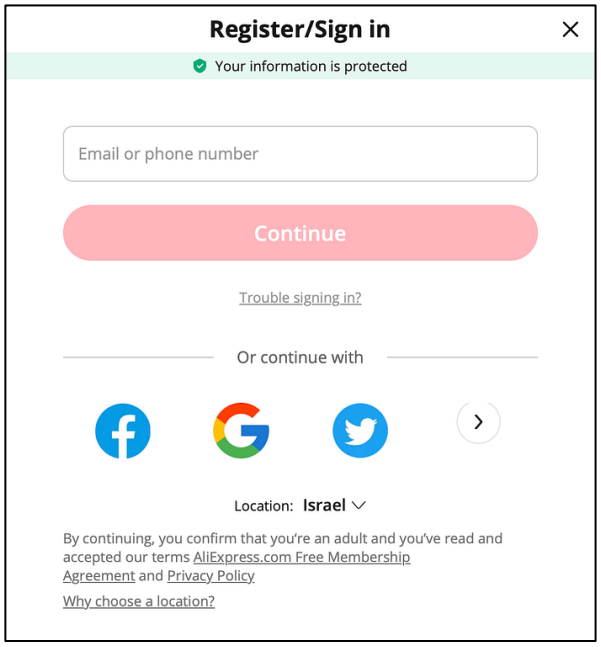
OAutheWHAT2.0

מאת סופיה שביקובסקי

הקדמה

כמה פעמים קרה לכם שנכנסתם לאתר רנדומלי באינטרנט והבנתם ששכחתם את הסיסמה לחשבון שלכם, אז עכשיו, במקום לגלול בנחת בטרמינל איקס או ב-Medium, אתם צריכים **שוב** להחליף סיסמה, בדיוק כמו בפעם שעברה שנכנסתם לאתר הזה. הסאגה הזאת לא נגמרת, מי זוכר את הסיסמה שלו בכלל בימינו?!

אז כן, הבעיה הזו הייתה רלוונטית המון זמן, עד שיום אחד פשוט הופיע הכפתור הבא, ומאז הכל השתנה.



אבל רגע – מה זה בכלל ואיך זה עובד?

בעיה ופתרון

תחשבו כך: יש לכם חשבון יציב באינטרנט, אולי זה גוגל, פייסבוק או אינסטגרם. אתם נמצאים בו קבוע ויודעים לעקוב אחרי כל מה שמתרחש בו. בתוך החשבונות האלו נמצא המידע המדויק עליכם כמו רשימת חברים, תחומי עניין, תמונות, מסמכים, הרשאות וכו'.

כדי שלא תצטרכו לעשות דופליקציה של אותו המידע בכל אתר שאתם נרשמים אליו, נוצר Framework שמאפשר התחברות ואימות בתצורת SSO וכן משיכת המידע הרלוונטי ישירות מהחשבון שלכם. הדבר הזה מתאפשר באמצעות טכנולוגיית OAuth, פרוטוקול שמאפשר לתת לאפליקציה גישה מוגבלת ומבוקרת למשאבים שלך, בלי לחשוף מידע פרטי כמו סיסמה.

במאמר הזה אני אתייחס יותר לשימוש ב-OAuth כפונקציה המשמשת את החשבונות לאימות, בתצורת SSO (Single Sign On), ולחולשות שיכולות להמצא בתצורה הזו.

OAuth

OAuth נועד לאפשר למשתמשים לתת גישה מוגבלת למשאבים שלהם באינטרנט לאפליקציות צד שלישי, מבלי לחשוף סיסמה, הפרוטוקול הזה פותח לראשונה ב-2006 על ידי צוותים מחברות כמו גוגל וטוויטר, כתגובה לצורך לפתרון של הבעיה שדיברנו עליה קודם, יותר ויותר שירותים רצו לאפשר אינטגרציה עם תוכנות חיצוניות, אבל בצורה מבוקרת.

הגרסה הנפוצה ביותר היא OAuth2.0, שפורסמה ב-2012. היא שינתה לגמרי את הדרך בה מתבצעת אותנתיקציה באינטרנט, במקום להעביר סיסמאות בין שירותים, המשתמש קיבל token (token) זמני שמאפשר לאפליקציה לבצע פעולות מסויימות בלבד ובזמן מוגבל.

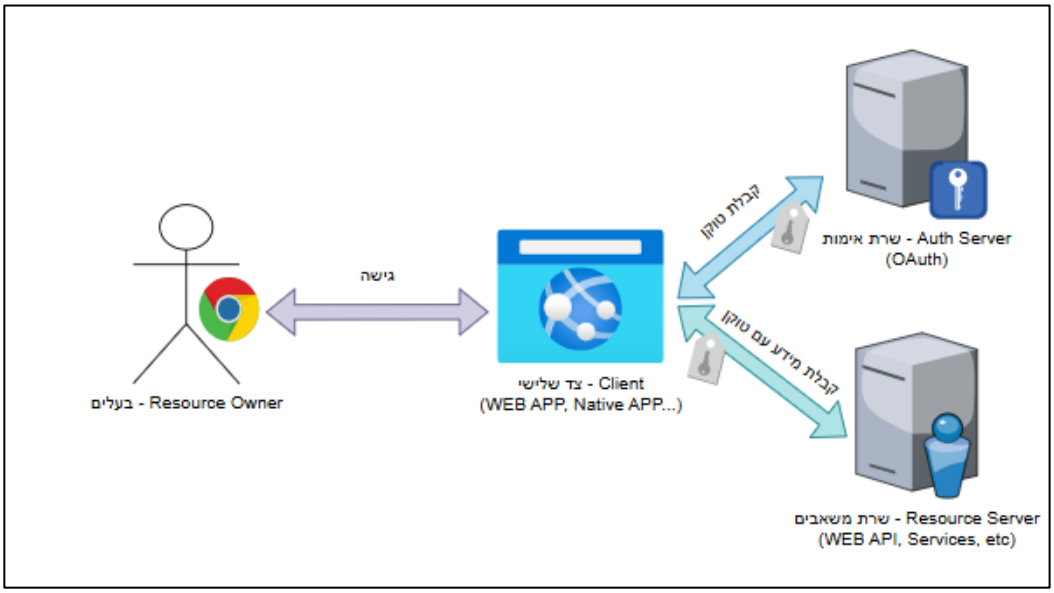
OIDC

בגלל שבמקור ה-Framework של OAuth היה שימושי לאותוריזציה בלבד, זאת אומרת אך ורק לגישה למשאבים ספציפים, ולא לצורך אותנטיקציה - אימות, נוצרה הרחבה שכן תאפשר לבצע את האימות באמצעות ה-OAuth והיא OIDC, בשמה המלא OpenID Connect, שמספקת שכבת אימות. OIDC מתבסס על OAuth ומוסיף אליו שימוש בtoken-ים מסוג JWT, סט scopes והרחבת מידע הרלוונטי למשתמש. בזכות ההרחבה הזו ההתחברות באמצעות SSO נהייתה הרבה יותר פשוטה ונפוצה באתרים ברחבי האינרנט, אבל יחד עם זאת, גם תצורת OIDC וגם OAuth מהווים חוליה שיכולה להיות מנוצלת אם לא מוגדרת היטב.

מבנה של ה-Framework

צדדים עיקריים (ACTORS):

- בעלים (Resource Owner) - המשתמש שמנהל את המידע והמשאבים (לדוגמא חשבון משתמש ב-GOOGLE).
- צד שלישי (Client Application) - הגוף שמבקש גישה למשאבי משתמש כדי לספק פונקציונליות כלשהי (לדוגמא "Sign in with GOOGLE").
- שרת אימות (Authorization Server) - הגוף שאחראי על אימות המשתמשים והנפקת Access Token, או Authorization Code.
- שרת משאבים (Resource Server) - שרת שאחראי על שמירת מידע של משתמשים וגישה אליהם דרך ה-token.



בעצם איך זה עובד בפועל:

בעלים (Resource Owner) הם המשתמש שמחזיק את החשבון או המידע הספציפי והרלוונטי. במקרה שלנו זה אני, משתמשת שיש לה חשבון Gmail שמכיל את רשימת אנשי הקשר שלי, מיילים, פרופיל וכו'.

אני רוצה להתחבר לאתר קניות כלשהו באינטרנט (Client Application), שמציע לי אפשרות להתחבר עם גוגל, הוא לא רוצה את הסיסמה שלי ולא שאמלא ידנית פרטים, אלא רק הרשאה לקרוא את הכתובת מייל שלי כדי ליצור חשבון או להתחבר אליו.

השרת שינהל את האוטנתקציה (הזדהות) שלי מול אותו האתר צד שלישי הוא השרת אימות (Authorization Server), לדוגמא שרת של GOOGLE שמבצע לוגין ומחזיר Access Token.

עם ה-token שקיבלנו מהשרת אימות נוכל לגשת למידע בשרת משאבים (Resource Server), שיכול להיות לדוגמא ה-API של Gmail, שמחזיק את הרשימת אנשי קשר שלי. האתר קניות יקבל את המידע הרלוונטי אך ורק אם ה-token שהוא קיבל מהשרת אימות תקף.

token-ים וסוגי ה-Grant Types

חלק מהותי בכל התהליך של קבלת גישה עם ה-OAuth הוא קבלת token, הדבר הזה מתבצע באמצעות ה-Grant Type, שזוהי בעצם "סוג דרך" שהאפליקציה (Client Application) מבקשת דרכה גישה לנתונים של המשתמש בשרתי המשאבים (Resource Server).

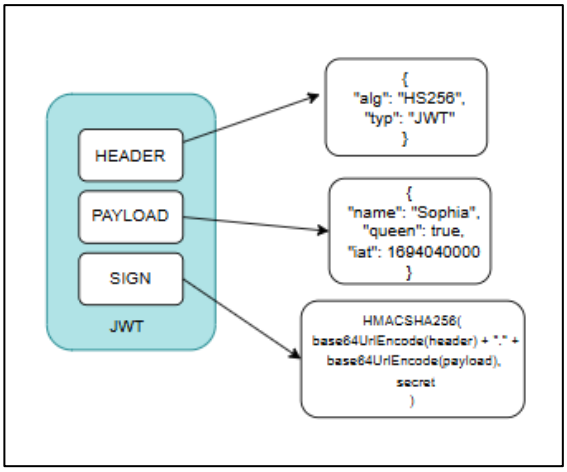
ה-Grant Type קובע את סדר הצעדים בתהליך, זאת אומרת מה האפליקציה שולחת, איזה token צריכה, איך הם מועברים ועוד. כמובן שזה משפיע ישירות על רמת האבטחה של ההחלפת מידע, כל OAuth Server חייב להיות מוגדר לתמוך ב-Grant Type מסויים, והאפליקציית צד שלישי בוחרת באיזה סוג היא רוצה להשתמש לצורך קבלת הרשאות.

סוג token-ים ב-Oauth:

בתהליך האימות עם ה-OAuth מופיעים מספר סוגי token-ים מרכזיים.

Access Token - token עם טווח חיים קצר שמאפשר ללקוח לבצע קריאות API לשרת משאבים. הוא מוגבל בזמן (לדוגמא למספר שעות), מקושר ל-SCOPE ספציפי (הרשאות ספציפיות שאפשר לקבל), בדרך כלל JWT.

- **JWT** או JSON Web Token הוא token דיגיטלי שמשמש להעברת מידע בין צדדים בצורה מאובטחת.



ה-JWT שימושי ונוח מכיוון שהוא מאפשר אימות (אותנטיקציה) כדי לזהות את המשתמש הספציפי והזדהות (אותוריזציה), בכדי לקבוע את ההרשאות שהמשתמש זכאי להן ואיזה פעולות הוא מורשה לעשות.

Refresh Token - token עם תכולת חיים ארוכה, שמאפשר ללקוח לקבל Access Token חדש מבלי שהמשתמש יצטרך להזדהות שוב. ה-token הזה נשמר בצד שרת ומאפשר חידוש token-ים בצורה מאובטחת.

Authorization Code - קוד חד פעמי שמתקבל אחרי שהמשתמש נותן הסכמה לאתר צד שלישי לגשת לנתונים שלו. בעצם הלקוח שולח את הקוד לשרת הרשאות יחד עם ה"סודות" של המשתמש, כמו Client_id ו-Client_secret, ובתמורה מקבל Access Token. זו דרך יחסית מאובטחת יותר, כי ה-Access Token לא עובר דרך הדפדפן.

סוגי Grant Types:

Authorization Code Grant Type:

סוג ה-grant הנפוץ ביותר באפליקציות Server-Side. היתרון שלו הוא רמת אבטחה הגבוהה, כי כל הנתונים הרגישים, כגון Access Token ופרטי המשתמש לא עוברים דרך הדפדפן אלא דרך ערוץ מאובטח בין שרתים.

ה-FLOW:

Authorization Request

הלקוח שולח בקשה לשרת ההרשאות לקבל הרשאה מהמשתמש, כולל פרמטרים כמו id, scope, state ועוד.

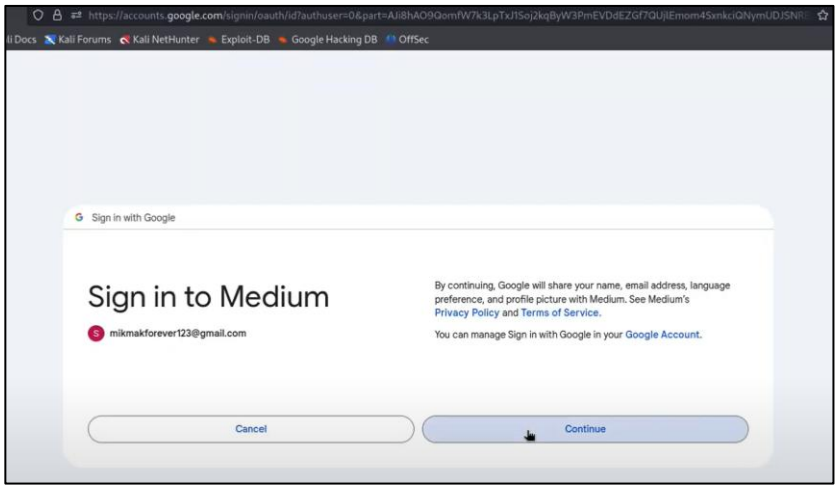
```
GET /authorization?client_id=12345 &redirect_uri=https://client-app.com/callback
&response_type=code
&scope=openid%20profile
&state=ae13d489bd00e3c24
HTTP/1.1
```

פרמטרים:

- Client id – מזהה של הצד השלישי אל מול השרתי אימות.
- Response type – מגדיר לאיזו תשובה מצפה הצד השלישי (קוד/token).
- Redirect uri - כתובת שאליה המשתמש יופנה אחרי התחברות או אחרי קבלת ההרשאות.
- Scope - רשימת הרשאות שהצד השלישי יכול לקבל על המשתמש, זה מגדיר איזה נתונים המשתמש מנגיש.
- State – ערך ייחודי שמקושר לסשן הנוכחי של המשתמש, זוהי הגנה מפני מתקפות CSRF.

User Login & Consent

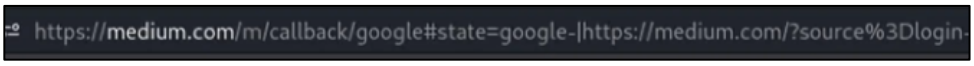
המשתמש נכנס לחשבון שלו ומסכים לאפשר ללקוח לגשת לנתונים שלו לפי ה-Scope שהוגדר. ברגע שהמשתמש אישר, האישור יהיה תקף וקיים כל עוד הסשן שלו פעיל.



Authorization Code Grant

אם המשתמש מסכים, הדפדפן מופנה לכתובת Callback, או בשמה השני – Redirect_URI, יחד עם ה-Authorization Code שקיבל וה-State.

```
GET /callback?code=a1b2c3d4e5f6g7h8&state=ae13d489bd00e3c24 HTTP/1.1
```



Access Token Request

עכשיו כשקיבלנו את הקוד, צריך להחליף אותו לtoken אמיתי. הלקוח שולח בקשה לשרת האימות כדי להחליף את הקוד ולקבל token. באמצעות בקשת POST מהשרת של האתר לקוח (אתר צד שלישי) לשרת אימות , המידע על המשתמש, כמו ה-Client_secret והקוד אימות עצמו עוברים ב"ערוץ מאובטח".

ערוץ מאובטח, משמעותו היא שהתקשורת הזאת מתבצעת Server-to-server, משמע הבקשה לא עוברת דרך הדפדפן, אלא בקשת HTTPS מתקבלת ישירות בשרת, כך שהמידע הרגיש לא נחשף בשום שלב, גם אם יאזינו לתעבורה של הדפדפן.

```
POST /token HTTP/1.1
Host: oauth-authorization-server.com
client_id=12345&client_secret=SECRET &redirect_uri=https://client-app.com/callback
&grant_type=authorization_code
&code=a1b2c3d4e5f6g7h8
```

Access Token Grant

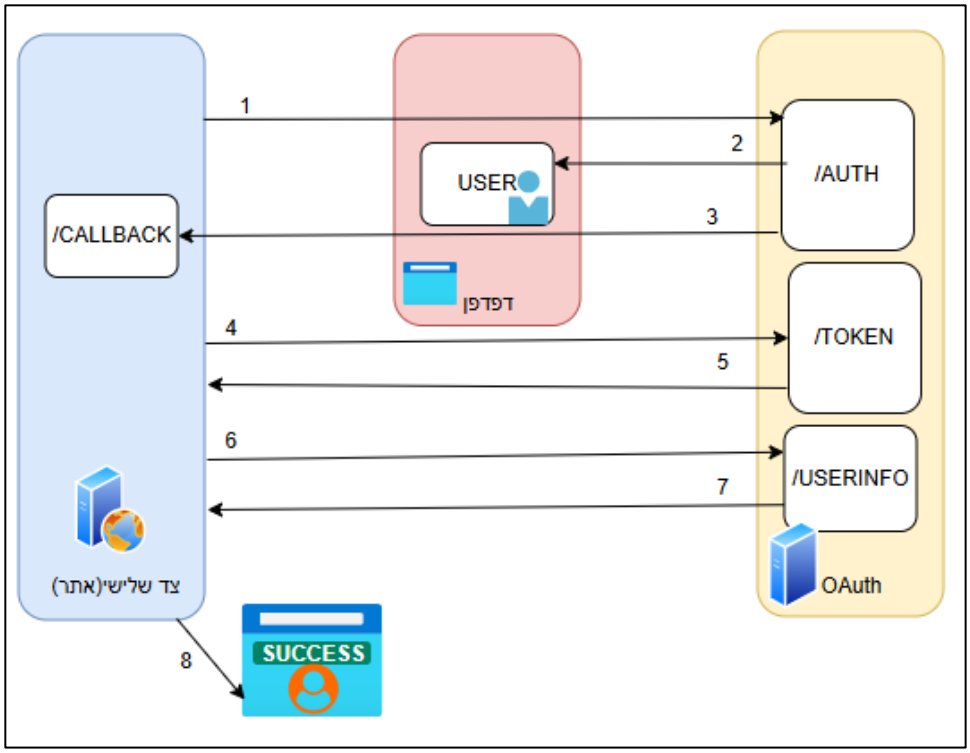
השרת מאשר ומחזיר Access Token לאתר צד שלישי (הלקוח), כולל ה-Scope.

API Call

הלקוח משתמש ב-Access Token כדי לגשת לנתונים של המשתמש בשרתי משאבים.

Resource Grant

שרת המשאבים מאשר את ה-token ושלח את הנתונים המבוקשים ללקוח.



Implicit Grant Type:

סוג ה-Grant שמתאים בעיקר ל-SPA, שזה Single Page App, או אפליקציות לא מאוד מתוחכמות ללא שטח אחסון. היתרון כאן הוא פשוט, אבל הפשטות הזו מגיעה גם עם קשיים אבטחתיים, מכיוון שכאן ה-token מגיע ישירות, ללא קוד "מתווך" כלשהו. הכל בדפדפן.

Authorization Request

הצד השלישי מבקש הרשאות מהמשתמש, אך הפעם הבקשה מוגדרת מראש על Response_Type=Token.

User Login and Consent

המשתמש מתחבר ומאשר לאתר צד שלישי לקבל את ההרשאות. התהליך כאן זהה כמו ב-Grant קודם.

Access Token Grant

במידה והמשתמש אישר את התהליך, הצד השלישי מקבל ישירות את ה-token בתוך ה-URL.

כאן ניתן לראות בפירוש שה-token חוזר אל הדפדפן ישירות מתוך ה-Callback.

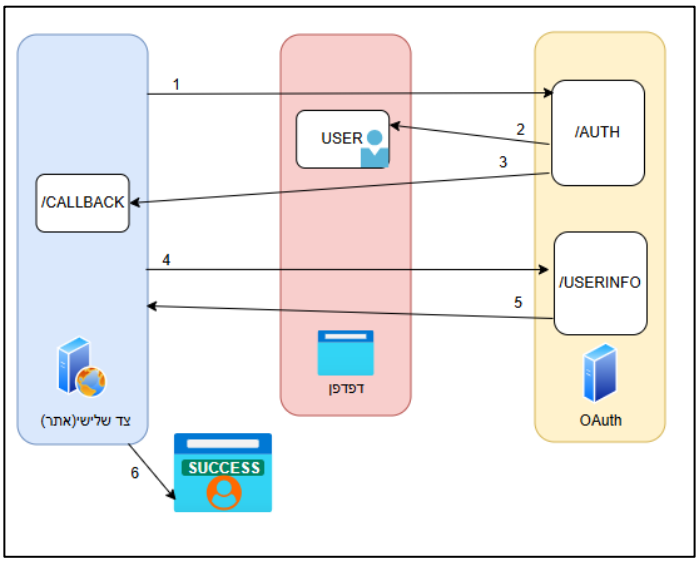
```
GET /callback#access_token=z0y9x8w7v6u5
&token_type=Bearer
&expires_in=5000
&scope=openid%20profile
&state=ae13d489bd00e3c24
HTTP/1.1
```

API Call

הצד השלישי מפענח את ה-token ומשתמש בו לצורך קבלת מידע מהמשתמש.

Resource Grant

שרת המשאבים מאשר את ה-token ושולח את הנתונים המבוקשים ללקוח.



:OpenID Connect

ה-OIDC משתמש ב-Oauth Authorization Code Flow, אבל מוסיף עליו שכבת אימות של זהות. ההבדל המרכזי, בנוסף ל-Access Token, הלקוח מקבל גם ID token שהוא JWT חתום עם Claims על המשתמש, שזה יכול להיות השם שלו, מייל, UID, מתי משתמש התחבר לאחרונה ועוד...

Authorization Request

כמו קודם, אך הפעם הבקשה תכלול Scope של OpenID.

```
GET /authorization?client_id=12345
&redirect_uri=https://client-app.com/callback
&response_type=code
&scope=openid%20profile%20email
&state=ae13d489bd00e3c24 HTTP/1.1
```

השוני כאן, לעומת התהליך הרגיל, הוא הוספת scope.

User Login & Consent

המשתמש מתחבר לחשבון גוגל שלו לדוגמא ומסכים לשתף את הנתונים שם (כמו מייל, שם ועוד.)

Authorization Code Grant

השרת מחזיר ל-Redirect URI את ה-Authorization Code עם ה-State.

```
GET /callback?code=a1b2c3d4e5f6g7h8&state=ae13d489bd00e3c24
```

Token Request

האתר צד שלישי שולח בקשת POST לשרת אימות (OAuth), ההבדל הוא שעכשיו הוא מבקש גם את ה- id token.

```
POST /token HTTP/1.1
Host: oauth-authorization-server.com
client_id=12345
client_secret=SECRET
redirect_uri=https://client-app.com/callback
grant_type=authorization_code
code=a1b2c3d4e5f6g7h8
```

Token Response

שרת אימות מחזיר את ה-Access Token ואת ה-ID Token.

```
{
"access_token": "z0y9x8w7v6u5",
"token_type": "Bearer",
"expires_in": 3600,
"id_token": "eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVC..."
}
```

API Call

הלקוח יכול להשתמש ב-Access Token כדי לגשת למשאבים בשרת (כמו אנשי קשר).

User Info via ID Token

או להשתמש ב-ID Token כדי לדעת מי המשתמש, מתי הוא התחבר וכו, בלי צורך בפניות נוספות לשרת משאבים.

חולשות ב-OAuth

אז כן, למרות שהפרוטוקולים OAuth ו-OIDC נועדו לאפשר אימות והרשאות בצורה מאובטחת ונוחה (ממש כמו Sign in with GOOGLE), הפרוטוקולים האלה עדיין חשופים לחולשות. הסיבה העיקרית היא שה-Framework מאוד מסתמך על הדפדפן, שרתי צד שלישי וtoken-ים שנעים על גבי הרשת. הנקודות תורפה של הפרוטוקול נובעות מחוסר אימות נכון, זליגת token-ים או פשוט תצורה לא מספיק נכונה של Redirect URIs ו-Scopes. בקצרה וכרגיל - החולשות נמצאות דווקא במערכות ואתרים שלא דואגים לעבוד עם קונפיגורציה נכונה.

סוגי חולשות:

Redirect URI Manipulation

במהלך תהליך האימות ב-OAuth, התגובה של השרת יכולה להיות רגישה, כי היא מכילה את החלק הקריטי ביותר, שהוא הקוד או ה-token של המשתמש. אם ניתן "ללכוד" את האלמנטים האלה, ניתן כך להשיג את המידע הרגיש ולהתאמת עם פרטים של משתמש אחר. הדרך הכי נפוצה לעשות את זה היא ניצול של הפרמטר Redirect_URI. אם השרת OAuth לא בודק את הפרמטר באופן מספיק קפדני או שהוא מוגדר בצורה כללית מידי (לדוגמא https://test.*), ניתן לנתב את המשתמש לפלטפורמה "זדונית" במקום לאפליקציית צד שלישי הרצויה.

לדוגמא, אם הייתה דרך לשנות את הכתובת Callback לכתובת של שרת חיצוני שאוסף את הקודי אימות\token-ים של כל מי שניגש אליו כפרמטר.

```
GET /authorization?client_id=1234
&redirect_uri=https://sus-app.com/collect
&response_type=code
&scope=openid%20profile
&state=ae13d489bd00e3c24 HTTP/1.1
Host: oauth-authorization-server.com
```

בדרך כלל החולשה נמנעת על ידי שרת OAuth, שבו מוגדרים הדומיינים הספציפיים אליהם ניתן לנתב את ה-token , אבל זה אכן אפשרי במקרה שבו יש באתר צד שלישי פגיעות ל-Open Redirect.

Open Redirect – הוא פגיעות WEB, מצב שבו אתר לגיטימי מאפשר להפנות משתמש לכתובת חיצונית בלי לוודא אם היא בטוחה, מה שיכול לאפשר לתוקף להפנות משתמשים לאתר זדוני, במקום כתובת לגיטימית.

```
GET /authorization?client_id=1234
&redirect_uri=https:
//client-app.com/redirect?next=https://sus-app.com/collect
&response_type=code
&scope=openid%20profile
&state=ae13d489bd00e3c24 HTTP/1.1
```

במקרה הזה המשתמש ינותב קודם ל-client-app.com/redirect, ואז ל-https://sus-app.com/collect. אם האתר צד שלישי פגיע, ה-Authorization Code יישלח לשרת של התוקף.

CSRF due to Missing State Parameter

ב-OAuth, פרמטר state הוא ערך אקראי וייחודי שנשלח מהאתר צד שלישי לשרת אימות בזמן בקשת ההרשאה. המטרה שלו היא לוודא שהבקשה שהתקבלה בשרת של האפליקציה היא אותה בקשה שהמשתמש התחיל. זה מונע מצב שבו ניתן להזריק קוד או token לא חוקי ולהתחבר בשם המשתמש. הבעיה היא שהפרמטר הזה הוא לא בדיוק חובה, לכן אם הוא חסר או לא נבדק כראוי, נוצרת חולשה קריטית שמאפשרת לנצל את החוסר אימות הזה.

לדוגמא כאשר משתמש רוצה להתחבר לאתר צד שלישי שמבצע אימות באמצעות גוגל, הוא שולח בקשת אימות לשרת אימות. השרת מאמת את המשתמש ומחזיר קוד לאתר צד שלישי, שאליו המשתמש מעוניין להתחבר. במצב תקין, הפרמטר state יישלח יחד עם הקוד, אך במידה והוא לא מוחל כראוי, הוא לא יהיה נוכח.

התוקף יכול לשלוח למשתמש קישור פישינגי זדוני, או בצורה כזו או אחרת לגרום למשתמש ללחוץ על הקישור כמו זה: "https://client-app.com/oauth?code=KJSUSIEGHBSHEOKJJSOIPE", במידה והמשתמש ילחץ על הקישור, הקוד יתקבל על ידי משתמש ויעבור לשרת של האפליקציה, השרת יחשוב שמדובר בקוד חוקי מהמשתמש עצמו ויקשר את החשבון של התוקף עם החשבון משתמש.

התוצאה - התוקף מתחבר לחשבון של המשתמש באתר צד שלישי עם הפרטי גוגל משלו.

Implicit Grant Exploitation

כפי שהבנו מקודם, תצורת Implicit Grant היא תצורת OAuth "מופשטת", שמאפשרת קבלת token ישירות אל הדפדפן, בלי החלפת קודים בין שרתים.

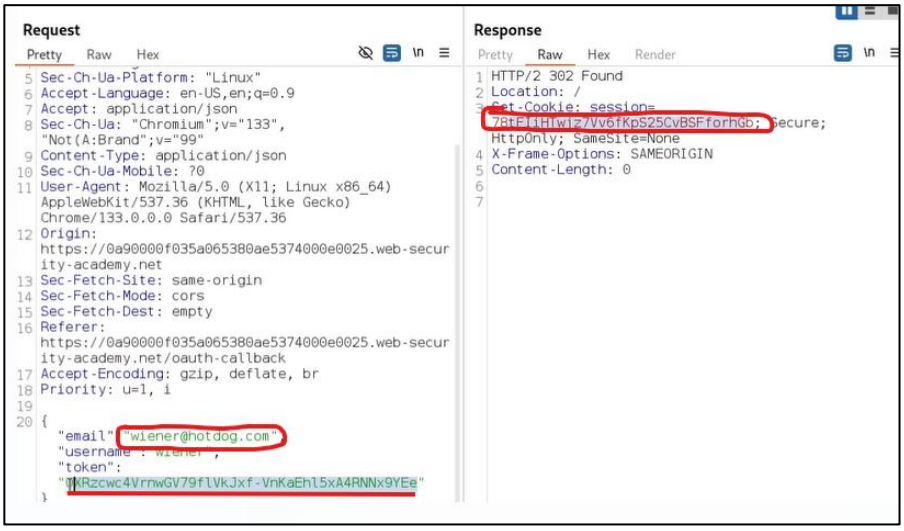
אם האתר צד שלישי לא מוודא בצורה נכונה שהמידע מגיע עם ה-Access Token שבאמת שייך למשתמש שמנסה להתחבר, מתקבלת אפשרות להזריק מידע לא נכון.

לדוגמא משתמש מנסה להתחבר עם גוגל לאתר צד שלישי, הדפדפן ישירות מקבלת Access Token:

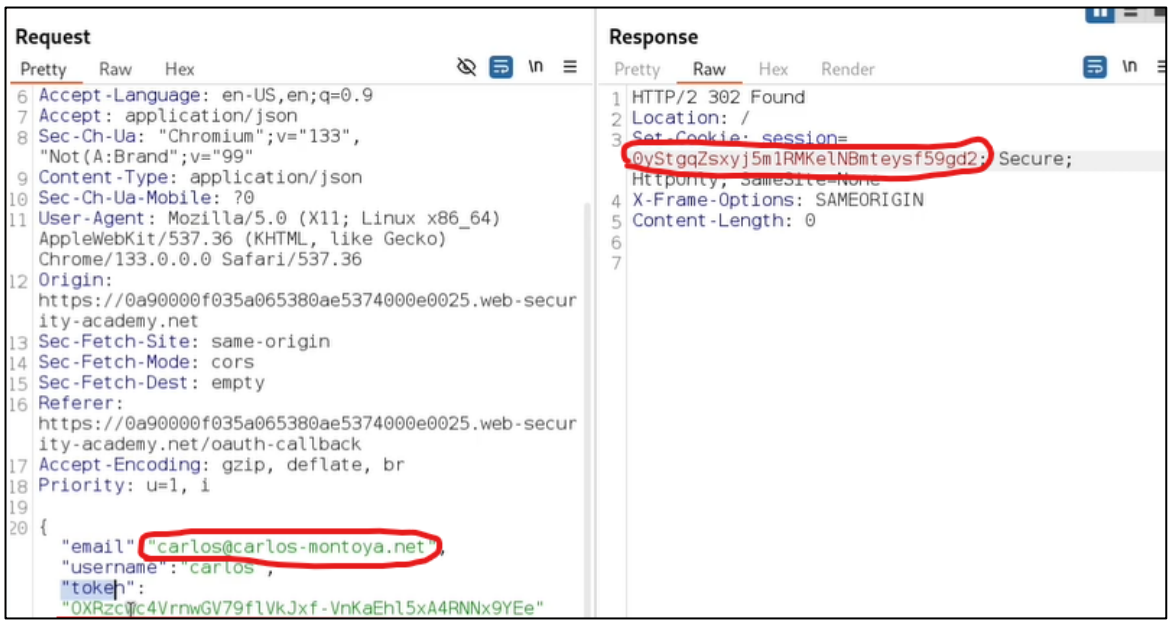
```
https://client-app.com/callback#access_token=1234&token_type=bearer&expires_in=3600
```

בגלל שכאן, השולח של ה-token הוא הדפדפן, אין שום תקשורת מאובטחת כמו תקשורת משרת לשרת וה-token חשוף, יש אפשרות להשיג token-ים של משתמשים אחרים. ניתו להעתיק token של משתמש אחר, או ליצור token מזוייף, לדוגמא באמצעות XSS או מתקפה אחרת, ה-token יישלח לאתר צד שלישי והוא יתקבל בלי אימות מסויים מול השרת אימות, מה שקורה מידי פעם באתרים שלא מוודאים את האמינות של token-ים לאחר קבלתם, ה-token המזוייף יוכל לאפשר גישה לחשבונות אחרים, כלומר – גניבת זהות.

במידה וה-token לא נבדק, ניתן לערוך פרטי משתמשים בתוך הבקשה וכך להשיג מידע, כמו coockies, וכך להשתלט על הסשן הקיים.



כאן ניתן לראות בקשת אותנטיקציה עם פרטי משתמש שיש לנו את ה-token שלנו (wiener, חובבי portswigger יזהו ישר), כתגובה קיבלנו את ה-cookie של המשתמש, וכך השגנו session בשמו.



אם נשנה רק את המייל והשם למשתמש קורבן שלנו, אם קיים session שלו במערכת, נוכל להשיג את ה-cookie שלו גם כן, בלי שהאתר צד שלישי יוודא את ה-token (שאפילו לא שינינו!) וכך נקבל גישה למשתמש שלו, בלי להשתמש לא בסיסמא ולא ב-token שלו.

Token Manipulation

במידה ונשלח JWT (לרוב ID Token) מזוייף לאתר צד שלישי והוא לא מוודא את התקינות/חתימה של ה-token מול המפתח הציבורי, האתר צד שלישי עלול לסמוך על כך שהמשתמש מזוהה ולהעניק לו גישה לנתונים או פעולות שהיו אמורות להיות מוגבלות למשתמש לגיטימי בלבד.

```
POST /oauth/callback HTTP/1.1
Host: client-app.com
Content-Type: application/x-www-form-urlencoded

id_token=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwiZW
1haWwiOiJhdHRhY2tlckBleGFtcGxlLmNvbSIsIm5hbWUiOiJBdHRhY2tlciIsImlhdCI6MTY5Nz
YyMjQwMCwiZXhwIjoxNzAwMjM4NDAwfQ.fakeSignature
```

אם האתר צד שלישי לא בודק את החתימה מול המפתח הציבורי של ה-OAuth Provider, הוא עלול להאמין שהמשתמש מזוהה ולהעניק לו הרשאות.

בנוסף, ניתן לשנות את הפרטים בתוך ה-Payload של ה-JWT וכך להשיג גישה למשתמש קיים.

Scope Abuse

אתר צד שלישי יכול לבקש הרשאות גבוהות מידי, או רחבות מידי (גישה ליותר מידי משאבים, ללא צורך), מעבר למה שהוא צריך. זה אפשרי מכיוון שה-Scopes מוגדרים בתור Strings והמשתמש לא תמיד יודע או שם לב או מבין אילו הרשאות הוא נותן.

Token Replay

ה-Access Token שנגנב יכול לשמש שוב ושוב לגישה לנתונים, גם אם פג תוקף, מכיוון שבאתר צד שלישי אין מנגנון של ניהול token-ים נכון.

Insecure Storage of Tokens

יש מצב שבו האחסון של הtoken-ים מצד האתר לקוח לא מיושם בצורה נכונה, לדוגמא אם מאוחסנים ב-Local Storage של השרת, או בקבצים לא מוצפנים. הדבר הזה מקל על גניבת token-ים.

OAuth2.1

כחלק מניסיון למנוע את כל החולשות האבטחתיות שנובעות משימוש ב-OAuth2.0, שזו הגרסה שדיברנו עליה לאורך כל המאמר, נמצא פתרון והוא שדרוג ה-Framework לגרסא משופרת, OAuth2.1, שמטרתה לקחת את כל הכשלים של הגרסא המקורית ולהפוך אותם כדרישות מחייבות, כדי לצמצם את המשטח תקיפה.

מהעדכונים שנוספו:

1. ביטול של תצורת Implicit Grant
2. חובת שימוש ב-PKCE, או בשמו המלא Proof Key for Code Exchange, מנגנון אבטחה ב-Oauth, שמוסיף סיסמה חד פעמית להחלפת הקוד בין השרתים, כדי לוודא שהקוד שנשלח לא יורט בשום שלב.
3. הגדרת Redirect_URI ספציפיים מראש.
4. חובת שימוש ב-nonce וב-state.
5. הקשחת ניהול ואחסון token-ים

מקורות מידע

- https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/05-Authorization_Testing/05-Testing_for_OAuth_Weaknesses
- https://astrix.security/learn/blog/part-2-how-attackers-exploit-oauth-a-deep-dive/
- https://portswigger.net/web-security/oauth#what-is-oauth
- https://www.vaadata.com/blog/understanding-oauth-2-0-and-its-common-vulnerabilities/
- https://oauth.net/2.1/