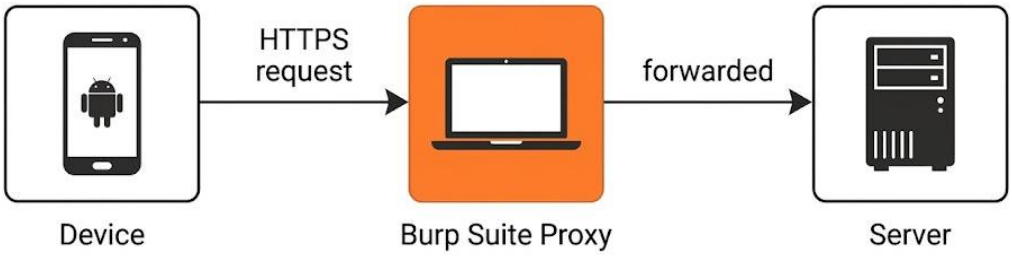
Android ורזולוציית המסך. בנוסף, קל יחסית להתקין עליו Burp Certificate ולראות תעבורה מוצפנת. ניתן להוריד את Genymotion בחינם לשימוש אישי מהאתר הרשמי. לאחר ההתקנה, יוצרים מכשיר וירטואלי עם גרסת Android שמתאימה לאפליקציה הנבדקת, ומגדירים בהגדרות הרשת של המכשיר את כתובת ה-IP של המחשב המארח ואת פורט ה-Proxy (ברירת מחדל ב-Burp Suite: 8080):



Burp Suite - פרוקסי לבדיקת אבטחה

Burp Suite הוא הכלי המרכזי של עולם Web ו-Mobile Security. הוא מיוצר על ידי חברת PortSwigger ופועל כ-Proxy שיושב בין המכשיר לשרת, ומאפשר ליירט, לבחון ולשנות בקשות HTTP בזמן אמת. גרסת ה-Community חינמית לחלוטין ומספיקה לרוב הבדיקות.

לאחר הפעלת Burp Suite, יש לייצא את ה-CA Certificate שלו ולהתקין אותו על האמולטור כ-Trusted CA. בגרסאות Android 7 ומעלה, אפליקציות לא סומכות על Certificates שהותקנו על ידי המשתמש כברירת מחדל. ניתן לעקוף זאת על ידי שינוי ה-Network Security Config של האפליקציה, שימוש ב-Magisk עם המודול MagiskTrustUserCerts (למכשיר פיזי מ-root), או שימוש ב-Frida לעקיפת SSL Pinning אם קיים.

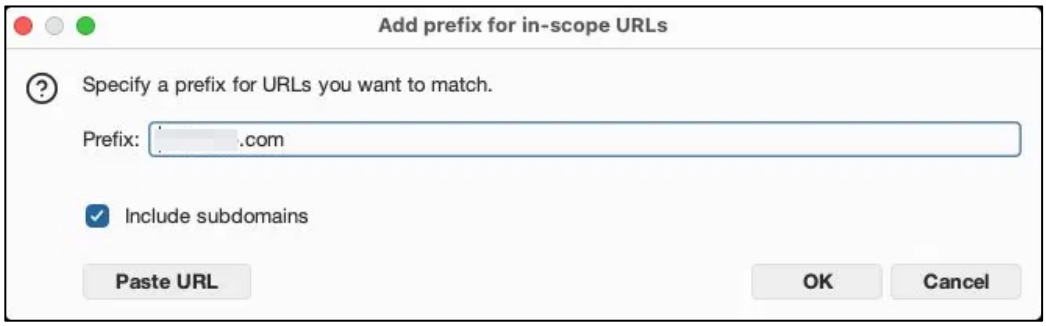


הגדרת ה-Scope וסביבת שני מכשירים

בדיקת **IDOR** דורשת לפחות שני חשבונות: חשבון התוקף וחשבון הקורבן. השתמשתי בשני מכשירים:

- **מכשיר התוקף**: אמולטור Genymotion עם Burp Suite מוגדר כ-Proxy
- **מכשיר הקורבן**: טלפון פיזי rooted (מכשיר Android) עם חשבון נפרד.

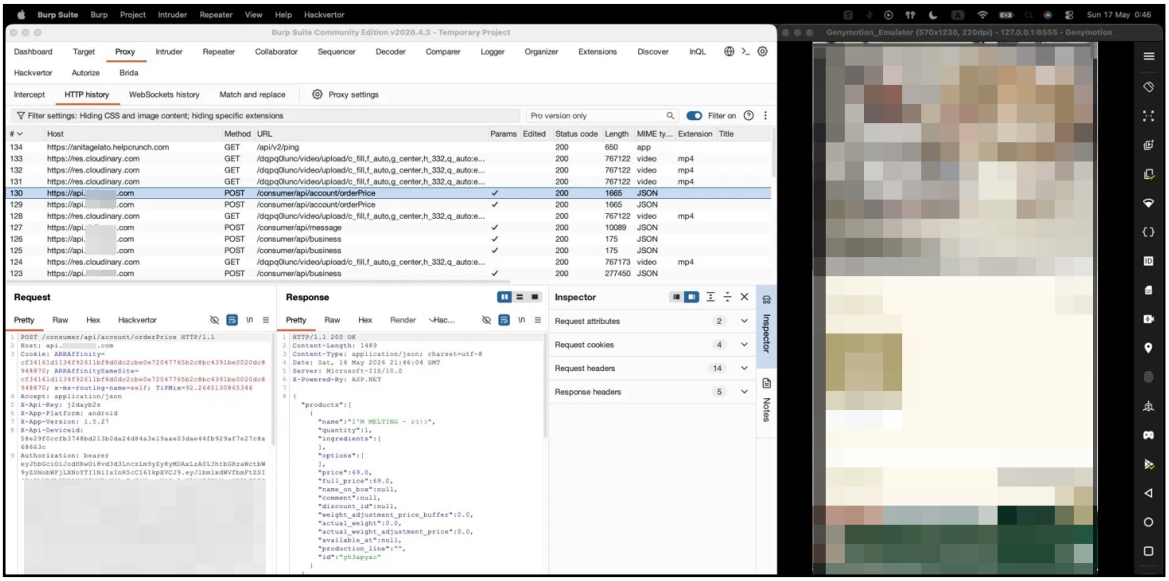
ב-Burp Suite, הגדרתי את ה-Scope לכלול רק את redacted.com וכל תת-הדומיינים שלו, כדי לסנן רעש מיותר מבקשות של שירותים אחרים (Cloudinary, CDN וכדומה).



מקרה הבוחן - כניסה לאפליקציה

האפליקציה הנבדקת היא שירות משלוחים ישראלי. כמו אפליקציות רבות בישראל, היא מבוססת על Backend חיצוני של ספק צד-שלישי. כשפתחתי את האפליקציה עם Burp Suite מוגדר, ראיתי שכל הקריאות עוברות לשרת ה-API של הספק.

זיהיתי שהאפליקציה משתמשת ב-JWT (JSON Web Token) לצורך אימות - הטוקן נשלח בכל בקשה ב-Header מסוג Authorization: Bearer. עד כאן הכל נראה תקין. הבעיה, כפי שנראה, היא שהשרת אמנם מאמת שהטוקן תקף - אבל אינו מוודא שהמשתמש ניגש רק למשאבים שלו.



חלק א' - IDOR על כתובות

מה זה IDOR?

IDOR (Insecure Direct Object Reference) הוא סוג של Broken Access Control. הוא מתרחש כאשר שרת מאפשר גישה למשאב על סמך מזהה שנשלח מהלקוח, מבלי לוודא שהמשתמש המבקש אכן בעל הרשות לגשת למשאב הזה. המזהה יכול להיות מספר סדרתי, UUID, או כל מחרוזת זיהוי אחרת.

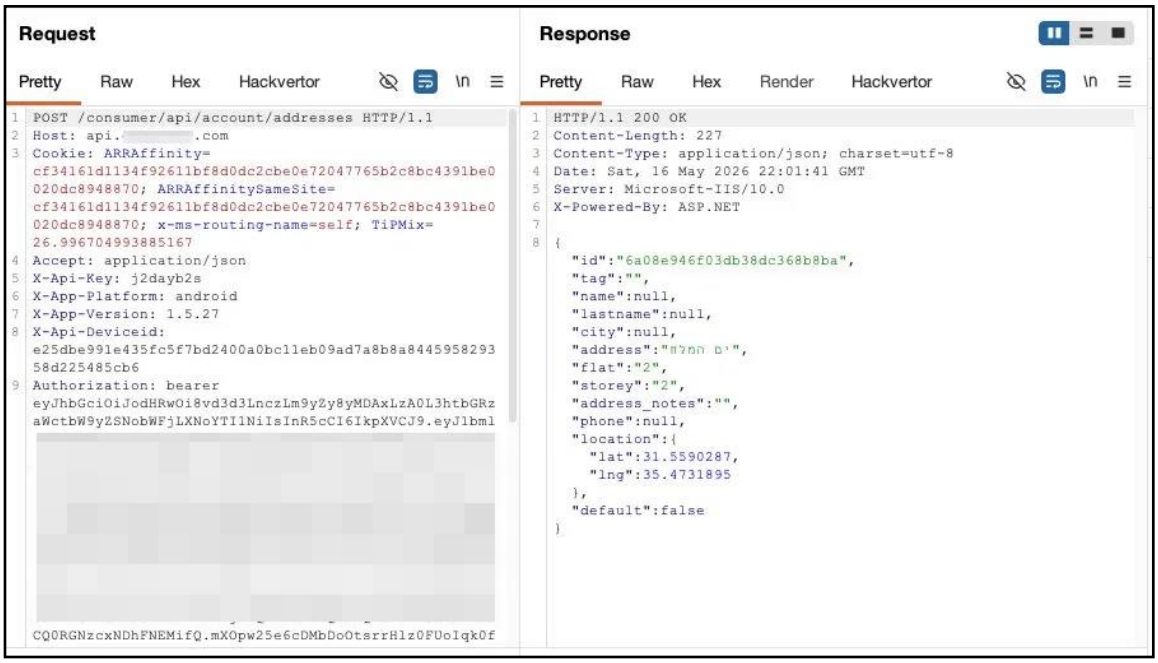
גילוי החולשה

ניווטתי לאזור הכתובות השמורות באפליקציה וערכתי כתובת קיימת. הבקשה שנשלחה נראתה כך:

```
PUT /consumer/api/account/addresses/6a02b23a66517d51e9e10f39 HTTP/1.1
Host: api.REDACTED.com
Authorization: Bearer <attacker_token>
```

גניבת כתובת הקורבן

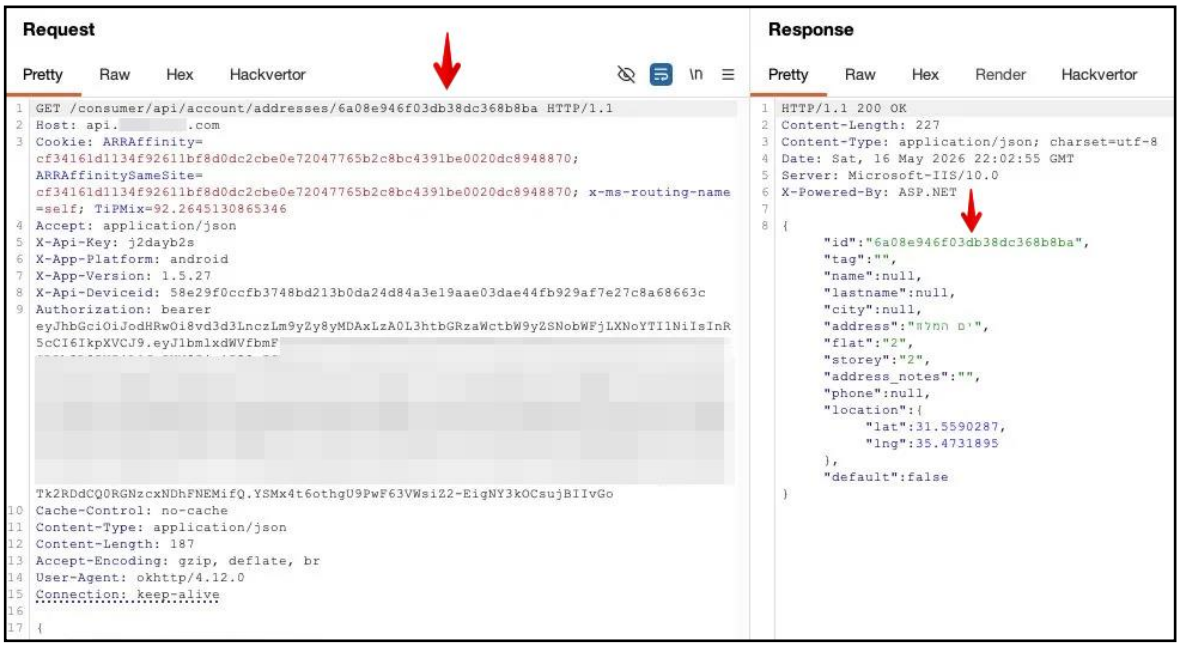
הקורבן יצר כתובת חדשה במכשיר שלו. יירטתי את תשובת ה-**POST** מצד הקורבן - שחשפה את מזהה הכתובת שנוצרה.



עם המזהה הזה, שלחתי GET request מחשבון התוקף, עם ה-JWT של התוקף בלבד:

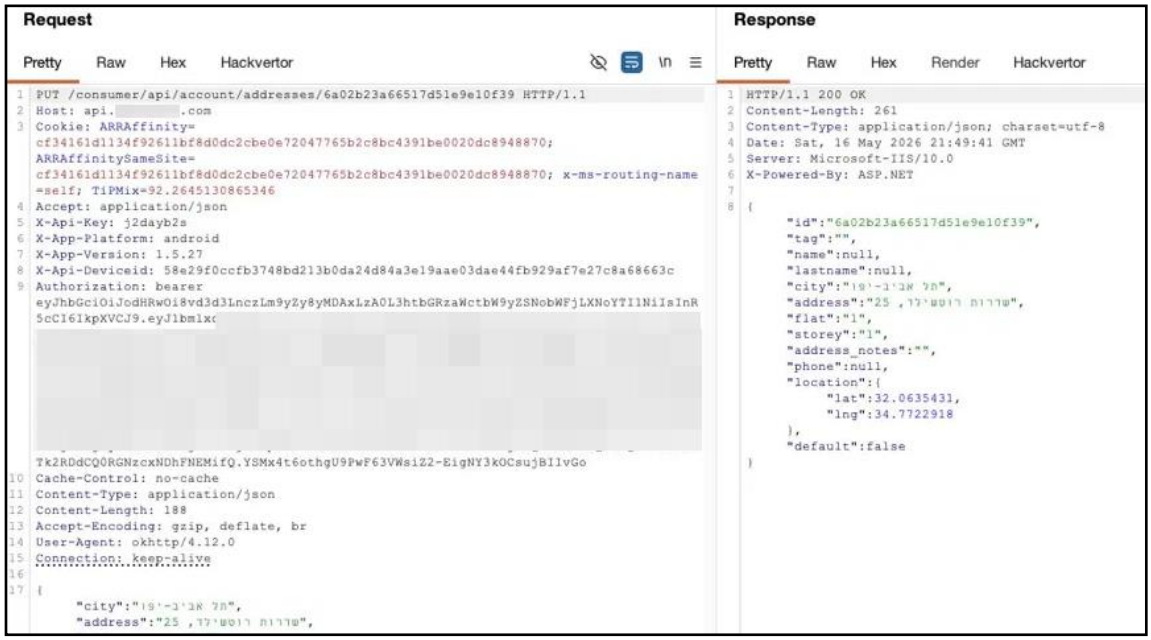
```
GET /consumer/api/account/addresses/6a08e946f03db38dc368b8ba HTTP/1.1
Host: api.REDACTED.com
Authorization: Bearer <attacker_token>
```

השרת החזיר HTTP 200 עם פרטי הכתובת המלאה של הקורבן - עיר, רחוב, מספר בית, GPS coordinates. אין שגיאת הרשאה, אין בדיקת בעלות.



שינוי כתובת הקורבן

עברתי שלב קדימה ושלחתי **PUT** לדריסת כתובת הקורבן עם נתונים בשליטתי. שדה address_notes הוגדר ל-**"Give me all of your ice cream"** - סתם כדי שהדגמה תהיה חד-משמעית. השרת החזיר **HTTP 200**.



במכשיר הקורבן, הכתובת השתנתה באופן מיידי ל-"The attacker city, The attacker house". המשתמש לא קיבל שום התראה.

מבחינה מעשית, תוקף יכול להפנות משלוח של קורבן לכתובת שבשליטתו, לשבש הזמנות, או לאסוף כתובות בית של משתמשים באופן שקט לחלוטין.

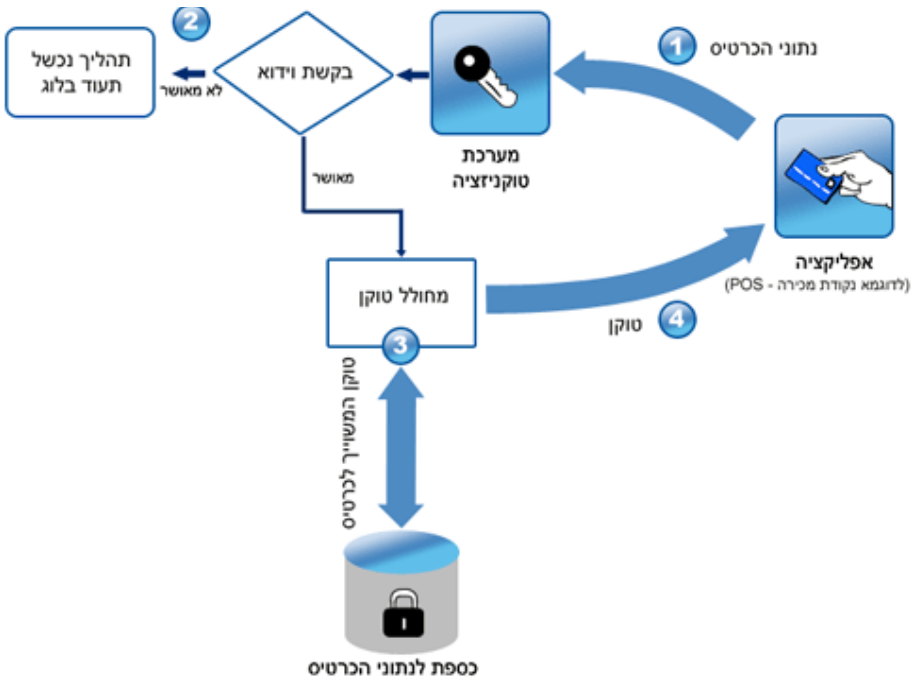
חלק ב' - גניבת כרטיס אשראי

זהו הממצא הכבד של המאמר.

כיצד עובד תהליך ה-Tokenization

האפליקציה משתמשת ב-שירות הסליקה כסליקת אשראי. שירות הסליקה הוא שירות Payment Processing ישראלי המספק, בין היתר, שירות Tokenization - האפליקציה אינה שומרת את פרטי כרטיס האשראי ישירות, אלא מקבלת טוקן (מחרוזת זיהוי תחליפית) המייצג את הכרטיס.
זרימת העבודה המלאה:

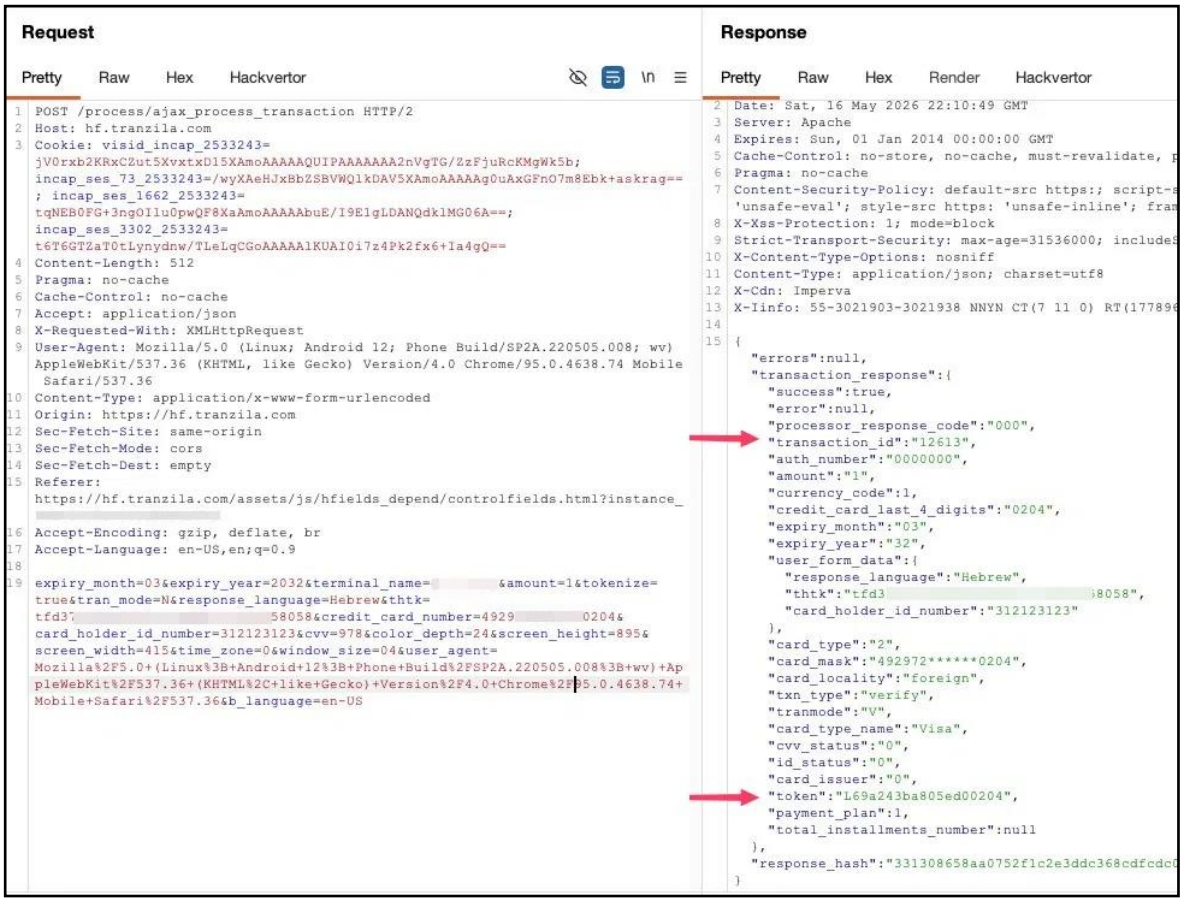
- האפליקציה שולחת את פרטי הכרטיס ישירות לשירות סליקה.
- שירות הסליקה שומר את הכרטיס ומחזיר transaction_id ו-token.
- האפליקציה שולחת את ה-transaction_id ל-api.redacted.com כדי לשייך את הכרטיס לחשבון.
- רכישות עתידיות משתמשות ב-token - מספר הכרטיס המלא לא נוגע בשרתי **הספק**.



על הנייר, זו ארכיטקטורה תקינה.

ה-Transaction ID הסדרתי

כשמשתמש מוסיף כרטיס, שירות הסליקה מחזיר בתשובה transaction_id - מספר שלם סדרתי. לא UUID, לא ערך אקראי. מספר שלם. כשהוספתי כרטיס, קיבלתי transaction_id מסוים.

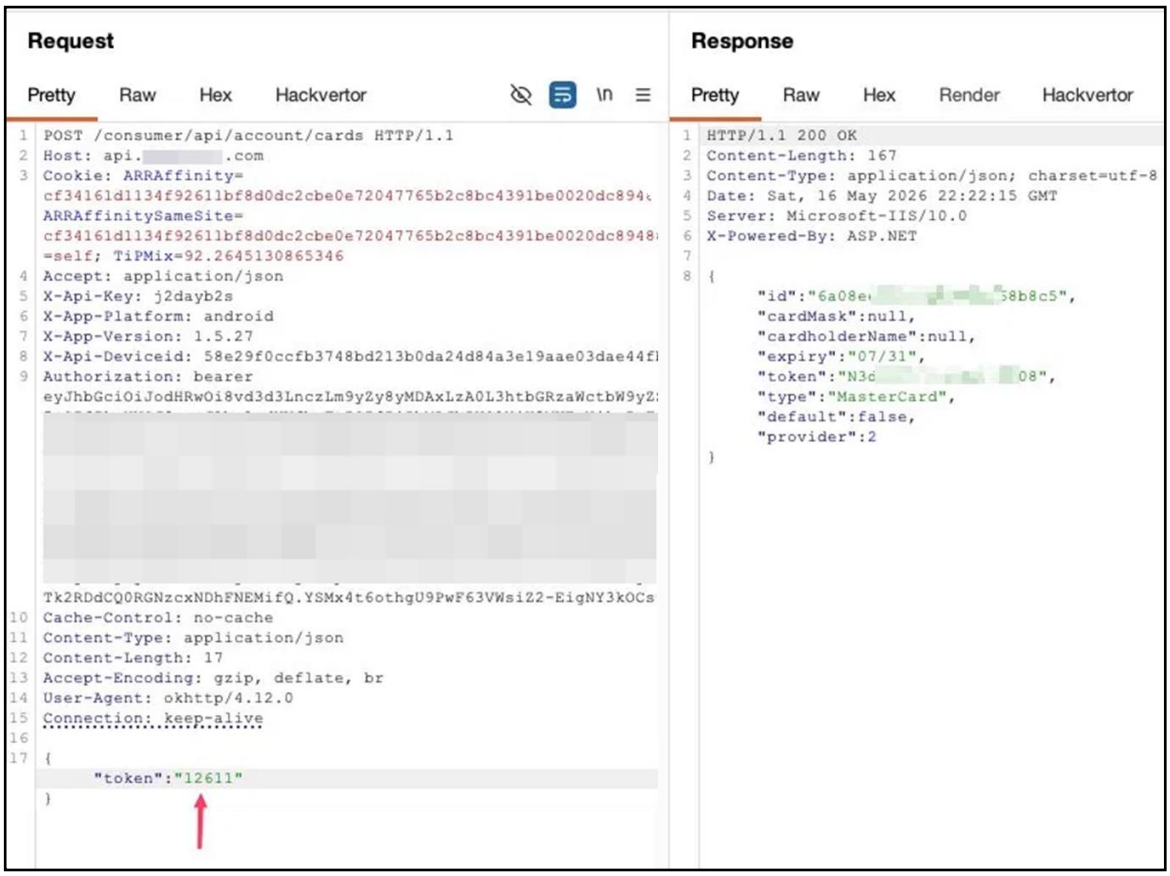


ה-transaction_id הזה נשלח ל-הספק כדי לשייך את הכרטיס לחשבון:

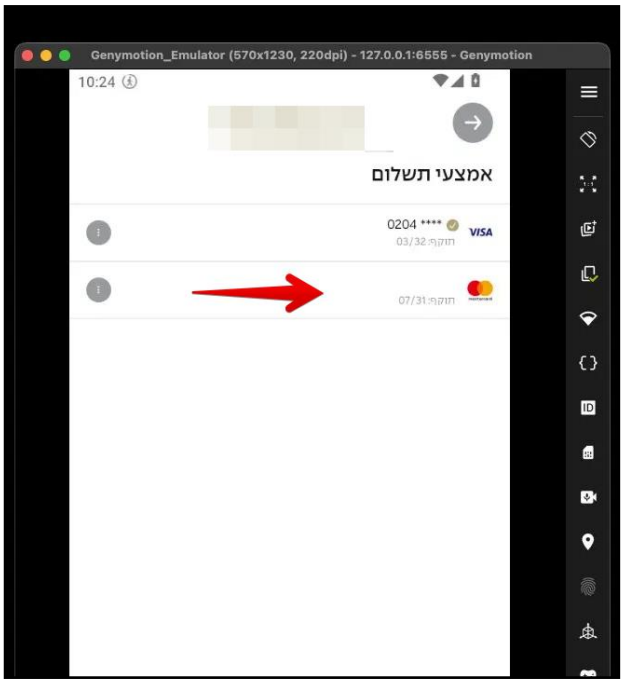
```
POST /consumer/api/account/cards HTTP/1.1
Host: api.REDACTED.com
Authorization: Bearer <attacker_token>

{"token": "N"}
```

השרת קיבל את ה-transaction_id, חיפש אצל שירות הסליקה את ה-token המשויך, ושייך אותו לחשבון.

מה קורה אם אשלח ערך אחר במקום N?



בדיוק מה שחשבתם. השרת לא בדק אם עסקת ה-שירות הסליקה שייכת לחשבון הנוכחי. כל transaction_id תקף - של כל משתמש בכל אפליקציה - מניב את ה-token של הכרטיס שלו. חשבון התוקף מציג כעת שני כרטיסים: כרטיסו שלו שלו, וכרטיס של אדם אחר של אדם אחר:

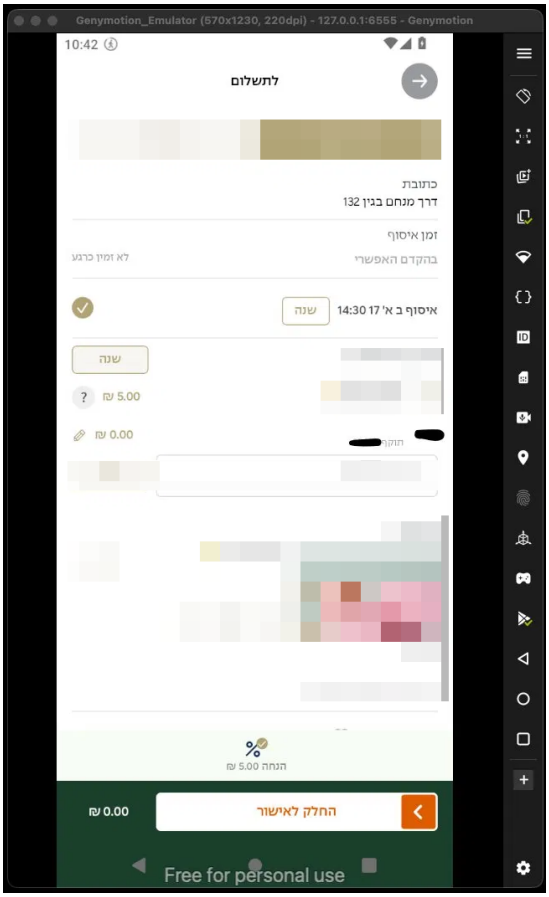


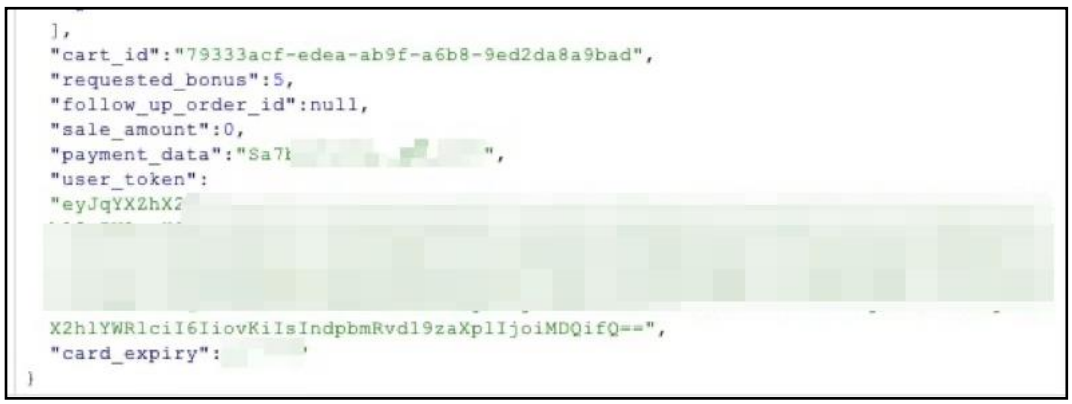
הוכחת ניצול - הזמנה בכרטיס גנוב

(הבהרה: לצורך ההדגמה שימשתי בעצמי כ"קורבן" - לא גנבתי מאף אחד). הקורבן הוסיף כרטיס אשראי אמיתי במכשיר שלו. שירות הסליקה הקצה לו transaction_id מסוים אחר. ממכשיר התוקף, שלחתי:

```
POST /consumer/api/account/cards HTTP/1.1
Host: api.REDACTED.com
Authorization: Bearer <attacker_token>

{"token": "N+1"}
```

כרטיס הקורבן הופיע בחשבון התוקף. ביצעתי הזמנה בכרטיס הגנוב - ההזמנה עברה, והקורבן חויב.





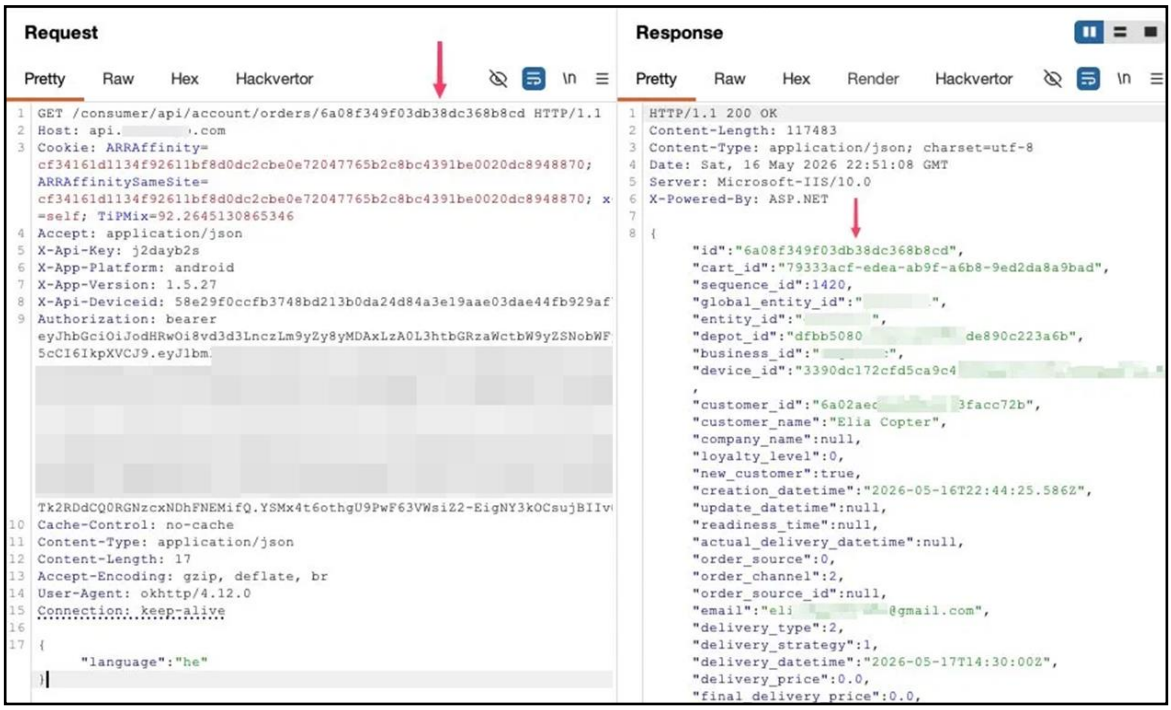
חלק ג' - IDOR על הזמנות ו-PII Dump

לאחר ביצוע ההזמנה, שמתי לב שנקודת הקצה של הזמנות עוקבת אחרי אותו דפוס:

```
GET /consumer/api/account/orders/<order_id> HTTP/1.1
Host: api.REDACTED.com
Authorization: Bearer <attacker_token>
```

התשובה היא כמות גדולה של מידע אישי. מעבר לתוכן ההזמנה, היא חשפה:

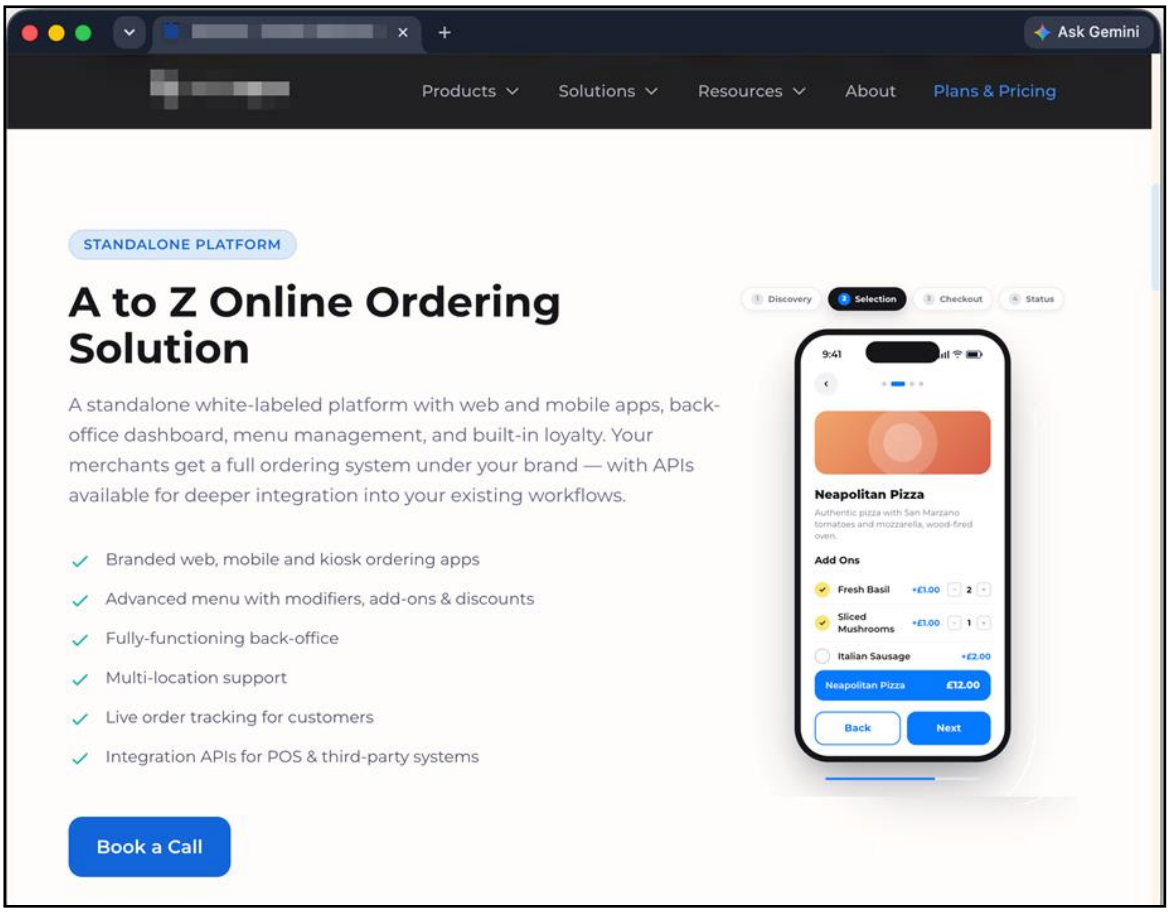
- שם מלא וכתובת אימייל של הלקוח.
- מספר טלפון גולמי ובפורמט E.164.
- כתובת משלוח עם קואורדינטות GPS.
- Cart ID, פריטים, כמויות ומחירים.
- payment_token - הטוקן ששימש לתשלום ההזמנה.
- Loyalty level ומטא-דאטה על החשבון.
- Business ID ו-Depot ID - שימושיים לתנועה לרוחב (Lateral Movement).



זוהי חשיפה חמורה יותר מ-IDOR על כתובות. בקשה אחת חושפת מספיק מידע לזיהוי מלא של הקורבן ולפנייה אליו. בשילוב עם גניבת כרטיס האשראי - תוקף יכול לאסוף טוקני תשלום בסדר גודל, כיוון שתשובת ה-order מכילה את ה-payment_token ששימש לרכישה. PUT על אותה נקודת קצה אינה מוגנת גם כן, ומאפשרת שינוי הזמנות בזמן אמת.

התמונה הגדולה - זה לא אפליקציה אחת

עד כאן ניתחנו חולשות באפליקציה אחת. עכשיו מגיע החלק שהופך את זה לסיפור אחר לגמרי.
ה-Backend של האפליקציה מסופק על ידי ספק White-Label ישראלי המשרת מאות עסקים ועשרות מיליוני הזמנות.



כל המותגים האלה רצים על אותו קוד, אותו API, ו-אותו בסיס נתונים משותף. בדקתי אפליקציה אחרת שמבוססת על אותו ספק - ואישרתי:

- כל ה-endpoints זהים.
- בסיס הנתונים משותף לכל האפליקציות - אין Tenant Isolation.
- JWT של האפליקציה עובד מול נתוני משתמשים של מותג אחר לחלוטין.

משטח התקיפה אינו אפליקציה אחת - הוא כל מערכת האקוסיסטם של הספק. כל עסק שמופיע ברשימת הלקוחות של הספק - חשוף לאותן המתקפות על המשתמשים שלו.

ניתוח - Authentication בלי Authorization

שורש הכשל הוא אחד: השרת מאמת את המשתמש (JWT תקף = מישהו מחובר), אך אינו מרשה אותו מול המשאב המבוקש (האם ה-JWT הזה שייך לבעל הכתובת/כרטיס/הזמנה?).

נמחיש זאת בדוגמה: שומר בכניסה לבניין בודק שיש לך תג כניסה תקף - אך לא בודק לאיזו קומה התג מאשר כניסה. כל תג תקף, מכל קומה ומכל מחלקה, פותח כל דלת.

תבנית כשל זו נפוצה במיוחד ב-APIs שנבנו בשלבים, כאשר מפתחים מוסיפים endpoints חדשים ומניחים ש-Middleware קיים יטפל בהרשאות - אך ה-Middleware בודק רק נוכחות של Token, לא בעלות על המשאב.

הממצא הקריטי ביותר (transaction_id סדרתי) מציג גם כשל ארכיטקטורלי נוסף: הנחת שמי שיודע transaction_id בהכרח ביצע את העסקה. Security through Obscurity - הסתמכות על כך שהמזהה לא ידוע - היא הגנה לקויה, בוודאי כשהמזהה הוא מספר סדרתי.

Mitigations - כיצד לתקן

בדיקת בעלות בכל endpoint: כל בקשה שמכילה מזהה של משאב חייבת להשוות את ה-owner_id של המשאב ל-user_id שמוצה מה-JWT. זה צריך להתבצע בשכבת ה-Business Logic, לא רק בשכבת ה-Authentication.

Authorization Middleware ייעודי: לא להסתפק ב-Authentication Middleware. לבנות שכבת הרשאות שמקבלת גם את המשאב המבוקש ובודקת בעלות, ולא רק נוכחות של Token.

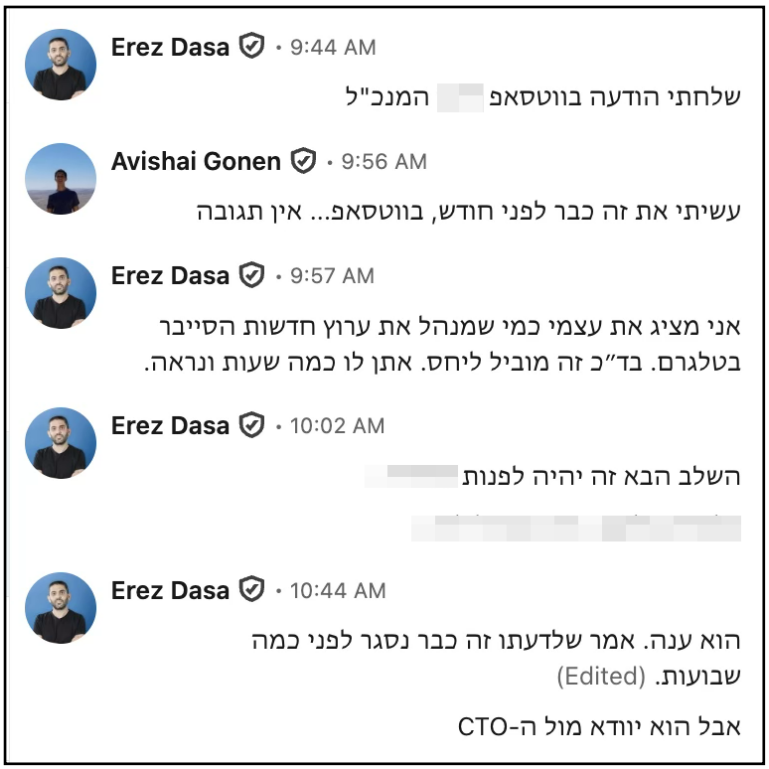
Tenant Isolation: בפלטפורמות Multi-Tenant כמו הספק, JWT של אפליקציה X לא אמור לעבוד מול נתונים של אפליקציה Y. כל JWT חייב לכלול Claim של ה-Tenant, ולכל endpoint לוודא שה-Tenant תואם.

אימות ownership של ransaction_idT: לפני שיוך כרטיס לחשבון, הספק צריך לוודא מול שירות הסליקה שה-transaction_id שייך לאותו משתמש. למשל, על ידי השוואת כתובת האימייל בין ה-תשובת שירות הסליקה לנתוני החשבון.

מזהים אקראיים: להחליף מזהים סדרתיים (כמו transaction_id) ב-UUIDs אקראיים. זה לא מרפא היעדר בדיקת הרשאות, אך מקשה משמעותית על ניחוש.

גילוי נאות ותגובת החברה

לאחר גילוי הממצאים, ניסיתי ליצור קשר עם החברה האחראית במספר ערוצים - לינקדין, וואטסאפ ואימייל - ולא קיבלתי מענה. לאחר כחודשיים של ניסיונות, פניתי ל**ארז דסה**, מנהל ערוץ הטלגרם "חדשות סייבר" (מעל 35,000 מנויים) ואחת הדמויות הבולטות בקהילת הסייבר הישראלית, וביקשתי את עזרתו בהעברת הדיווח. ארז פנה ישירות אל המנכ"ל - ומסתבר שהחברה קיבלה את הדיווחים שלי וסגרה, תוך כדי התעלמות מוחלטת ממני. תודה לארז על המענה המהיר והנכונות לעזור.



מבחינת ציר הזמן - החולשות תוקנו לפני פרסום המאמר, כך שהמידע הזה מפורסם בהתאם לעקרונות ה-Responsible Disclosure.

בנימה אישית, מבאס שהייתי צריך לרדוף אחרי החברה במשך חודשים ולערב גורמים חיצוניים כדי שהדיווח יגיע למנכ"ל, וגם אז לא קיבלתי התייחסות ישירה. כל מה שרציתי היה לעזור ולמנוע חשיפה של כתובות, פרטים אישיים ושימוש לרעה בכרטיסי אשראי.

מעבר לפגיעה האפשרית במשתמשים, אירוע כזה היה עלול לחשוף את החברה לתביעות אזרחיות ולהליכי אכיפה מצד הרשות להגנת הפרטיות, בין היתר בהתאם לחוק הגנת הפרטיות ולתקנות אבטחת המידע בישראל. בסופו של דבר, ניסיתי למנוע נזק משמעותי גם למשתמשים וגם לחברה עצמה.

סיכום

במאמר הזה עברנו על תהליך מלא של בדיקת אפליקציית מובייל - מהקמת סביבת עבודה עם Genymotion ו-Burp Suite, ועד לניצול שרשרת של חולשות Broken Access Control ו-IDOR.
ראינו כיצד כשל ארכיטקטורלי אחד - היעדר בדיקת בעלות על משאבים - מאפשר קריאת וכתיבת כתובות, גניבת טוקני כרטיסי אשראי באמצעות transaction_id סדרתי, ודלף מידע אישי מלא מתשובות IDOR על הזמנות.

הנקודה המרכזית אינה החולשה הבודדת, אלא היקף ההשפעה שלה. כשפלטפורמת White-Label משותפת - קוד, API ובסיס נתונים - נפגעת, כל הלקוחות שלה נפגעים. מעל מאות עסקים ומיליוני משתמשים חשופים לאותה חולשה.

המסקנה המרכזית: Authentication ו-Authorization הם שני דברים שונים. לוודא שמשתמש מחובר - זה הצעד הראשון. לוודא שהוא מורשה לגשת למשאב הספציפי - זה הצעד שנשכח לעתים קרובות מדי.

על המחבר

קוראים לי אבישי גונן, אני חוקר סייבר ובזמני הפנוי אוהב לעשות CTF-ים, Bug Bounty וסתם מחקרים עצמאיים באינטרנט. יש לי מעל ל-30 CVE-ים שרשומים על שמי שמצאתי בזמן עבודתי ב-Pluto Security בתור חוקר חולשות. מוזמנים להעיף מבט באתר שלי שמכיל את החולשות שמצאתי, וכן writeup-ים לאתגרים שעשיתי בעיקר מהעולמות של וב, מובייל ובינארי:
https://avishaigonen123.github.io

מקורות מידע

- OWASP Top 10 2021 - A01: Broken Access Control - https://owasp.org/Top10/A01_2021-Broken_Access_Control
- PortSwigger Web Security Academy - IDOR - https://portswigger.net/web-security/access-control/idor
- Genymotion - Android Emulator - https://www.genymotion.com/
- Burp Suite - PortSwigger - https://portswigger.net/burp
- PortSwigger - Configuring an Android device to work with Burp Suite - https://portswigger.net/burp/documentation/desktop/mobile/config-android-device
- The documentation of the transaction process at tranzila.com - https://tranzila.com/tokenization.html