

Access Tokens in Azure ועוד דברים נחמדים

מאת ספיר פדרובסקי

הקדמה

במשך שנים אותנטיקציה היתה עבורי Kerberos, מנגנון חכם ומתוחכם שמרשים אותי בכל פעם מחדש. אבל העולם מתקדם וכדאי להתקדם ביחד איתו. כיום, שימוש בפלטפורמה עננית כלשהי זה MUST.

ואיך נלמד את עולם הענן בלי הדובדבן שבקצפת? אותנטיקציה!

משום שאני ומיקרוסופט זה לנצח, במאמר זה אתמקד ב-Azure ובשמש של מיקרוסופט למנגנון OAuth2.0. נדבר על Access tokens, Refresh tokens ועל מספר מתקפות וגילויים סביב התחום הזה.

Azure ב-OAuth 2.0

Azure Active Directory היא הפלטפורמה בה משתמש Azure כדי לספק גישה לשירותים השונים שהוא מספק. כשאני אומרת "לספק גישה" אני מתכוונת לשני דברים:

1. בדיקת הרשאות Authorization

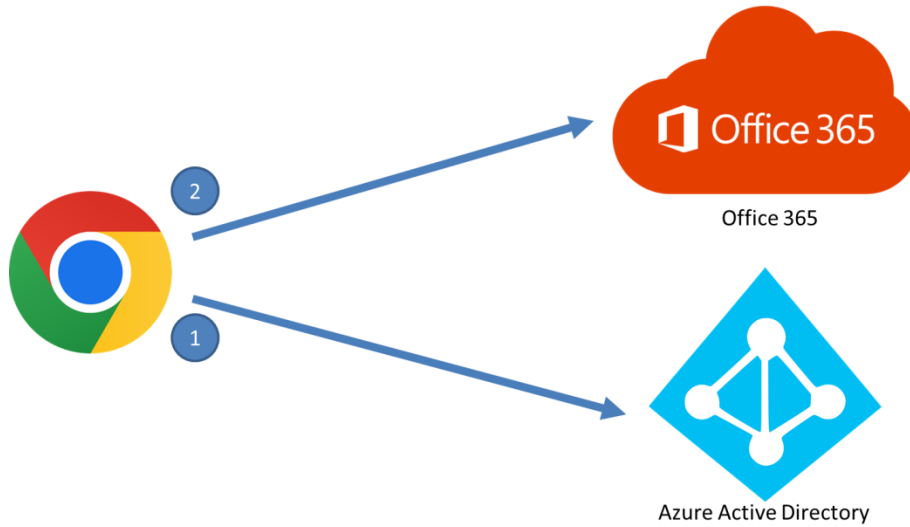
2. אימות המשתמש Authentication

Azure AD מבצע את שני תפקידיו באמצעות התקנים OAuth2.0 ו-OpenID connect בהתאמה. אך כמובן שהמימוש של מיקרוסופט ל-OAuth2.0 הוא טיפה שונה מההגדרה המקורית שלו ב-RFC, ולשינויים האלו יש מספר השלכות.

הנפשות הפועלות

נניח שאני רוצה להתחבר ל-Office365 שהוא אחד השירותים ש-Azure מספק. אז יש לי דפדפן, אני נכנסת ל-Office 365.

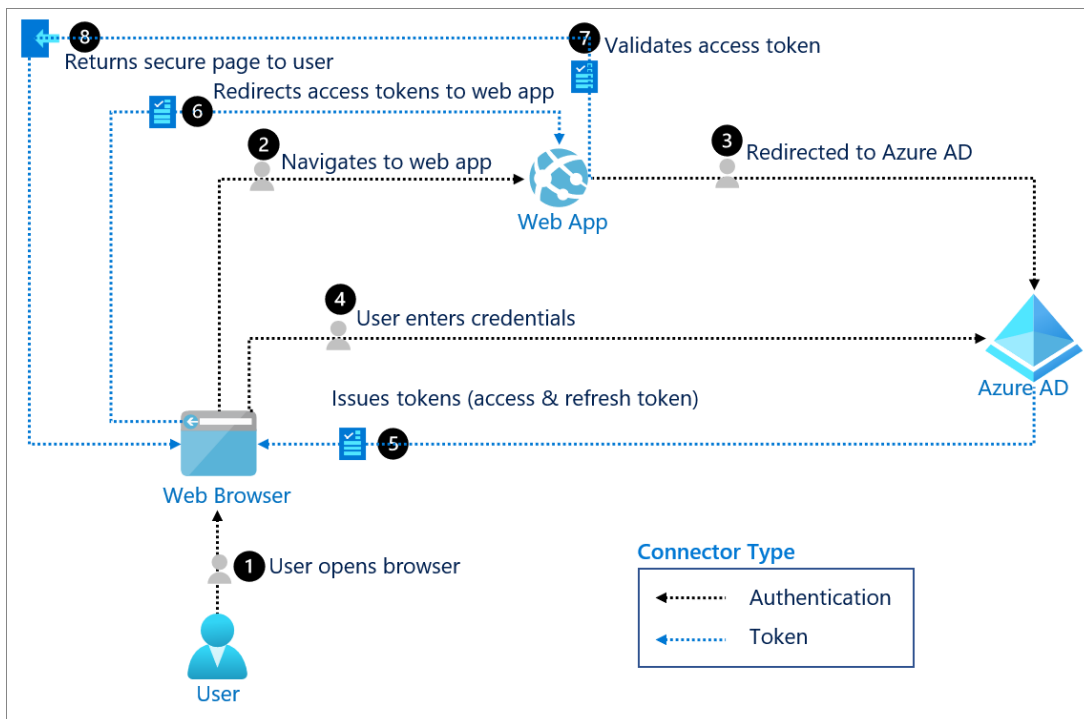
בואו נבחן את התרחיש הזה רגע:



נכנסתי לאתר ולאחר מכן אני מכניסה שם משתמש וסיסמא. אז ב-Kerberos יש לנו Tickets, מה יש לנו כאן? ומה בעצם קורה שם? אלו הרכיבים משתתפים בתהליך:

1. המשתמש
2. הדפדפן
3. האפליקציה (למשל, Office 365)
4. Azure AD, ה-Identity provider

ה-Flow שלנו הוא כזה (נלקח מהמאמר של מיקרוסופט על המימוש שלה ל-OAuth2.0, קישור בסוף המאמר):



פירוט השלבים

המשתמש פותח את הדפדפן ונכנס לעמוד האפליקציה, בתרחיש שלנו: נכנסנו ל-Office 365. האפליקציה אוטומטית מפנה אותנו ל-Azure AD שהוא בעצם ה-Identity Provider שלנו ויספק לנו שני דברים:

1. Authentication

2. Authorization

עד פה עשינו את זה:



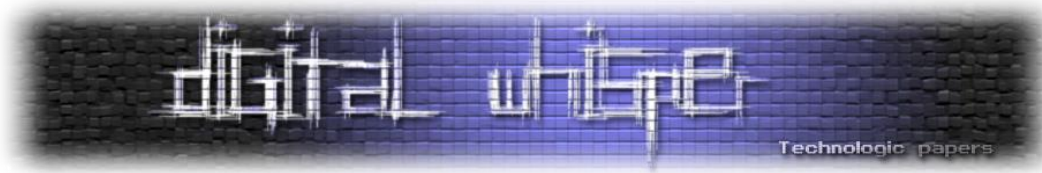
לאחר שהכנסנו את שם המשתמש והסיסמא (או השתמשנו באחת מדרכי ההזדהות האחרות שנדבר עליהן בהמשך), Azure AD ינפיק עבורינו Token, למען האמת, שני Token-ים.

לאחר מכן הדפדפן יפנה אותנו בחזרה לעמוד האפליקציה, שתבצע ולידציה ל-Token שלנו ובמידה והוא תקין תחזיר לנו את העמוד המבוקש. פשוט, לא? אז אפשר לצלול קצת יותר לעומק.

הנה תמונה של התחברות ל-Azure portal:

General	
Request URL:	https://login.microsoftonline.com/5c93733a-13df-4f58-abc7-fd9226f855c3/oauth2/v2.0/token
Request Method:	POST
Status Code:	● 200 OK
Remote Address:	[2603:1027:1:28::11]:443
Referrer Policy:	strict-origin-when-cross-origin

ה-GUID שניתן לראות הוא בעצם ה-Tenant ID שלי. זהו בעצם המזהה החדד ערכי של הסביבת Azure שלי.



מסקנות מהתשובה

1. סוג ה-Token שלנו נקרא Bearer token. וזה מאוד חשוב כי זה סוג ספציפי של Token. המשמעות שלו אומרת, שכל מי ש"נושא" אותו, לא צריך להוכיח את זהותו. זאת אומרת, שידיעת ה-Token מספיקה כדי להשתמש בו.
- בשונה למשל, מ-Kerberos, שם, גם אם יש לך את הכרטיס, ללא ה-Session Key (שלא היית יכול להשיג ללא הסוד של הלקוח), לא תוכל להשתמש בכרטיס.
2. Scope - באמצעות השדה הזה נוכל ללמוד באילו משאבים נוכל להשתמש ובאילו הרשאות בדרך כלל מדובר במשאבים שמאפשרים API שונים כמו ניהול הסביבה, זה מאוד מעניין ונגע בכך בהמשך.
3. Access token - תקף לשעה אחת בלבד. בו נשתמש כדי לגשת למשאבים, בכל בקשת גישה למשאב חייב להעביר Access token
4. Refresh token - תקף ל-90 יום. נשתמש בו כדי לבקש Access tokens
5. Client info - מכיל מידע על המשתמש, כמו User principal name, Location, ולא נדבר עליו במאמר זה 😊

אז מה זה Token, מה הוא מכיל, איך הוא נראה, ולמה יש יותר מאחד?

Azure משתמש ב-Token מסוג JWT - Json Web Token, שהוא בעצם סטרינג במבנה Json מקודד ב-Base64. השימוש ב-Token קורה ב-Header של הבקשה, למשל:

```
GET /Something HTTP/1.1
HOST: random.host
Authorization: Bearer jvfdalkjvskdlasdckji
```

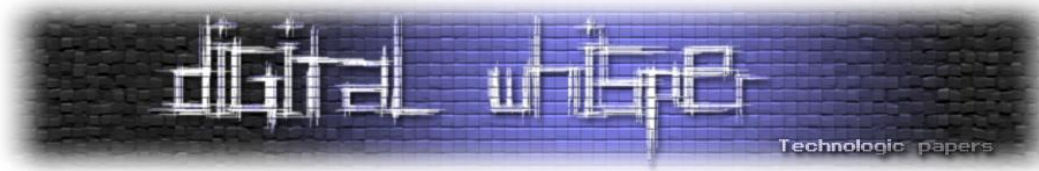
חשוב להדגיש ש-JWT יכול להיות כמה דברים:

1. JWE - Json Web Encryption
2. JWS - Json Web Signature

JWS

משמש ל-Access token והוא בנוי מ-3 חלקים:

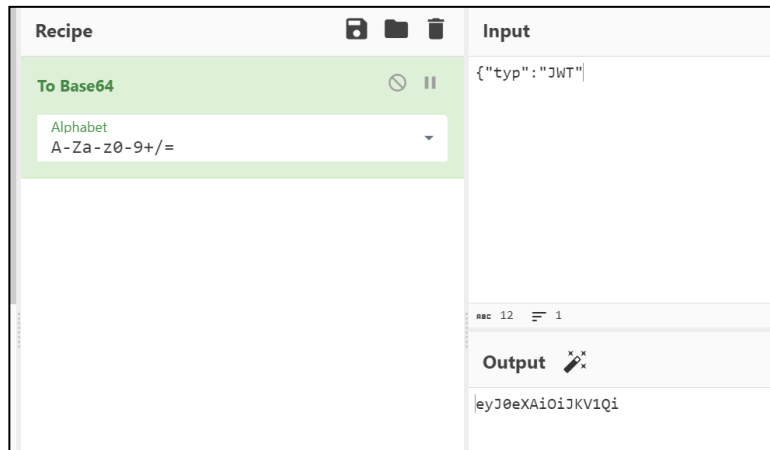
- JOSE - JavaScript Object and Encryption Header
- Payload (בדרך כלל הסטרינג שמזהה את הלקוח בפורמט JSON)
- Signature (כביכול אופציונאלי אבל למזלנו ב-Azure זה תמיד ממומש)



וכל היופי הזה יראה בעצם ככה:

Base64(UTF8(JOSE Header)) Base64(Payload) Base64(Signature)

משום שהמבנה הוא קבוע, ה-Access token תמיד יתחיל ב- {"typ": "JWT"} ומשום שעושים עליו Base64 בלי שום Salt, ה-Access token תמיד יתחיל עם אותן אותיות:



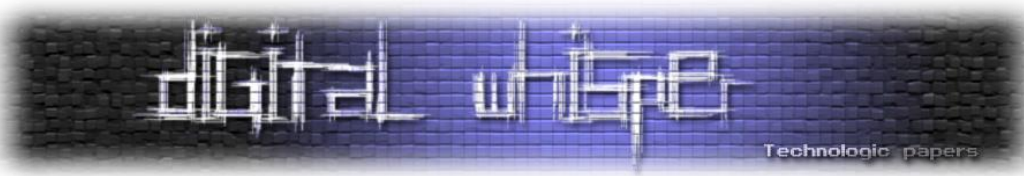
אולי זה לא נראה משמעותי עכשיו, אבל היכולת לזהות סטרינגים כ-Access Tokens היא ממש שימושית, ונראה את זה בהמשך.

JOSE Header בדרך כלל יראה ככה:

```
{
  "typ": "JWT",
  "alg": "RS256",
  "x5t": "-KI3Q9nNR7bRofxmeZoXqbHZGew",
  "kid": "-KI3Q9nNR7bRofxmeZoXqbHZGew"
}
```

השדות הם:

1. סוג ה-Token
2. אלגוריתם ההצפנה (יכול להיות גם HMAC-SHA256 להצפנה סימטרית)
3. X509 Certificate SHA1 Thumbprint
4. Key id



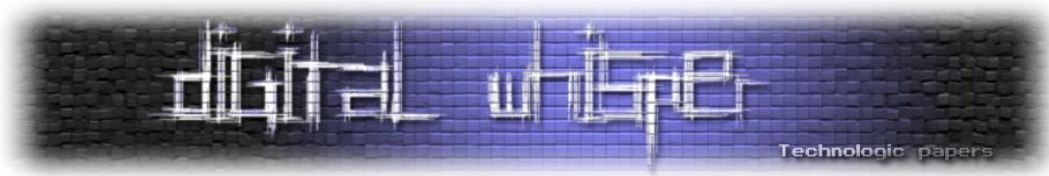
ה-Payload נראה בערך ככה:

```
{
  "aud": "https://management.core.windows.net/",
  "iss": "https://sts.windows.net/5c93733a-13df-4f58-abc7-fd9226f855c3/",
  "iat": 1691947861,
  "nbf": 1691947861,
  "exp": 1691953131,
  "acr": "1",
  "aio": "AVQAq/8UAAAAU2ysYMIaN3R1On4QRTd9QViyDo/f38wzIc4mdl138JZSDBIDxQsvYqohoZcYTIieWAx05Xi/0aGxk9fLFkt03gRp531pTyuIcCjwjWfdgfgQ=",
  "altsecid": "1:live.com:000340015ED6B2CD",
  "amr": [
    "pwd"
  ],
  "appid": "c44b4083-3bb0-49c1-b47d-974e53cbdf3c",
  "appidacr": "0",
  "email": "sapxfed@outlook.com",
  "family_name": "federovsky",
  "given_name": "sapir",
  "groups": [
    "d885beb6-d26d-4845-9449-d851b3195245"
  ],
  "idp": "live.com",
  "ipaddr": "2a10:8012:9:4321:96e:f232:4cc8:2382",
  "name": "sapir federovsky",
  "oid": "0970f620-2d52-4998-9fd0-e34fc36b88be",
  "puid": "10032002D55F81FA",
  "rh": "0.Aa4A0nOTXN8TWE-rx_2SjvhVw0ZIf3kAutdPukPawfj2MBOuAD8.",
  "scp": "user_impersonation",
  "sub": "vVn52NPPke7jpaCAqF-XgHXfNdq0iNX56w4W3Nk8j24",
  "tid": "5c93733a-13df-4f58-abc7-fd9226f855c3",
  "unique_name": "live.com#sapxfed@outlook.com",
  "uti": "8_RONa2CmUiPaVw4gFAUAA",
  "ver": "1.0",
  "wids": [
    "62e90394-69f5-4237-9190-012177145e10"
  ],
  "xms_tcdt": 1691942150
}
```

אפרט רק חלק מהשדות:

- aud - המשאב, ה-API שנוכל להשתמש בו
 - iss - מנפיק ה-Token החלק האחרון יהיה ה-Tenant ID שלכם
 - nbf/exp - תאריך התחלה וסיום של תוקף ה-Token
 - scp - ה-Scope של ה-Token, הרשאות
- הנה דוגמא יותר מעניינת לשדה הזה מזו שבתמונה:

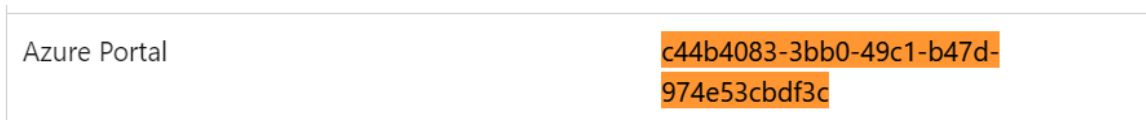
```
"scp": "AccessReview.ReadWrite.All AuditLog.Read.All
ConsentRequest.Create ConsentRequest.Read
ConsentRequest.ReadApprove.All
ConsentRequest.ReadWrite.All
CustomSecAttributeAuditLogs.Read.All
Directory.AccessAsUser.All Directory.Read.All
Directory.ReadWrite.All Directory.Write.Restricted
DirectoryRecommendations.Read.All
DirectoryRecommendations.ReadWrite.All email
EntitlementManagement.Read.All Group.ReadWrite.All
IdentityProvider.ReadWrite.All
IdentityRiskEvent.ReadWrite.All
IdentityUserFlow.Read.All openid Policy.Read.All
Policy.Read.IdentityProtection
Policy.ReadWrite.AuthenticationFlows
Policy.ReadWrite.AuthenticationMethod
Policy.ReadWrite.ConditionalAccess
Policy.ReadWrite.ExternalIdentities
Policy.ReadWrite.IdentityProtection
Policy.ReadWrite.MobilityManagement profile
Reports.Read.All RoleManagement.ReadWrite.Directory
SecurityEvents.ReadWrite.All
TrustFrameworkKeySet.Read.All User.Export.All
User.ReadWrite.All
UserAuthenticationMethod.ReadWrite.All",
```



- amr - באיזו שיטה התאמתתי כדי לקבל את ה-Token
- Tenant id - tid
- appid - האפליקציה הרלוונטית, נוכל לראות את ה GUID של כל האפליקציות כאן:

<https://learn.microsoft.com/en-us/troubleshoot/azure/active-directory/verify-first-party-apps-sign-in>

במקרה שלנו, נוכל לראות שהאפליקציה היא Azure portal:



תהליך הולידציה על החתימה

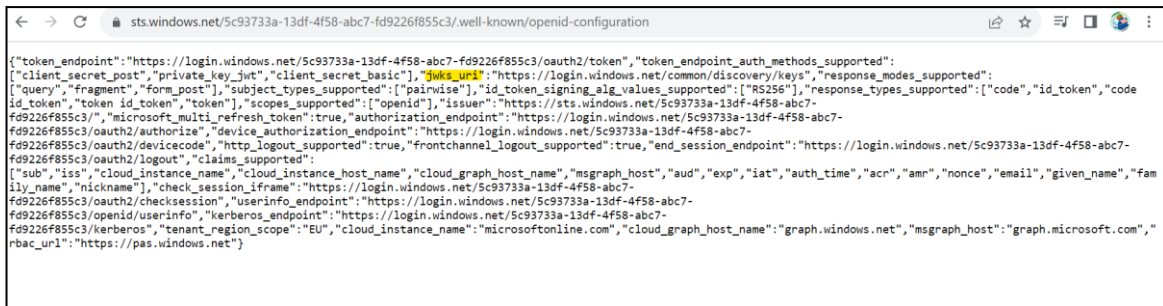
בסוף אנחנו כאן בשביל אותנטיקציה, אז איך מתבצע תהליך אימות החתימה? נכנסים ל-Openid Configuration שימצא ב-URL הבא:

`<iss>/well-known/openid-configuration`

במקרה שלנו זה יהיה הנתביב:

<https://sts.windows.net/5c93733a-13df-4f58-abc7-fd9226f855c3/well-known/openid-configuration>

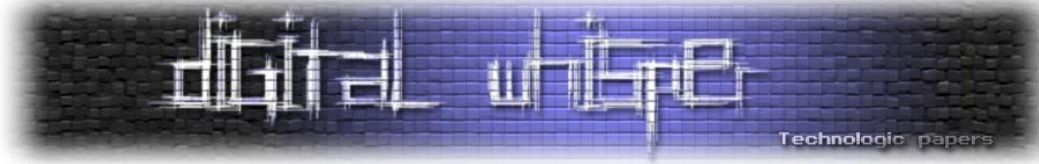
שם נמצא את ההגדרה הבאה - `jwtks_uri`:



שמפנה אותנו ל:

<https://login.windows.net/common/discovery/keys>

עכשיו, נצטרך למצוא את המפתח המתאים, נעשה זו באמצעות ה-`id` Key שנמצא ב-`JOSE Header`.



במקרה שלנו:

```
login.windows.net/common/discovery/keys
{"keys": [{"kty": "RSA", "use": "sig", "kid": "220p3UupbjAYKYGaXE3181V0TOI", "x5t": "220p3UupbjAYKYGaXE3181V0TOI", "n": "wE...nR7bRofmeZoXqbHZGew", "e": "AQAB", "x5c": "IF9k3KjpTyOy25-9_DrDvPGjV1QK0iCjBf9yQAL-pBec9r-XkAS-C4zTn1ZRw--GELabuo9U-UsrTKj4zxFDEP-R5Rp0GshcC9S1kU1Steuuh4F8M3XFR2GB8c..."}, {"kty": "RSA", "use": "sig", "kid": "220p3UupbjAYKYGaXE3181V0TOI", "x5t": "220p3UupbjAYKYGaXE3181V0TOI", "n": "wE...nR7bRofmeZoXqbHZGew", "e": "AQAB", "x5c": "IF9k3KjpTyOy25-9_DrDvPGjV1QK0iCjBf9yQAL-pBec9r-XkAS-C4zTn1ZRw--GELabuo9U-UsrTKj4zxFDEP-R5Rp0GshcC9S1kU1Steuuh4F8M3XFR2GB8c..."}]
```

עכשיו נוכל לקחת את ה-Public key או התעודה ולבצע את הולידציה על החתימה שלנו ☺. אם אתם רוצים לראות את הJWT שלכם, אתם יכולים לפרסר אותו בקלות באתר הזה:

<https://jwt.io/>

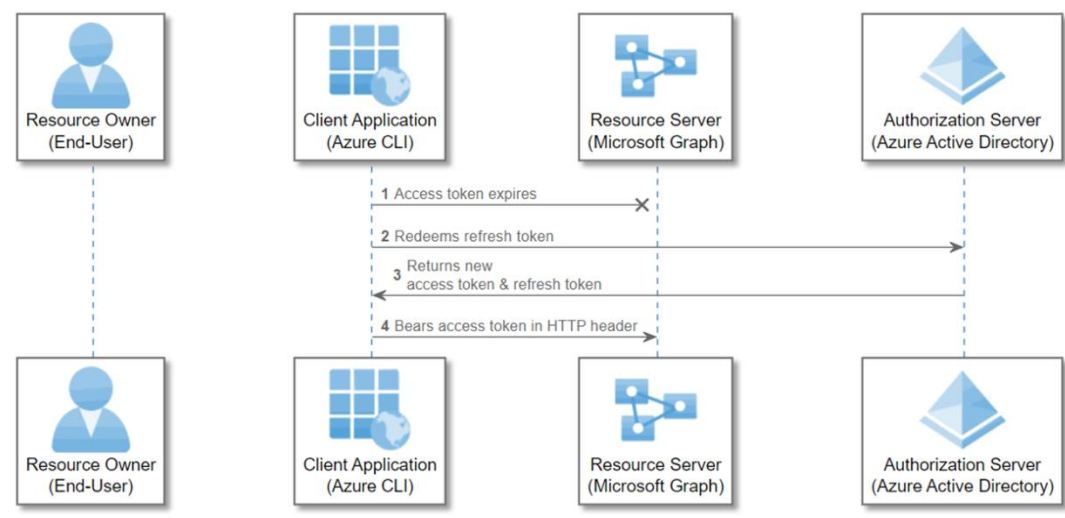
או לחילופין באמצעות הפקודה:

```
read-aadintaccesstoken
(שהיא חלק מהמודול ה-Powershell:-adinternals)
```

Refresh tokens

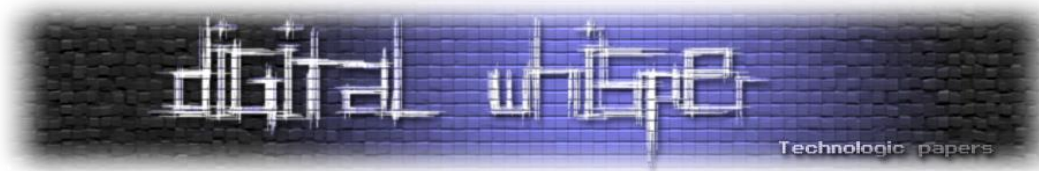
אז מה הם בעצם Refresh Tokens?

ה-Token הזה תקף ל-90 יום (לעומת ה-Access token שתקף בלבד) ואנחנו משתמשים בו (אוטומטית) כל פעם כדי לבקש Access token חדש. הוא מוצפן ומפוענח אך ורק על ידי מפתח ציבורי ופרטי של מיקרוסופט שלא ידוע לאף אחד פרט למיקרוסופט. לכן לא ניתן לפענח אותו כפי שעשינו עם Access token. התהליך לקבלת Token חדש באמצעות Refresh token נראה בערך ככה:



Access Tokens in Azure

www.DigitalWhisper.co.il



כפי שניתן לראות, התהליך הזה ממש שקוף עבור המשתמש, עבורו, הוא פשוט באורך פלא לא צריך להכניס שם משתמש וסימא מחדש כל שעה.

מהמידע הזה מובן כי אם תוקף מצליח לשים את ידו על Refresh token הוא בעצם יכול להנפיק לעצמו Access tokens חדשים ל-90 יום. וכאשר אתה מנפיק Access token אתה מנפיק גם Refresh token ככה שבעצם כל עוד התוקף דואג לעשות זאת לפחות פעם אחת ב-90 יום, תהיה לו יכולת להנפיק Token-ים לנצח!

לכן, OAuth2.0 מצויד בכמה כללים שמטרתם למתן את רמת הנזק:

1. Same scope - כלל זה אומר שניתן להנפיק Access token חדש רק אם הוא תואם את ה-Scope של ה-Token המקורי או שהוא מכיל גרסה מצומצמת של הרשאות ב-Token המקורי. (זאת אומרת קטן שווה).

זאת ועוד, גם Refresh token חדש חייב להיות תואם ב-Scope וב-Resource. הישר מה-RFC:

If refresh tokens are issued, those refresh tokens MUST be bound to the scope and resource servers as consented by the resource owner. This is to prevent privilege escalation by the legitimate client and reduce the impact of refresh token leakage.

זאת אומרת, אם קיבלתי Token שמאפשר לי הרשאות Application.Read.All, אני לא אוכל להנפיק באמצעותו Token חדש עם הרשאות Application.ReadWrite.All.

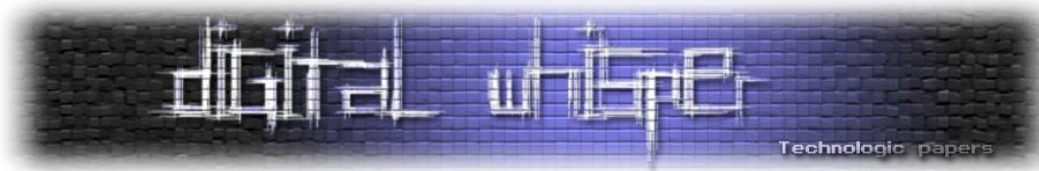
2. Same client - ה-Refresh token צריך להיות קשור ל-Client שהוא הונפק עבורו. ועל ה-Authorization Server או במקרה שלנו Azure AD חלה החובה לבדוק את הקשר הזה בכל פעם שמונפק Refresh token חדש. הישר מה-RFC:

The authorization server should match every refresh token to the identifier of the client to whom it was issued. The authorization server should check that the same "client_id" is present for every request to refresh the access token. If possible (e.g., confidential clients), the authorization server should authenticate the respective client. This is a countermeasure against refresh token theft or leakage.

סך הכל נשמע חכם, אבל במיקרוסופט כמו במיקרוסופט החליטו לכופף טיפה את החוקים. אז המימוש של OAuth2.0 במיקרוסופט נראה יותר ככה: MRRT - Multi Resource Refresh Tokens.

MRRT

MRRT מתעלם מהכלל הראשון (להגביל את הגישה בהתאם ל-Scope המקורי) ויותר מתנהג כמו TGT בפרוטוקול Kerberos, זאת אומרת: נוכל להשתמש ב-MRRT כדי לאמת את עצמנו ולבקש כל Access token שנרצה. (כמובן שנקבל אותו רק במידה ואנו זכאים אליו). אבל נקודת המפתח כאן היא שאותו MRRT יכול לשמש אותי לבקש כל Access token שארצה ולכל Resource.



לכן, אם תוקף שם את ידו על ה-Refresh token שלי, לא משנה אם רק השתמשתי בו בשביל להנפיק Access token בהרשאות User.read.all, במידה ואני זכאית למשל להרשאות UserAuthenticationMethod.ReadWrite, הוא יוכל להנפיק Access token עם ה-Scope הזה.

את החוק השני מיקרוסופט דווקא כן מקיימים, זאת אומרת שה-Token כן צריך להיות לאותו User ולאותו Tenant application. אבל! MRRT לא בודקים את ה-Tenant. זאת אומרת, נוכל לבקש MRRT לכל Tenant אליו יש לנו הרשאות.

לפי הדוקומנטציה של מיקרוסופט:

"Refresh tokens are bound to a combination of user and client, but aren't tied to a resource or tenant... a client can use a refresh token to acquire access tokens across any combination of resource and tenant where it has permission to do so."

- חשוב לציין כי בעבר MRRT היו Legacy feature אבל כיום זו היא ההתנהגות הדיפולטית של כל ה-Refresh tokens ב-Azure AD.

FOCI

אז נכון שאמרנו שלפחות מיקרוסופט שמרו על הכלל השני? אז זה לא בדיוק נכון. התגלה כי אפשר לקבל באמצעות Token שהונפק ל-First-party Microsoft client applications, Token ל-First-party Microsoft client applications אחר.

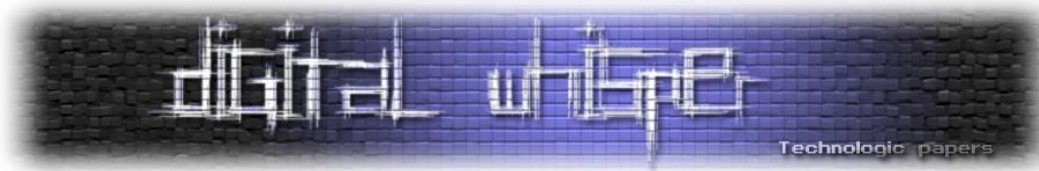
- כשאני אומרת First party אני מתכוונת לאפליקציות מובנות של מיקרוסופט, כמו למשל, Microsoft teams. זאת אומרת שמדובר במקרה שמפר את כלל מספר 2, זו היא ההתנהגות לא צפויה שלא תועדה בדוקומנטציה של מיקרוסופט.

במחקר מצאו סך הכל 15 אפליקציות שניתן להפיק מה-Token שלהן Token חדש לאפליקציות אחרות(מתוך ה-15). אז מה הן האפליקציות האלו? ולמה רק איתן ההתנהגות הזו מתקיימת?

האפליקציות האלה מוגדרות כ-FOCI, Family Of Client Ids. מונח זה מוזכר בדוקומנטציה של מיקרוסופט סך הכל פעמיים. ופעם אחת זה כדי להגדיר את ראשי התיבות האלו. האיזכור השני מדבר על הזדהות לאפליקציות Office ממכשירי מובייל.

לאחר מחקר קצת יותר מעמיק, FOCI הוא בגדול:

"FUTURE SERVER WORK WILL ALLOW CLIENT IDS TO BE GROUPED ON THE SERVER SIDE IN A WAY WHERE A RT FOR ONE CLIENT ID CAN BE REDEEMED FOR A AT AND RT FOR A DIFFERENT CLIENT ID AS LONG AS THEY'RE IN THE SAME GROUP. THIS WILL MOVE US CLOSER TO BEING ABLE TO PROVIDE SSO-LIKE FUNCTIONALITY BETWEEN APPS WITHOUT REQUIRING THE BROKER (OR WORKPLACE JOIN)."



זאת אומרת, קיבוץ של Client Ids בצד השרת כך שבאמצעות Refresh token שהונפק עבור אפליקציה אחת, יהיה ניתן להנפיק Access tokens לאפליקציות האחרות שבאותה הקבוצה. המטרה היא כמובן נוחות, לספק תמיכה ב-SSO דרך מובייל לאפליקציות מיקרוסופט שונות.

הקונספט הוא בעצם ליצור Cache משותף לכל קבוצה שכזו, בה יהיה ה-Token והוא יהיה תקף לרכישת Access tokens לכלל האפליקציות בקבוצה.

כיום, ידועה בציבור רק קבוצה FOCI אחת, אך הסבירות היא שיש עוד והן לא התגלו עדיין. אוסיף את רשימת ה-Client Ids של ה-FOCI בסוף המאמר.

ההשלכות של התנהגות זו כמובן ברורות, שכן היא מפרה את כלל מספר 2. התנהגות זו מקלה על התוקף משמעותית, שכן במידה והשיג MRRT של אחד מהאפליקציות ב-FOCI, הוא יכול כעת להנפיק Access tokens לכלל האפליקציות ב-FOCI.

כפי שהובטח, דברים נחמדים

התאמתות באמצעות Device Code Flow

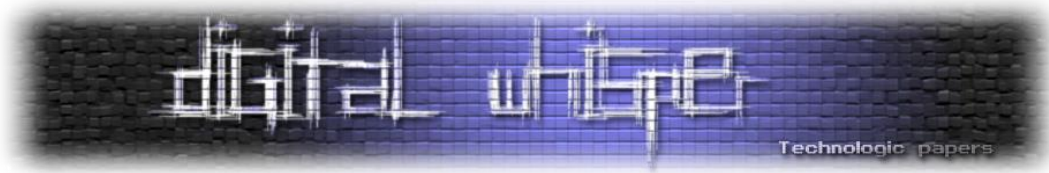
הקונספט של התאמתות בצורה כזו היא למכשירים שלא מאפשרים התאמתות באמצעות שם משתמש וסיסמא, בדרך כלל מדובר על תרחיש שבו אין לי דפדפן. התאמתות זו מאפשרת באופן דיפולטי ותוקפים אוהבים להשתמש בה בתרחישי Phishing.

התהליך הוא כזה, בהתאמתות מסוג Device code flow המשתמש יקבל קוד, הוא יצטרך להשתמש במכשיר אחר שיש בו דפדפן, להכנס ל-URI הבא:

<https://microsoft.com/devicelogin>

ולהכניס את הקוד. משם יעבור המשתמש תהליך התאמתות רגיל ובסופו המכשיר ללא הדפדפן יוכל לבצע קריאות API למשאב המבוקש. תרחיש התקיפה הוא כדלקמן: תוקף יושב בבית עם PowerShell, הוא מתכנן להשתמש בקריאת ה-API הבאה:

```
HTTP Copy
POST https://graph.microsoft.com/v1.0/users
Content-type: application/json
{
  "accountEnabled": true,
  "displayName": "Adele Vance",
  "mailNickname": "AdeleV",
  "userPrincipalName": "AdeleV@contoso.onmicrosoft.com",
  "passwordProfile" : {
    "forceChangePasswordNextSignIn": true,
    "password": "xWwvJ]6NMw+blW-d"
  }
}
```



באמצעות בקשת POST זו יוכל התוקף להוסיף משתמש חדש ל-Tenant עליו יש לו הרשאות. כמובן שב-Header הבקשה הזו נצטרך את ה-Access token. אז איך התוקף יכול להשיג Access token?

אחרי שהרצנו את הפקודה הבאה: (מתוך aadinternals):

```
PS C:\WINDOWS\system32> $at = Get-AADIntAccessTokenForAADGraph -UseDeviceCode -Tenant 5c93733a-13df-4f58-abc7-fd9226f855c3
To sign in, use a web browser to open the page https://microsoft.com/devicelogin and enter the code DERRUGZL3 to authenticate.
```

קיבלנו URI וקוד. כעת, נוכל לשלוח הודעת פשינג שעושה בקשת POST ל-URI הזו עם הקוד, וכל מה שהקורבן צריך לעשות הוא רק ללחוץ ולאשר. השיטה הזו נורא נפוצה בימים אלו.

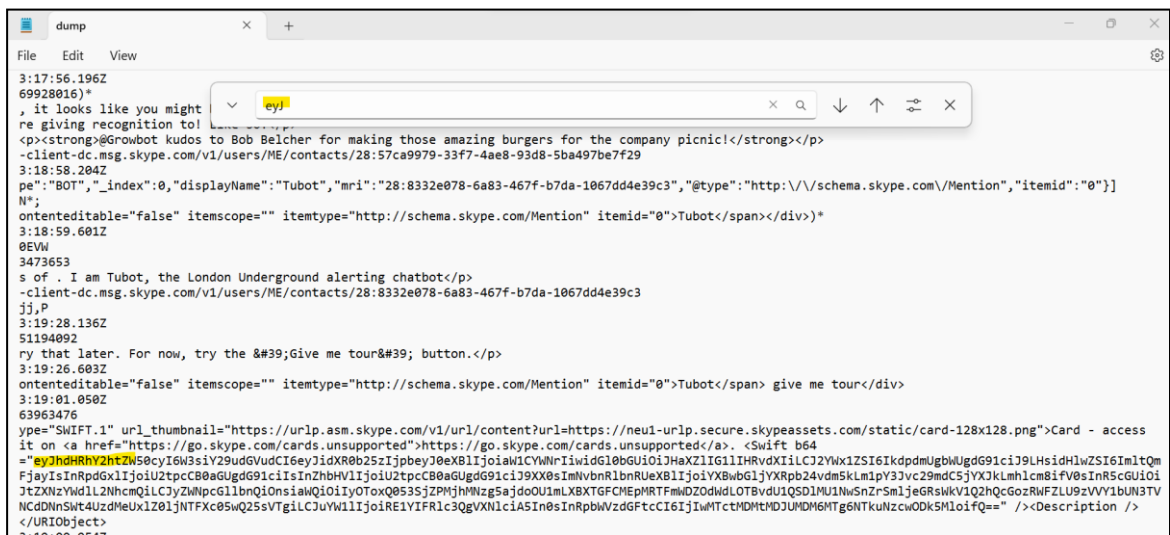
לאחר האישור, התוקף בבית מקבל את ההרשאות שהיה צריך וכעת הוא יכול להשתמש ב-Access token שקיבל ולבצע באמצעותו קריאות API!

On-Prem cached tokens

עוד טכניקה מאוד נחמדה, לא מזמן התגלה כי מיקרוסופט לא ממש טובים באחסון של Access tokens כשהם מגיע לאפליקציות On-Prem למינהן, והם פשוט מאוחסנים ב-Clear text!

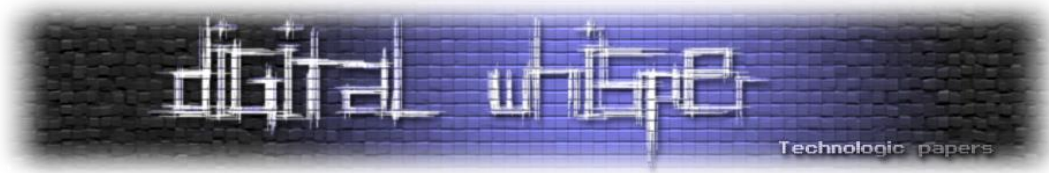
זוכרים שאמרתי שזה שימושי שכל Access token תמיד מתחיל באותה צורה? זה הרגע להוכיח את התועלת שבדבר.

כל מה שצריך זה Dump קטן לתהליך של Teams וה-Token שלנו:



- עובד כיום על מגוון רחב של אפליקציות Office

Storm-0558



אם כבר מדברים, המתקפה הכי מעניינת במרחב ה-Access tokens התרחשה ממש לא מזמן.

- כל המידע על מתקפה זו מגיע מהחקירה של מיקרוסופט בנושא:

<https://www.microsoft.com/en-us/security/blog/2023/07/14/analysis-of-storm-0558-techniques-for-unauthorized-email-access/>

הקרדיט מגיע לקבוצה סינית בשם Storm-0558 שידועה בתחום הריגול. קבוצה זו הצליחה לא פחות מאשר לזייף Access tokens ולהתחבר למיילים של משתמשים ספציפים ברחבי העולם.

האירוע הזה התרחש ממש לא מזמן במאי 2023.

במקרה שלנו שירות היעד היה OWA - Outlook Web Application. לטענת מיקרוסופט, הקבוצה פרצה ל-25 ארגונים בלבד, ביניהם ארגונים ממשלתיים, וכל שעשה הוא שליפת מידע שהיה במיילים.

כפי שתיארתי קודם, ולידציה על ה-Token נעשית באמצעות המפתח הציבורי. במידה ותוקף שם את ידו על המפתח הפרטי, הוא יכול לחתום כל Token שירצה ובכך להפוך אותו לתקף.

שיטה זו נקראת Token forgery. לפי הדוח של מיקרוסופט, הקבוצה הצליחה לשים את ידה על מפתח פרטי שכבר לא פעיל (Inactive). כרגע אין תשובה לשאלה כיצד הקבוצה הצליחה להשיג את המפתח. זו היתה הבעיה הראשונה.

הבעיה השנייה היתה, שהמפתח היה אמור לאפשר חתימה על Token-ים של סוג ספציפי של חשבונות בשם MSA accounts שהם בעצם חשבונות אישיים (לא כמו חשבון של מקום עבודה).

אך עקב בעיית ולידציה בקוד של מיקרוסופט, המפתח היווה חתימה תקפה ל-Token-ים של Azure AD. לאחר שהשיגו את ה-Token הראשוני, הם השתמשו ב-API של OWA שנקרא:

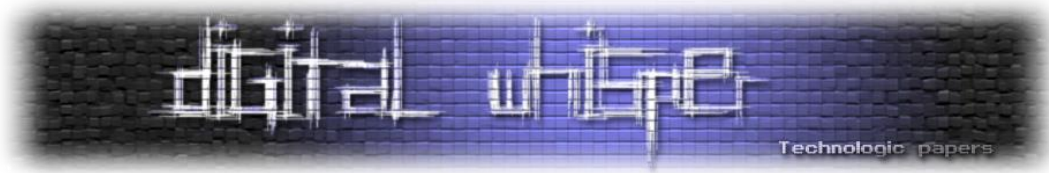
GetAccessTokenForResource

בו היה באג, ובאמצעותו הנפיקו Token ל-Exchange online. החקירה בנושא עוד טרייה ובתהליכים, היא מאוד מעניינת ואני ממליצה לעקוב!

בעמוד של מיקרוסופט תמצאו הסברים נוספים על המתקפה, כיצד מיקרוסופט זיהתה את הקבוצה, דרכי מניעה ומזהים. כמו כן, תוכלו לקרוא את המאמר המעולה של WIZ על ההשלכות של זיוף ה-Token-ים בשיטה בה השתמשו התוקפים.

המאמר של WIZ:

<https://www.wiz.io/blog/storm-0558-compromised-microsoft-key-enables-authentication-of-countless-micr>



סיכום

השימוש בענן הולך וגובר עם הזמן, ולא נראה שהוא יפסק מתישהו. לכן בתור אנשי סקיריטי חשוב להבין את המשמעות של ענן בארגון, מה ההשלכות והסכנות הכרוכות בדבר, כיצד מתבצעת האותנטיקציה בענן, וכיצד פועלים תוקפים היום כדי להשיג גישה לענן.

המאמר הזה מהווה רק תמצית מהרעיונות, המתקפות והקונספטים הקשורים לאותנטיקציה בענן, ואין ספק שיש להעמיק יותר ויותר ולהיות מוכנים לדבר הגדול הבא.

בהצלחה לנו ☺

על הכותבת

@sapirxfed - עושה retweet לכל דבר שקשור ל-AD או AAD. אוהבת לכתוב קוד, מנסה למצוא חולשות, ומעריצה שרופה של DigitalWhisper!

ביבליוגרפיה

הקרדיט כל כולו מגיע לחברה מקסימה בשם SecureWorks וללא אחר מאשר Nestori Syynimaa או בשמו היותר מוכר: @drAzureAD

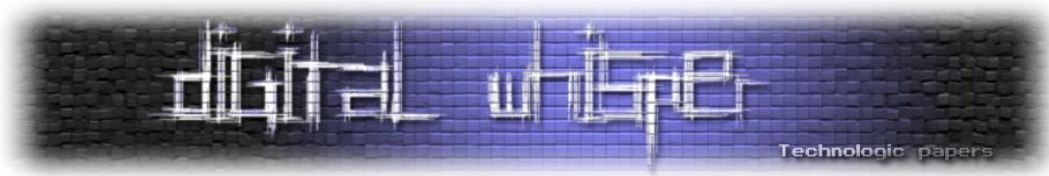
<https://aadinternals.com>

דנת Token-ים שהועברה במספר כנסים בעולם:

<https://www.youtube.com/watch?v=8qEh1pc0tT8&t=7702s>

מאמר על FOCI:

<https://github.com/secureworks/family-of-client-ids-research>



מתוך המאמר, טבלת ה-FOCI הידועה כיום:

Application ID	Application Name
00b41c95-dab0-4487-9791-b9d2c32c80f2	Office 365 Management
04b07795-8ddb-461a-bbee-02f9e1bf7b46	Microsoft Azure CLI
1950a258-227b-4e31-a9cf-717495945fc2	Microsoft Azure PowerShell
1fec8e78-bce4-4aaf-ab1b-5451cc387264	Microsoft Teams
26a7ee05-5602-4d76-a7ba-eae8b7b67941	Windows Search
27922004-5251-4030-b22d-91ecd9a37ea4	Outlook Mobile
4813382a-8fa7-425e-ab75-3b753aab3abb	Microsoft Authenticator App
ab9b8c07-8f02-4f72-87fa-80105867a763	OneDrive SyncEngine
d3590ed6-52b3-4102-aeff-aad2292ab01c	Microsoft Office
872cd9fa-d31f-45e0-9eab-6e460a02d1f1	Visual Studio
af124e86-4e96-495a-b70a-90f90ab96707	OneDrive iOS App
2d7f3606-b07d-41d1-b9d2-0d0c9296a6e8	Microsoft Bing Search for Microsoft Edge
844cca35-0656-46ce-b636-13f48b0eecbd	Microsoft Stream Mobile Native
87749df4-7ccf-48f8-aa87-704bad0e0e16	Microsoft Teams - Device Admin Agent
cf36b471-5b44-428c-9ce7-313bf84528de	Microsoft Bing Search

דוקומנטציה של מיקרוסופט על OAuth2.0:

<https://learn.microsoft.com/en-us/azure/active-directory/architecture/auth-oauth2>

RFC-ים:

<https://datatracker.ietf.org/doc/html/rfc6819#section-5.2.2.2>

<https://datatracker.ietf.org/doc/html/draft-ietf-oauth-security-topics#section-4.14.2-2>

המאמר על חילוף ה-Token מתוך אפליקציות On-Prem של Office:

<https://mrd0x.com/stealing-tokens-from-office-applications/>