digital whisper
Technologic Papers

Trust, but verify

מאת איתמר נגינסקי כהן

הקדמה

כמעט כל מערכת Web מודרנית בנויה משיחה מתמשכת בין שני צדדים: ה-Client מציג למשתמש את הממשק, והשרת מנהל את המידע והלוגיקה שמאחוריו. ה-Client מדווח שהמשתמש השלים פעולה, שהאירוע התרחש בזמן מסוים או שמגיע לו תגמול והשרת מקבל את הדיווח ומחליט כיצד לעדכן את מצבו של החשבון.

אבל מה קורה כאשר השרת אינו רק מקבל את הדיווח - אלא גם מאמין לו מבלי לוודא?

העיקרון **Trust, but Verify**, שמגיע במקור מרוסית "*Доверяй, но проверяй*" הוא ביטוי רוסי סובייטי שהתפרסם במערב במהלך שנות ה-80, לאחר שנשיא ארצות הברית רונלד רייגן נהג להשתמש בו במסגרת המשא ומתן עם ברית המועצות על פיקוח ובקרת נשק. הרעיון היה פשוט: אפשר להגיע להסכמות ואף לתת אמון בצד השני, אך האמון אינו יכול להחליף מנגנוני בדיקה עצמאיים.

בעולם אבטחת האפליקציות, המשמעות מחמירה יותר. למעשה כאשר מדובר במידע שמגיע מה-Client, נקודת המוצא לא צריכה להיות שהמידע אמין עד שיוכח אחרת, אלא שכל נתון חייב לעבור אימות לפני שהוא משפיע על מצב המערכת.

במהלך מחקר שביצעתי על Duolingo מצאתי שני מנגנונים שהמחישו היטב את הבעיה. הראשון אפשר להשפיע על מנגנון ה-Streak באמצעות נתוני זמן שלא אומתו. השני אפשר לבצע שוב ושוב פעולה הקשורה לקבלת Gems, אף שהאירוע הלוגי שאותו היא ייצגה היה אמור להתרחש פעם אחת בלבד.

מטרת המאמר אינה להציג מדריך להשגת Streak או Gems, אלא להשתמש ב-Duolingo כמקרה בוחן (Case Study). לכן במהלך המאמר אחליף שמות של אנד פוינטים, מטרת השינוי היא כדי להמחיש את אופי החולשות ואת הכשל שמאחוריהן תוך צמצום האפשרות לנצל את המידע על המערכת מאחר וחולשות אלה עדיין פעילות.

לפני שמתחילים, מי בכלל שולט ב-Client?

כאשר משתמש פועל דרך ממשק המשתמש הרשמי, קל לחשוב שה-Client הוא חלק מהמערכת שעליו ניתן לסמוך. הרי הממשק מגביל את הפעולות שאפשר לבצע: כפתורים מופיעים רק במצבים מסוימים, שדות מסוימים אינם ניתנים לעריכה, ופעולות מתבצעות לפי סדר שהוגדר מראש.

אלא שמבחינה אבטחתית ממשק המשתמש אינו דבר מגביל מכיוון שה-Client רץ על מכשיר שנמצא בשליטת המשתמש. בדפדפן ניתן לצפות בקוד ה-JavaScript ובבקשות הרשת באמצעות כלי הפיתוח (DevTools). באפליקציות ניתן לנתח את התעבורה, לבחון את מבנה הבקשות ולעיתים גם לשנות את אופן פעולת התוכנה באמצעות תוכנות כגון: Burp suite, Caido, Fiddler ו-mitmproxy. גם אם המשתמש לא משנה את האפליקציה עצמה, הוא עדיין יכול לנסות לשלוח לשרת בקשה שונה מזו שהממשק הרשמי היה יוצר.

לכן, העובדה שהממשק אינו מאפשר לבחור תאריך מהעבר אינה מוכיחה שהשרת דוחה תאריך כזה. העובדה שכפתור לקבלת תגמול נעלם לאחר לחיצה אחת אינה מוכיחה שלא ניתן לשלוח שוב את אותה הבקשה ולקבל שוב את התגמול. הגבלה שקיימת רק בצד ה-Client היא הגבלת הממשק, לא מנגנון אבטחה.

ניקח לדוגמה מערכת דמיונית המעניקה למשתמש עשרה מטבעות לאחר השלמת משימה. הממשק מציג כפתור, ולאחר הלחיצה הכפתור נעלם מממשק המשתמש. מבחינת המשתמש הרגיל, הפעולה הסתיימה ולא ניתן לבצע אותה שוב.

אבל היעלמות הכפתור לא משנה שום דבר בשרת, אם הבקשה שנשלחה בעת לחיצה על הכפתור נראית כך:

```
POST /api/rewards/claim { "rewardId": "daily-task" }
```

השאלה החשובה היא לא האם הכפתור עדיין מוצג בממשק המשתמש, אלא כיצד השרת מגיב אם אותה בקשה מתקבלת פעם נוספת. אם בכל בקשה השרת מוסיף מחדש עשרה מטבעות לחשבון, המערכת למעשה סומכת על כך שה-Client לא יבקש את אותו התגמול פעמיים.

זו הנחה בעייתית. השרת צריך לדעת בעצמו האם התגמול כבר מומש, ולא להסיק זאת מהתנהגות האפליקציה.

אותו עיקרון חל גם על מידע הקשור לזמן. נניח שה-Client שולח לשרת דיווח על השלמת שיעור:

```
POST /api/start/session {
    "completed": true,
    "startTime": "2026-07-06 16:00", #YYYY-MM-DD HH:mm
    "endTime": "2026-07-06 16:05"
}
```

השדות startTime ו-endTime יכולים לשמש לצורכי סטטיסטיקה, מדידת משך הפעילות או הצגת היסטוריית הלימוד. אולם ברגע שהשרת משתמש בהם כדי לקבוע באיזה יום נרשמה הפעילות, הם מפסיקים להיות מידע תיאורי והופכים למידע בעל סמכות ומשמעות. במצב כזה ה-Client לא רק מדווח על האירוע: הוא גם משפיע על האופן שבו השרת מפרש אותו.

היכרות עם דואולינגו

דואולינגו היא אחת מאפליקציות לימוד השפות הגדולות בעולם עם למעלה מ-100 מיליון משתמשים פעילים בחודש. חלק גדול מהצלחתה נובע ממנגנוני המשחוק המשולבים כמעט בכל פעולה שהמשתמש מבצע. המטרה של מערכת שכזאת היא לעודד את המשתמש לחזור לאפליקציה ולהרגיש התקדמות גם כאשר תהליך הלמידה עצמו ארוך (שפה חדשה וכדומה).

שני המנגנונים הרלוונטיים למאמר הם ה-Streak וה-Gems:

- Streak - ה-Streak מייצג את מספר הימים הרצופים שבמהלכם המשתמש השלים פעילות לימודית. בכל יום שהמשתמש מבצע שיעור הרצף גדל ביום נוסף. אם המשתמש לא מבצע את השיעור בזמן הרצף עלול להישבר (לא תמיד מכיוון שיש מה שנקרא Streak freeze אבל זה לא רלוונטי למאמר)
- Gems - ה-Gems הם מטבע וירטואלי המשמש בתוך דואולינגו לרכישת פריטים מהחנות. המשתמש יכול לקבל Gems באמצעות השלמת משימות, פתיחת תיבות, עמידה ביעדי למידה, קבלת תגמולים אחרים.

ניתוח טכני

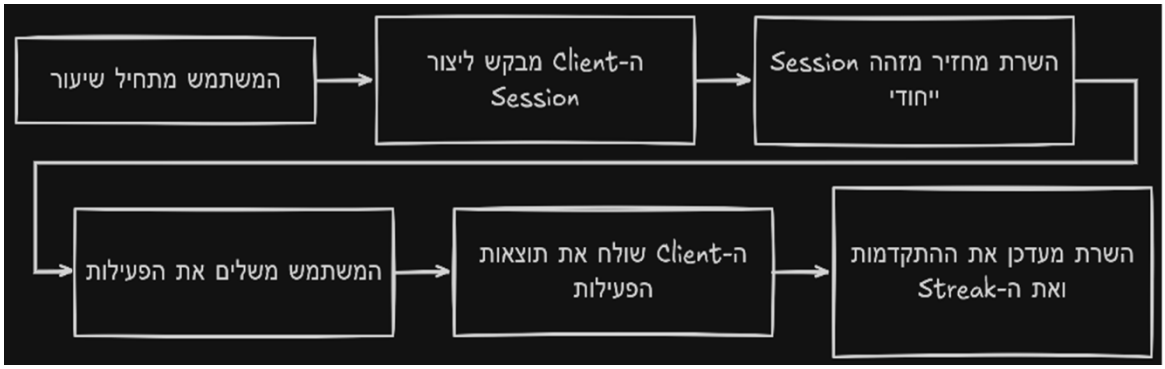
בשלב הראשון בחנתי את התקשורת בין ה-Client לבין שרתי דואולינגו. המטרה לא הייתה לקרוא את כל התעבורה, אלא לזהות את הבקשות שנשלחו כאשר התרחשו פעולות מסוימות. ספציפית: התחלת שיעור, השלמת שיעור וקבלת תגמול (Gems).

בהתחלה השתמשתי בכלי פרוקסי המאפשר לצפות בבקשות HTTP ובתגובות השרת. אני אישית מאוד אוהב את הכלי Caido ולכן השתמשתי בו גם במהלך המחקר הזה אך ישנם כלים נוספים כגון Burp suite, mitmproxy, Fiddler.

מיפוי תהליך התחלת והשלמת שיעור

לפני שניתן לבדוק את מנגנון ה-Streak צריך להבין כיצד פעילות רגילה נראית מנקודת מבטו של השרת.

בצורה מופשטת התהליך נראה כך:



במהלך שליחת בקשת השלמת הפעילות ה-Client שלח בקשה שכללה מידע על השיעור, תוצאתו וזמני ההתחלה והסיום שלו.

בצורה מופשטת הבקשה נראית כך:

```
PUT /api/v1/Session/Complete/
<SESSION_ID>
Authorization: Bearer <JWT_TOKEN>
Content-Type: application/json
```

והשדות התואמים:

```
{
  "completed": true,
  "client_start_time": 946684800,
  "client_end_time": 946684860,
  "failed": false,
  "hearts_lost": 1
}
```

כפי שאפשר לראות בשדות התואמים של הבקשה ה-Client דיווח לשרת לא רק מתי שהפעילות הסתיימה, אלא גם מתי היא התחילה וגם מתי היא הסתיימה. בשלב הזה עדיין לא הייתי בטוח האם השרת משתמש בשדות הללו, האם זה לצורך אנליטיקה בלבד או שהם משפיעים גם על הלוגיקה של ה-Streak.

כדי לבדוק זאת, היה צריך לנסות לשנות משתנה אחד בכל פעם ולבדוק מה התוצאה.

הממצא הראשון - כאשר ה-Client קובע מתי הפעילות התרחשה

בתור התחלה השלמתי שיעור בצורה רגילה עם חשבון חדש עם Streak מאופס. כצפוי, ה-Streak גדל ביום אחד. לאחר מכן חזרתי על אותו Flow אך הפעם שיניתי את ערכי הזמן (client_start_time ו-client_end_time) של הבקשה ליום שלפני.

הציפייה הראשונית שלי הייתה שהשרת יתעלם מהזמן שהגיע מה-Client, או שלכל הפחות ישווה אותו לזמן הנוכחי של השרת וידחה ערכים שאינם הגיוניים. בפועל התברר שנתוני הזמן השפיעו על חישוב ה-Streak ובאמת כתוצאה התווסף לי יום אחד ל-Streak.

כלומר השרת לא השתמש בערכים רק כמידע תיאורי על השיעור או רק למטרת אנליטיקה. הוא איפשר להם להשפיע על עובדה שהייתה אמורה להיקבע על ידי המערכת (היום שאליו השיעור משויך). ה-Client יכול לדווח כי פעילות התחילה בשעה מסוימת. אבל אין סיבה להניח שהדיווח נכון רק משום שהגיע מהאפליקציה הרשמית. משתמש יכול לשנות את הבקשה, לשלוח אותה מחדש או לבנות Client שונה לגמרי.

שורש הכשל

שורש הכשל הוא לא עצם קיומם של שדות זמן בבקשה. במקרים רבים יש צורך לגיטימי לקבל מה-Client מידע שכזה, למשל כדי למדוד כמה זמן המשתמש הקדיש לשיעור. הבעיה נוצרת כשאותו מידע משמש כמקור הסמכותי היחיד להחלטה שמשפיעה על מצב המשתמש.

במימוש בטוח יותר, השרת יכול לשמור בעצמו את זמן יצירת ה-Session ולקבל את זמן סיום ה-Session על פי מתי בקשת סיום השיעור מתקבלת. אם יש צורך לקבל גם את זמן ה-Client אפשר לשמור אותו בשביל אנליטיקה, אבל לא להסתמך עליו בלבד כדי לשנות את ה-Streak.

הממצא השני - תגמול שניתן לממש יותר מפעם אחת

לאחר בדיקת מנגנון ה-Streak רציתי לנסות לחקור את מערכת ה-Gems. לאחר ששאלתי כמה חברים שפעילים בדואולינגו נאמר לי שהדרך המרכזית להשיג Gems היא דרך משימות (Quests).

כאשר משתמש משלים משימה המזכה אותו ב-Gems, ה-Client מציג כפתור לקבלת התגמול. לאחר הלחיצה הכפתור נעלם, הבקשה נשלחת והיתרה בחשבון מתעדכנת.

בצורה מופשטת הבקשה נראית כך:

```
POST /api/v1/rewards/<REWARD_ID>/
claim
Authorization: Bearer <JWT_TOKEN>
Content-Type: application/json
```

והשדות התואמים:

```
{
    "consumed": true,
    "source":
"activity_completion"
}
```

לאחר שהתגמול (Gems) התקבל, שלחתי שוב את אותה הפעולה באותו חשבון. במימוש בטוח, השרת היה אמור לזהות כי אותו תגמול כבר התקבל. והוא יכול היה לדחות את הבקשה, להחזיר תשובה המציינת שהתגמול כבר מומש או להחזיר את תוצאת הפעולה הקודמת בלי לשנות פעם נוספת את היתרה.

בפועל, הפעולה החוזרת גרמה לקבלת התגמול פעם נוספת. כאשר פעולה כזו ניתנת לחזרה, ניתן לבצע "התקפת שליחה מחדש" של הבקשה ולהגדיל את יתרת ה-Gems בלי להשלים אירוע חדש שמזכה בתגמול.

אידמפוטנט

כדי להבין את הבעיה, רצוי להכיר את המונח אידמפוטנט.

פעולה אידמפוטנט היא פעולה שביצועה פעם אחת או יותר מוביל לאותו מצב סופי.

לדוגמה, בקשה שמגדירה שדה מסוים לערך קבוע יכולה להיות אידמפוטנטית. אם נגדיר את הערך ל-true פעם אחת או עשר פעמים, התוצאה הסופית תישאר זהה.

דוגמה נוספת מחיי היום-יום שלנו היא כפתור ה"עצור" באוטובוס. כאשר הנוסע הראשון לוחץ על הכפתור, הנהג מקבל התראה שעליו לעצור בתחנה הבאה. אם נוסעים נוספים ילחצו על אותו כפתור לפני ההגעה לתחנה, לא יתווספו התראות נוספות ולא תשתנה התוצאה האוטובוס עדיין יעצור פעם אחת בלבד. במילים אחרות, לחיצות חוזרות אינן משנות את מצב המערכת מעבר ללחיצה הראשונה.

לעומת זאת, פעולה שמוסיפה בכל פעם עשרה Gems ליתרה אינה אידמפוטנט. כל שליחה נוספת משנה שוב את מצב החשבון.

לא כל Endpoint חייב להיות אידמפוטנט. עם זאת כאשר הפעולה מייצגת מימוש של תגמול **חד-פעמי**, השרת חייב לאכוף שהאירוע עצמו לא יוכל להתרחש יותר מפעם אחת. ממש לא מספיק שה-Client ישלח את השדה:

```
{
    "consumed": true
}
```

השרת צריך לבדוק האם התגמול כבר נמצא במצב consumed במסד הנתונים. הוא אינו יכול להסתמך על כך שהכפתור נעלם או שהאפליקציה הרשמית אינה שולחת את הבקשה פעם נוספת.

שני באגים - אותו גבול אמון

במבט ראשון, אין קשר בין שינוי הזמן של שיעור לבין קבלת ה-Gems. האחד משפיע על מנגנון המבוסס על תאריכים, והשני משפיע על יתרה של ערך ה-Gems. עם זאת בשני המקרים השרת הסתמך על הנחה דומה.

במקרה של ה-Streak: ה-Client דיווח מתי האירוע התרחש והשרת אפשר לדיווח להשפיע על מצב החשבון ללא בדיקה. במקרה של ה-Gems: ה-Client דיווח כי הוא מבקש לצרוך תגמול חד פעמי, והשרת לא אכף בצורה מספקת את זה שהאירוע יכול להתרחש פעם אחת בלבד.

בשני המקרים הבקשה התקבלה כהוכחה לכך שהפעולה תקינה. ואולם, בקשה אינה הוכחה. היא טענה בלבד שהשרת עדיין צריך לאכוף.

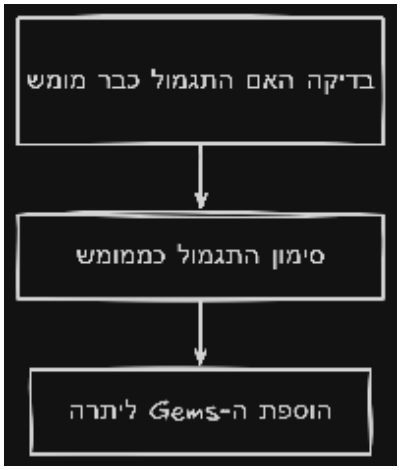
כיצד ניתן היה למנוע את הכשלים?

במנגנון ה-Streak, זמן השרת צריך להיות מקור הסמכות העיקרי השרת יכול ליצור את ה-Session ולשמור את זמן יצירתו, ולאחר השלמת הפעילות לשמור את זמן קבלת הבקשה.

זמן שמגיע מה-Client יכול לשמש רק כמידע משלים. אם קיים פער גדול בין שני מקורות הזמן, ניתן לדחות את הפעולה או להתעלם מזמן ה-Client לצורך חישוב ה-Streak.

בנוסף, במנגנון התגמולים, צריך לשמור על קשר חד ערכי בין המשתמש לבין התגמול שמומש. לדוגמה מסד הנתונים יכול להכיל רשימה שמציינת כי משתמש מסוים כבר מימש תגמול מסוים.

האכיפה צריכה להתבצע כחלק אטומי:



שלושת השלבים צריכים להיחשב פעולה אחת. אחרת, שתי בקשות שמגיעות כמעט באותו הזמן עלולות לעבור את הבדיקה לפני שאחת מהן מספיקה לעדכן את מסד הנתונים.

סיכום

שני הממצאים שהוצגו במאמר ממחישים עיקרון אחד: אסור להסתמך על מידע שמגיע מה-Client מבלי לאמת אותו בצד השרת. בין אם מדובר בנתוני זמן, במימוש תגמול או בכל פעולה אחרת, השרת הוא זה שצריך להיות מקור הסמכות ולאכוף את כללי המערכת.

במהלך המחקר מצאתי גם ממצאים נוספים שהתבססו על כשלים תכנוניים דומים, ובהם דרכים לקבל הטבות שלא נועדו להתקבל (ובמקור עולות כסף), לבצע עקיפות של מנגנוני "אני לא רובוט" ועוד, בחרתי שלא לפרטם במאמר משום שמטרתו להסביר את העיקרון, ולא להסביר כיצד לנצל את המערכת.

לאחר השלמת המחקר ניסיתי ליצור קשר עם דואולינגו ולדווח באופן מסודר דרך כתובת המייל Security@Duolingo.com על כל הממצאים, אך לצערי לא קיבלתי מענה. אני מקווה שהמאמר יתרום להעלאת המודעות לחשיבותו של העיקרון הפשוט אך הקריטי: **Trust, but Verify**.

לקוראים שנהנו מהמאמר או שמעוניינים להעמיק בתחום, אני ממליץ בחום על הפודקאסט Critical Thinking, העוסק בעיקר בWeb Application Security ובנושאים כמו Bug Bounty וטכניקות למציאת חולשות.

על המחבר

איתמר נגינסקי כהן, בן 17, מתעניין בתחום ה-Web-Exploitation. כיום שואף להשתלב במסלול גאמא בצה"ל ולהמשיך להעמיק בתחומי אבטחת המידע מחקר חולשות ופיתוח מאובטח.

Github: https://github.com/itamar1337 (לא פעיל)

Discord: itamar1338 (https://discord.com/users/1115351644922708070)

מקורות מידע

- https://restfulapi.net/idempotent-rest-apis/
- https://medium.com/@reetesh043/rest-api-design-what-is-idempotency-18218e1ff73c
- https://developer.mozilla.org/en-US/docs/Web/HTTP/Reference/Headers/Idempotency-Key
- https://he.wikipedia.org/wiki/%D7%90%D7%99%D7%93%D7%9E%D7%A4%D7%95%D7%98%D7%A0%D7%98
- https://he.wikipedia.org/wiki/%D7%A4%D7%A2%D7%95%D7%9C%D7%94_%D7%90%D7%98%D7%95%D7%9E%D7%99%D7%AA
- https://investors.duolingo.com/static-files/aab30d54-eb91-422e-b365-c03859fea85c
- https://duolingo.fandom.com/wiki/Streak
- https://duolingo.fandom.com/wiki/Gem
- https://duolingo.fandom.com/wiki/Quests
- https://www.caido.io/
- https://portswigger.net/burp/communitydownload
- https://www.mitmproxy.org/
- https://www.telerik.com/download/fiddler