

# Digital Whisper

גליון 154, ספטמבר 2023

מערכת המגזין:

מייסדים:

אפיק קסטיאל, ניר אדר

מוביל הפרויקט:

אפיק קסטיאל

עורכים:

אפיק קסטיאל

כתבים:

יונתן בר אור, ספיר פדרובסקי, עידן שכטר

יש לראות בכל האמור במגזין Digital Whisper מידע כללי בלבד. כל פעולה שנעשית על פי המידע והפרטים האמורים במגזין Digital Whisper הינה על אחריות הקורא בלבד. בשום מקרה בעלי Digital Whisper ו/או הכותבים השונים אינם אחראים בשום צורה ואופן לתוצאות השימוש במידע המובא במגזין. עשיית שימוש במידע המובא במגזין הינה על אחריותו של הקורא בלבד.

פניות, תגובות, כתבות וכל הערה אחרת - נא לשלוח אל [editor@digitalwhisper.co.il](mailto:editor@digitalwhisper.co.il)

---

## דבר העורך

---

ברוכים הבאים לגליון ה-154 של DigitalWhisper!

אנחנו חיים בעידן עם קצב מטורף. בעידן שבו האנושות מצליחה להמציא, לייצר ולרתום כל כך הרבה טכנולוגיות שונות ומורכבות על מנת להציב ולהשיג את מטרותיה פעם אחר פעם. עדכונים על פריצות דרך מדעיות או הישגים משמעותיים כבירים טכנולוגיים שהאנושות מצליחה לייצר, פוקדים אותנו מספר פעמים בחודש. הינה לקט רק מהחודשים האחרונים שקפצו לי לראש בזמן כתיבת הפסקה: ההתקדמות המרשימה של ה-NIF לקראת [היתוך גרעיני מבוקר](#) - מה שאולי יאפשר בעתיד את הפסקת המרוץ הבלתי פוסק אחר מקורות אנרגיה. מחקרים פורצי דרך בממשקי אדם-מכונה, שאפשרו קריאת אותות ישירות ממוחו של [מטופל שהיה משותק בכל גפיו כדי לאפשר לו לקום ולהתחיל לצעוד](#), והעדכון על כך ש-[Neuralink קיבלה אישור מה-FDA לעשות ניסויים בשתלי מוח לבני אדם](#), אחרי שהציגה תוצאות מרשימות בקריאת מחשבותיהם של קופים. כל הטירוף שיש סביב עולם הבינה המלאכותית, מודלים כמו ChatGPT, Midjourney ושות' מקפיצים פלאים כל תחום שבו משלבים אותם. וכמובן - [הנחיתה של החללית ההודית צ'אנדריאן 3 על הירח](#), שהתרחשה רק לפני ימים בודדים. ויש, כמובן, עוד מלא...

כל פריצות הדרך הללו מתבססות על טכנולוגיות שמלוות אותן ומאפשרות אותן, וקצב הגדילה והפיתוח שלהן הוא כנראה אקספוננציאלי, כי כל פיתוח טכנולוגי מהווה כלי פיתוח לעוד שלל פיתוחים טכנולוגיים, וכל פריצת דרך מדעית מאפשרת פריצות דרך טכנולוגיות נוספות.

קצב הגדילה הטכנולוגי כל כך מטורף, שכבר כמעט ולא ניתן לשלוט ב-Stack טכנולוגי שלם. סט כל הטכנולוגיות המרכיבות פתרון מסויים כל כך גדול שהאדם העוסק בנושא חייב להחליט שמאיזשהי שכבה אי-אפשר להתעמק יותר, יש לקבל את הטכנולוגיה הקיימת כמו שהיא ומכאן מתחילים. אם פעם כחלק משיעורי ביולוגיה במוסד אקדמי כלשהו, על התלמידים היה נדרש לדעת להרכיב בעצמם את המיקרוסקופ, ובסוף התואר כל סטודנט ידע איך לפרק אותו לגורמיו או לתקן כל תקלה בו, היום המצב הוא אינו כזה, והמיקרוסקופ הוא כמעט בגדר "קופסא שחורה". יש בגישה הנ"ל כמובן לא מעט חסרונות, אך יש בה גם יתרונות - מי שלומד ביולוגיה יכול בשנות לימודיו להתעסק אך ורק במחקר הביולוגי עצמו וללמוד יותר על התחום. תשאירו למהנדסים לפרק, לתקן ולהרכיב את הציוד הרפואי.

באותה הגישה, כך פועל גם עולם הפיתוח ואבטחת המידע. מפתח Web שמעוניין לפתח שירות Web-י מאובטח כלשהו, ומחליט להשתמש בשרת nginx או Apache, לא יכול לעבור על כל הקוד שלו, גם במקרים בהם הקוד של כל ה-Stack הטכנולוגי מבוסס קוד פתוח. לבקש ממפתח להכיר לעומק ולקמפל בעצמו כל טכנולוגיה שבה הוא משתמש זאת פשוט דרישה לא סבירה ולא אפקטיבית. במקרה הזה, עליו פשוט לקרוא



את ה-Manual, או לממש את ה-Interface המתאים ולהניח שהצד השני, שאחראי על המימוש, עובד ללא תקלות.

קצב הגדילה, מונע מהיכולת של הפרט הבודד לשלוט בכל הטכנולוגיה שבה הוא עושה שימוש ומחייבת אותנו להפריט את העבודה למספר צוותים, צוותי Front-End, צוותי Back-End, צוותי System, תשתיות ו-DevOps. אך עדיין, הגידול המעריכי הוא כל כך מופרע, כך שגם לצוות עם מספיק כוח אדם אין שום סיכוי להכיר לעומק את רזי כלל הטכנולוגיות שבהן הוא עושה שימוש. לצפות ממנו לבצע שימוש נכון וחכם באותן הטכנולוגיות? כן, לצפות שהוא יצליח לפתור תקלות מורכבות ב-Stack שעליו הוא אחראי? בהחלט. אך להכיר לעומק בדיוק איך כל ספרייה וספרייה הטעונה לכל תהליך שמבצע עיבוד למידע עליו הצוות אחראי - פשוט אין סיכוי. הארגון חייב לגדול כדי לשרוד. וכך, בדרך כלל גם ה-Stack הטכנולוגי שבו הוא עושה שימוש.

אם פעם, עוד כשה-Stack הטכנולוגי הסטנדרטי היה "פשוט" (ובנוי ממערכת הפעלה, שרת Apache בודד שהיה מריץ PHP שמגיש תוכן דינאמי), לא היה ניתן לצפות שמפתח בודד יכיר הכל, היום, Stack סטנדרטי יהיה בנוי מעשרות או מאות מיקרו-Services, שכל אחד מריץ טכנולוגיה אחרת, והאורכסטרייה שלהם מתבססת גם היא על טכנולוגיה נוספת, והלוגיקה שאחראית על מתי כל דבר יתבצע יכולה להיגזר מחישובים שמבוצעים בכלל בארגון שונה משלנו, אין שום סיכוי שניתן לעקוב אחרי כל ה-Flow ובלי הטכנולוגיות שעוברות ומנתחות את המידע שלנו.

רמת המורכבות של סביבת ה-Production לא עלתה סתם. היא נובעת מכך שהבעיה שאיתה ארגונים צריכים להתמודד נהייתה מורכבת יותר. יש יותר משתמשים, שמייצרים יותר בקשות שצריך לטפל בהן, המניפולציות שצריך לבצע על המידע נהיו מורכבות יותר, והפיצ'רים שצריך לספק דורשים טכנולוגיות רבות יותר. הכל צריך להיות מוגש בקצב מהיר יותר ובהרבה מאוד מקרים, עלות הטעות היא גבוהה יותר. החיים לא נעשים פשוטים יותר מבחינה טכנולוגית אף פעם.

פעם, היה מספיק לעשות Pen Testing אחת לרבעון, לעשות סקרי סיכונים שנתיים כדי להבין במה צריך להשקיע, או להזמין צוות Red-Team חיצוני שיבצע "רענון" או הדמייה לצוות ה-IT בארגון. בשנים האחרונות אפשר לראות מגמה כך שיותר ויותר ארגונים הקימו צוותי Red-Team פנימיים שמבצעים תרגולות באופן כמעט קבוע, ואימוץ של גישות ומוצרים עם כותרות כמו "Continuous Security Validation", כאלה שמוודאות תמיד שהטכנולוגיות בארגון מעודכנות ובקונפיגורציית Best Practice בכל רגע נהיות נפוצות יותר ויותר.

זאת כמובן מגמה חיובית, אך עושה רושם שגם היא כבר איננה מספיקה. [החודש פורסם כי שני חברים בקבוצת ההאקרים LAPSUS\\$ הורשעו](#) בבית דין בלונדון. נראה שהילדים היו בני 16 ו-17 בזמן שביצעו את פשעיהם (שבין היתר כוללים, פריצה לחברות ענק כגון NVIDIA, RockStar Games, Otka וחברות טלקום גדולות כגון Orange, Everything Everywhere, ו-British Telecom).



מדובר בגיל שערויייתי לכל דעה. אין ספק שהחבר'ה האלה מוכשרים מאוד, כמות הנזק והרעש שהם הצליחו לייצר היא כמעט לאין-שיעור, וזה עוד לפני שהם סיימו את העשור השני לחייהם. על השיטות השונות בהן חברי LAPSUS\$ נקטו כדי לחדור לארגונים השונים כבר כתבנו בגליונות עבר, וקל מאוד להסכים עם כך שהן אינן מורכבות מאוד מהפן הטכנולוגי. נראה שהרבה מהחבר'ה השתמשו ברוב המקרים, בעובדים ממורמרים, שמתוך ייאוש או כעס על המעסיק - העניקו הרשאת גישה (או ממש הביאו את פרטי הגישה ואת הקוד של ה-MFA) כדי להתחבר ב-VPN לרשת הארגון או לסביבה הנתקפת. בשום מתקפה לא נצפה שימוש ב-Oday או בשיטת תקיפה מורכבת שאינה נמצאת ברשימות של MITRE.

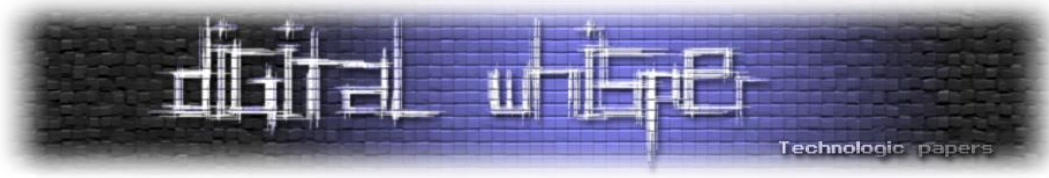
הנ"ל מתאפשר, בעיקר מכך שכאשר יש לתוקף סיוע מגורם פנימי - כמעט ולא ניתן לעצור אותו מלהכנס. אולם, המצב שבו הוא יצליח להמשיך לגשת לרכיבים ברשת מבלי להתפס, נובע ממורכבות הטכנולוגיה, ומכך שכמעט ולא ניתן לנטר כל חוליה קטנה בשרשרת. ה-Stack פשוט מורכב מדי מכדי שיהיה ניתן לנטר אותו בצורה כזאת שלא יהיו False Positives. ובדיוק שם התוקף יכול להיכנס, ואת ה-Misconfigurations האלה ניתן לנצל כדי להתקדם עוד אל עבר היעד.

המורכבות הטכנולוגית נובעת משלל סיבות, המרכזית שבהן היא שהבעיה שארגונים מנסים לפתור והשירות שאותו הם מספקים ללקוח נהיו מורכבים לאין שיעור. כל חברה מציעה פתרון ניטור שונה לחלק מסויים ברשת, רוצים לוגים מסביבת ה-Kubernetes? תשתמשו בפתרון X, רוצים ניטור על כתיבת קבצים לתיקיות מסוימות בשרת? תשתמשו בפתרון Y. הפתרונות של נרמול הלוגים והזרמתם לפתרון דמוי אלסטיק גם לא מספקים ולא הרמטיים דיים, וכל עוד לא תתחיל מגמה של פישוט ה-Stack הטכנולוגי של הארגון (מה שכיום לא נראה כל כך אפשרי מגודל מגוון המערכות) - ארגוני פשיעה כמו LAPSUS\$ יוכלו להמשיך לשגשג ולפעול.

אז שיהיה בהצלחה לכולנו!

וכמובן, לפני שנעבור למאמרים הנפלאים שמתפרסמים בגליון החודש, נרצה להגיד תודה לכל מי שכתב ועמל על כך. אז תודה לכל הכותבים! תודה רבה ליונתן בר אור תודה רבה לספיר פדרובסקי ותודה רבה לעידן שכטר!

**קריאה נעימה,  
אפיק קסטיאל**



---

## תוכן עניינים

---

2	דבר העורך
5	תוכן עניינים
6	איך לחטוף מיגרנה (CVE-2023-32369)
14	Access Tokens in Azure ועוד דברים נחמדים
30	מבוא למחקר אפליקציות
45	דברי סיכום

# איך לחטוף מיגרנה (CVE-2023-32369)

מאת יונתן בר אור

## הקדמה

מאמר זה מסכם מחקר שפורסם על ידי קבוצת Microsoft Defender ופורסם במאי 2023.

המחקר מתאר כיצד חיפוש שגרותי של קוד זדוני על גבי טלמטריות EDR, גרם לגילוי design issue ב-macOS שעוקף את מנגנון SIP. לאחר השמשת החולשה הצוות דיווח על הממצאים לחברת אפל, שסגרה את החולשה (כעת ידועה גם כ-CVE-2023-32369).

מטרת מאמר זה היא הנגשת המחקר בעברית ותיאור של מספר פרטים נוספים שהושטו מפרסום המחקר לצורך נוחות קריאה.

## קצת מידע על SIP

בדומה ל-SE Linux על מערכות לינוקס, גם ב-macOS משתמש root אינו באמת כל יכול - מנגנון בשם SIP (קיצור שם System Integrity Protection, ידוע גם כ-rootless) מונע הדבקה של מערכת ההפעלה עצמה, כולל:

- מניעת כתיבה של קבצים לאזורים מוגנים במערכת ההפעלה (בדרך כלל תחת System / אבל לא רק).
- מונע טעינת דרייברים (macOS kernel extensions) שלא אושרו מראש על ידי אפל.
- מונע דיבוג של תהליכים שחתומים על ידי אפל.
- מונע kernel debugging, כולל DTrace.
- מונע שינוי של משתני NVRAM.

כפי שניתן להתרשם, SIP הוא מנגנון הגנה חזק מאד. מערכות macOS מגיעות עם SIP דלוק כברירת מחדל, והדרך הלגיטימית היחידה לכבות אותו היא לעשות reboot למצב מיוחד הנקרא Recovery OS שממנו ניתן לכבות את SIP - לא פעולה שניתנת לאוטומציה וגם לא משהו שמשתמש קצה "רגיל" צפוי לעשות.

מנגנון SIP עצמו מתוחזק על ידי הקרנל ומושפע ממשתני NVRAM (לכן ההגנה על משתני NVRAM הכרחית, אגב) וספציפית ממשתנה בשם csr-active-config. שווה לציין שעקיפה של כל מנגנון אכיפה שהוזכר עלול להפיל את כל SIP (כתיבת משתני NVRAM באופן טריוויאלי, אבל גם דברים אחרים כגון טעינת דרייברים, דיבוג של תהליכים חתומים או אפילו כתיבה על קבצים מוגנים).



מבין כל האכיפות הללו, האכיפה המורגשת ביותר על ידי משתמשי קצה הוא מניעת כתיבה של קבצים לאזורים מוגנים, והיסטורית - חלק גדול מעקיפות SIP התמקדו באזור זה.

נקודה חשובה להמשך המאמר היא שחברת אפל נתקלת בבעייה - בעת עדכון, מערכת ההפעלה צריכה לדרוס קבצים שנמצאים תחת אזורים מוגנים ולכן צריכה בעצמה לעקוף את SIP! לשם כך, הוגדרו תהליכים שחתומים על ידי אפל ועבורם מוגדרות יכולות השמורות רק להם - אותן יכולות ידועות כ-[Entitlements](#). ישנם Entitlements רבים (למעשה, ישנו [Database](#) בו הם מתוחזקים) אך אנחנו מעוניינים בשניים מאד ספציפיים:

1. `com.apple.rootless.install` - בינארי המחזיק ב-`entitlement` זה יכול לעקוף את אכיפת SIP על מערכת הקבצים.

2. `com.apple.rootless.install.heritable` - בינארי המחזק ב-`entitlement` זה מוריש לבניו את `com.apple.rootless.install`.

באופן טבעי, בינארים אלו הם ה-"קורבנות הטבעיים" של חולשת מעקף SIP, והיום נתמקד באחד מהם.

## גילוי החולשה

צוות המחקר של Microsoft Defender ביצע חיפוש שגרתי של קוד זדוני על מערכות macOS וגילה הרצה של בינארי בשם `drop_sip`. אותו בינארי נראה נפוץ מאד (המון הרצות על תחנות קצה רבות) - ושמו מרמז על מעקף של SIP. הצוות בחן את אותו בינארי ונראה שהוא מאפשר מחדש אכיפת SIP על מערכת הקבצים ולאחר מכן פשוט מריץ שורת פקודה שהוא מקבל בארגומנטים):

```
1 int64 __fastcall start(__int64 argc, char **argv, char *const *envp)
2 {
3     const char *err_msg_fmt; // r14
4     const char *prog_name; // rax
5
6     if ( csops(0LL, 12LL, 0LL, 0LL) )
7     {
8         err_msg_fmt = "%s: Failed to clear flag.\n";
9     }
10    else
11    {
12        execve(argv[1], argv + 1, envp);
13        err_msg_fmt = "%s: Failed to exec.\n";
14    }
15    prog_name = getprogname();
16    fprintf(&_mh_execute_header.magic, err_msg_fmt, prog_name);
17    return 1LL;
18 }
```

הקריאה `csops` היא syscall פרטי של אפל, והמספר 12 שמור כקבוע `CS_OPS_CLEARINSTALLER` - אותו קבוע "מנקה" את `com.apple.rootless.install` מהתהליך.



הגילוי כי drop\_sip הוא בינארי סטנדרטי ולא מזיק היה גילוי טוב (הצוות חושב שאפל צריכים לשנות את השם ל-"drop\_drop\_sip") אך מעלה סוגייה מעניינת - מדוע ש-drop\_sip יניח מראש שהוא יכול לעקוף אכיפת SIP על מערכת הקבצים?

התשובה היא ש-drop\_sip הוא תהליך בן של תהליך בשם systemmigrationd, ואותו תהליך אכן מחזיק ב-  
:com.apple.rootless.heritable

```
root@McJbo Desktop # codesign -dvv --entitlements - /System/Library/PrivateFrameworks/systemmigrationd | grep rootless
Executable=/System/Library/PrivateFrameworks/SystemMigration.framework/Versions/A/Resources/systemmigrationd
Identifier=com.apple.systemmigrationd
Format=Mach-O universal (x86_64 arm64e)
CodeDirectory v=20400 size=755 flags=0x0(none) hashes=13+7 location=embedded
Platform identifier=14
Signature size=4442
Authority=Software Signing
Authority=Apple Code Signing Certification Authority
Authority=Apple Root CA
Signed Time=Dec 4, 2022 at 8:00:06 PM
Info.plist=not bound
TeamIdentifier=not set
Sealed Resources=none
Internal requirements count=1 size=76
  [Key] com.apple.rootless.volume.Preboot
  [Key] com.apple.rootless.install.heritable
  [Key] com.apple.rootless.datavault.controller
root@McJbo Desktop #
```

התהליך systemmigrationd הוא daemon שאחראי ל-Migration ("ניוד" של מידע בין macOS ל-macOS ואפילו מ-Windows). מכיוון ש-systemmigrationd מחזיק ב-entitlementment חזק כל כך תהינו אילו עוד תהליכים רצים תחתיו, ומהר מאד גילינו שניים מעניינים: bash ו-perl. הסיבה לכך ששניהם מעניינים הם כי הם interpreters, ולכן קל יחסית לגרום להם להריץ קוד כרצוננו (הזרקות אינו עניין פשוט ב-macOS, גם כאשר רצים כ-root, והסיבה לכך נעוצה גם באכיפות שונות על ידי SIP, בין היתר). ובכן, מהר מאד גילינו שניתן להשפיע על perl ועל bash בקלות, עם משתני סביבה:

- perl יחפש משתנה סביבה בשם PERL5OPT ומכיל command-line-switches שיכולים להריץ פקודות perl נוספות.
- bash יחפש משתנה סביבה בשם BASH\_ENV שיכול להחזיק פקודות bash להריץ בעת עלייה במצב לא אינטרקטיבי.

ב-macOS הדרך לקבוע משתני סביבה תחת launchd (תהליך האב של הכל ב-macOS, כולל של daemons כגון systemmigrationd) היא עם launchctl. הנה דוגמה לקביעת משתנה סביבה שיריץ את הקובץ /private/tmp/migraine.sh בכל פעם ש-perl מתחיל לרוץ:

```
launchctl setenv PERL5OPT '-Mwarnings;system("/private/tmp/migraine.sh")
```

ובכן, הרצת Migration מתאימה אכן גורמת לקוד שלנו לרוץ! בדוגמה שלנו דרסנו את הקובץ /Library/Apple/System/Library/Extensions/AppleKextExclusionList.kext/Contents/Resources/Exc-eltionLists.plist - זהו קובץ שמוגן על ידי SIP ומכיל מידע על Kernel extensions שניתנים לטעינה.





דריסה שכזו יכולה לאפשר טעינת Kernel extensions משלנו, אבל אנחנו רק כתבנו בקובץ Migraine לצורך ההדגמה:

```
mipearse@Mikes-MBP ~ % csrutil status
System Integrity Protection status: enabled.
mipearse@Mikes-MBP ~ % cat /Library/Apple/System/Library/Extensions/AppleKextExcludeList.kext/Contents/Resources/ExceptionLists.plist
Migraine
mipearse@Mikes-MBP ~ % ls -la0 /Library/Apple/System/Library/Extensions/AppleKextExcludeList.kext/Contents/Resources/ExceptionLists.plist
-rw-r--r--  1 root  wheel  restricted  9 Mar  3 13:52 /Library/Apple/System/Library/Extensions/AppleKextExcludeList.kext/Contents/Resources/ExceptionLists.plist
mipearse@Mikes-MBP ~ %
```

## אוטומציה

בשלב זה כבר יכולנו לדווח לאפל על החולשה, אבל רצינו לבצע אוטומציה, שכן הרצת Migration (המתבצעת באמצעות אפליקציה בשם Migration Assistant) היא תהליך ידני מאד שגם גורם לניתוק המשתמשים המחוברים - כלומר, החולשה שמצאנו טובה מספיק רק עבור תוקף עם גישה פיזית. לכן, החלטנו להשקיע מחקר נוסף ולבדוק האם ניתן לגרום לחולשה לרוץ ללא גישה פיזית, בהנחה שהתוקף רץ מרחוק. להלן רצף האירועים בביצוע Migration:

א. אפליקציית ה-Migration Assistant משתמשת בכלי בשם Setup Assistant לצורך התחלת הניוד, כאשר תהליך בשם MBSystemAdministration מתווך ביניהם על ידי שימוש ב-XPC. כמו כן, ניתוק המשתמשים הפעילים מתבצע בשלב זה על ידי קריאה למתודה בשם:

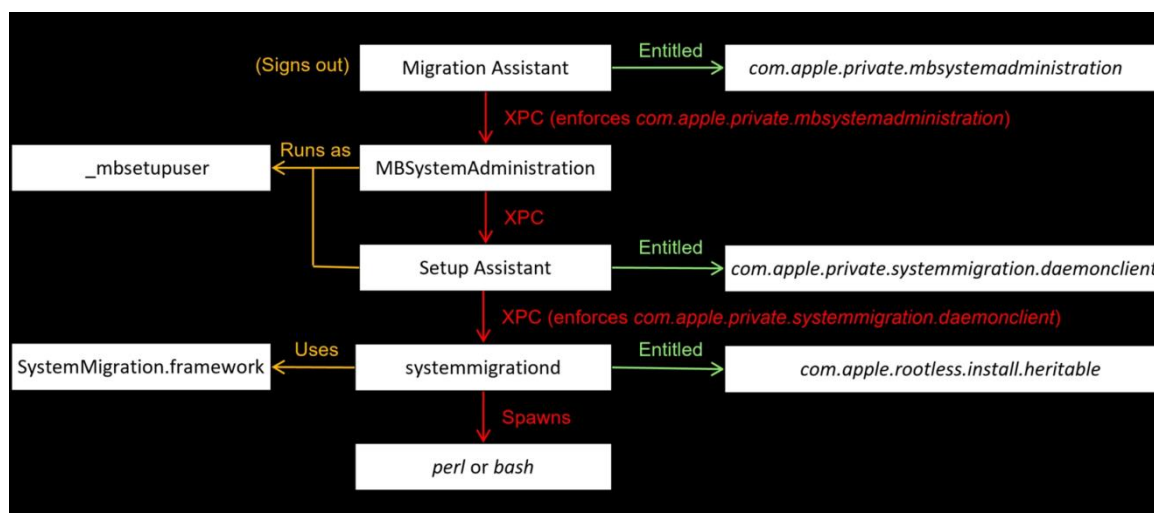
```
SACLOStartLogoutWithOptions
```

ב. כפי שציינו, תהליך ה-MBSystemAdministration מתווך בין Migration Assistant ו-Setup Assistant. תהליך זה מוודא שהתהליך הקורא (Migration Assistant במקרה שלנו) מחזיק ב-Entitlement בשם com.apple.private.mbsystemadministration - אחרת הוא יסרב לשרת את הבקשה. באופן מעניין, מכיוון שכל המשתמשים הפעילים נותקו בשלב זה, MBSystemAdministration רץ בתור משתמש חבוי בשם \_mbsetupuser המאפשר ביצוע פעולות UI גם לאחר שכל המשתמשים נותקו.

ג. כלי ה-Setup Assistant מקבל בקשות דרך MBSystemAdministration ומבצע בקשות XPC בעצמו לשירותי Mach שונים שממומשים על ידי systemmigrationd, שזהו בדיוק ה-daemon שיכול לעקוף אכיפת SIP. אותו systemmigrationd מוודא שהתהליך הקורא (Setup Assistant) מחזיק ב-Entitlement בשם com.apple.private.systemmigration.daemonclient - אחרת הוא יסרב לבקשה.

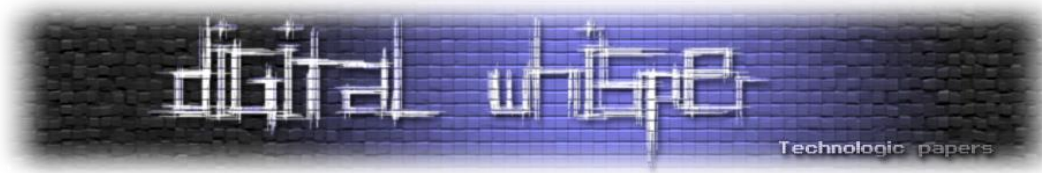
כדאי לציין ש-systemmigrationd גם הוא, עושה שימוש רבות ב-Private Framework בשם SystemMigration.framework, ואותו framework מממש שרת הממתין לבקשות (המימוש הוא במתודה בשם startListeningForConnections). הבקשות מגיעות על גבי מערכת הקבצים תחת התיקיה /Library/SystemMigration/Queue (המוגנת על ידי SIP). קבצים בתיקיה זו מתארים את בקשת הניוד, ולאחר ניטור ישנו את שמם לקובץ בשם "In-Flight". זהו השלב בו הניוד עצמו מתבצע, כולל קריאה לסקריפטים שירוצו על ידי perl או bash.

להלן תיאור סכמטי המסכם את תהליך הניוד:



אם כן, המטרה היא לדלג על ביצוע ההתנתקות על ידי Migration Assistant. הנסיונות הראשונים שלנו כללו בעיקר ביצוע Patching מסוגים שונים, אך ללא הצלחה - למשל, ביצוע Patching ל- SACLOStartLogoutWithOptions למשל יגרום לקריסה כמעט מיידית עקב [Pointer Authentication Code](#). נסיונות דומים גם לא צלחו - למשל, ב-macOS בינארים רבים מכילים [כמה גרסאות המקומפלים לארכיטקטורות שונות](#) (למשל Intel ו-ARM), ושליפת הגרסה המתאימה לארכיטקטורה שעליה רצים גורמת באופן דומה לכשלון (הפעם הסיבה היא ש-Migration Assistant מאבד את ה-Entitlement הדרוש עבורו).

רעיון נוסף שעלה לצוות הוא בחינת שרשרת האירועים באופן מדוקדק יותר - גילינו כי Setup Assistant עם ארגומנט MiniBuddyYes-, ותהינו מהי משמעותו. התקווה שלנו היא שנוכל להריץ את Setup Assistant ישירות ועל ידי כך לעקוף לגמרי את ניתוק המשתמשים הפעילים.



## בחינת המימוש של Setup Assistant חשפה מספר דגלים מעיינים:

```
54 if ( (unsigned int)objc_msgSend(v14, "isEqualToString:", CFSTR("-Min1BuddyYes")) )
55 {
56     v15 = self;
57     v16 = "setPrimaryBuddyType:";
58     goto LABEL_12;
59 }
60 if ( (unsigned int)objc_msgSend(v14, "isEqualToString:", CFSTR("-MigrationBuddyYes")) )
61 {
62     v17 = self;
63     v18 = "setPrimaryBuddyType:";
64 LABEL_23:
65     objc_msgSend(v17, v18, 2LL);
66     goto LABEL_13;
67 }
68 if ( (unsigned int)objc_msgSend(v14, "isEqualToString:", CFSTR("-EOSBuddyYes")) )
69 {
70     -[MacBuddyAppDelegate setPrimaryBuddyType:](self, "setPrimaryBuddyType:", 3LL);
71     goto LABEL_13;
72 }
73 if ( (unsigned int)objc_msgSend(v14, "isEqualToString:", CFSTR("-ResumeBuddyYes")) )
74 {
75     v17 = self;
76     v18 = "setSecondaryBuddyType:";
77     goto LABEL_23;
78 }
79 if ( (unsigned int)objc_msgSend(v14, "isEqualToString:", CFSTR("-XCUIest")) )
80 {
81     firstLoginContext = self->firstLoginContext;
82     self->firstLoginContext = (NSString *)CFSTR("kShouldUseTestValues");
83     objc_release(firstLoginContext);
84 }
```

בנוסף, גילינו מתודה בשם useDebugParameters שאף היא מחפשת דגל (הפעם בשם MBDDebug-):

```
1 bool __cdecl -[MacBuddyAppDelegate useDebugParameters](MacBuddyAppDelegate *self, SEL a2)
2 {
3     // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]
4
5     if ( getuid() )
6         return 0;
7     v4 = +[NSProcessInfo processInfo](&OBJC_CLASS__NSProcessInfo, "processInfo");
8     v5 = objc_retainAutoreleasedReturnValue(v4);
9     v6 = -[NSProcessInfo arguments](v5, "arguments");
10    v7 = objc_retainAutoreleasedReturnValue(v6);
11    objc_release(v5);
12    v8 = (unsigned int)-[NSArray containsObject:](v7, "containsObject:", CFSTR("-MBDebug"));
13    v3 = v8;
14    if ( !v8 )
15        goto LABEL_30;
16    v29 = v8;
17    -[MacBuddyAppDelegate setRunningAsDebug:](self, "setRunningAsDebug:", 1LL);
18    -[MacBuddyAppDelegate setPrimaryBuddyType:](self, "setPrimaryBuddyType:", 0LL);
19    -[MacBuddyAppDelegate setSecondaryBuddyType:](self, "setSecondaryBuddyType:", 0LL);
20    v33 = 0u;
21    v34 = 0u;
22    v31 = 0u;
23    v32 = 0u;
24    v30 = v7;
25    v9 = objc_retain(v7);
26    v10 = -[NSArray countByEnumeratingWithState:objects:count:](
27        v9,
28        "countByEnumeratingWithState:objects:count:",
29        &v31,
30        v35,
31        16LL);
32    if ( !v10 )
33        goto LABEL_27;
34    v11 = v10;
    v12 = *( OWORD *)v32;
```



באופן מפתיע, הרצה של Setup Assistant עם הדגל MBDebug - אכן גורמת לניוד ללא ניתוק! דגל נוסף בשם ResumeBuddyYes - מדלג אפילו על כמה שלבי UI. מכיוון שעדיין תוקף צריך לבצע שלבי UI, השתמשנו ביכולות אוטומציה (עם Apple Script) כדי לבצע לחיצות אוטומטיות (בדומה ל-AutoIT ב-Windows). לפיכך, ההשמה פשוטה:

1. הכנה של Time Machine Backup בגודל של 1GB, לצורך ניוד, וביצוע mount שלו עם hdiutil.
2. הכנה של payload שירוץ ללא הגבלות SIP. בהדגמה שלנו כתבנו פשוט את המילה Migraine לתוך קובץ, אבל ניתן להריץ כל דבר.
3. הגדרת משתנה הסביבה PERLSOPT באמצעות launchctl על מנת לגרום ל-payload שלנו לרוץ כאשר perl מתחיל.
4. הרצת Setup Assistant עם הדגלים MBDebug ו-ResumeBuddyYes.
5. הרצת Apple Script שמבצע באופן אוטומטי לחיצות, כך שנבחרת האפשרות "From a Mac, Time Machine backup or Startup Disk" ולאחר מכן לחיצה על Continue מספר פעמים.

## השלכות החולשה ותיקונה

לעקיפת SIP יש השלכות משמעותיות על מערכת ההפעלה, שכן תוקף יכול לבצע פעולות מסוכנות רבות:  
א. יצירת קבצים בלתי ניתנים למחיקה: זוהי ההשלכה הישירה ביותר ומעניינת מאד יצרני Endpoint Protection כגון [Microsoft Defender](#). קבצים המוגנים על ידי SIP בלתי ניתנים למחיקה באופן אוטומטי, ועם המעבר של אפל מ-Kernel Extensions לתשתית בשם [Endpoint Security Framework](#), באופן פרקטי לא קיים פתרון לבעיה שבה malware עוקף SIP.

ב. הרחבת משטח התקיפה באופן משמעותי, כולל ל-kernel: [המאמר המעולה של Mickey Jin](#) מתאר כיצד ניתן להריץ קוד kernel בהנתן עקיפת SIP. באופן דומה לנקודה הקודמת, ההשלכה של מעבר ל-Endpoint Security Framework היא שקוד זדוני שרץ ב-kernel בלתי ניתן לניטור הולם.

ג. יצירת rootkit: תוקף שיכול לטעון קוד kernel יכול לממש rootkit על כך המשתמע מכך - החבאת קבצים ותהליכים, החבאת חיבורי רשת ועוד. ספציפית, Microsoft Defender מכילה פיצ'ר בשם [Tamper Protection](#) המגן על התהליכים והקבצים הרלוונטיים עבורו, והחשש הוא עקיפת ה-Tamper Protection, למשל.

ד. עקיפת TCC: בנוסף לטעינת קוד kernel, [המאמר של Mickey Jin](#) מתאר כיצד ניתן לעקוף מנגנון הגנה אחר בשם TCC. מנגנון זה אחראי להגנה על פרטיות משתמש הקצה, כולל גישה לקבצים פרטיים, למצלמה, למיקרופון, לשירותי מיקום ועוד.

## סיכום

פנינו לחברת אפל באופן דיסקרטי ולאחר מספר חודשים [החולשה תוקנה](#) תחת [CVE-2023-32369](#). ב-18 במאי 2023. אנחנו מודים לאפל על התיקון וממליצים לכל המשתמשים לוודא שמערכת ההפעלה שלהם מעודכנת.

באפריל 2023, לאחר התיקון ב-Beta, הצוות שיתף פעולה עם אפל לוודא שהחולשה נסגרה באופן הרמטי. מעניין לגלות מספר תיקונים משמעותיים מאד:

1. אפל החליטה לחסום לגמרי שינוי `global environment variables` כאשר SIP דלוק. באופן כללי לא ניתן להשפיע על המשתנים של `launchd`. זהו פתרון "ברוטאלי" אבל הוא פותר באופן אפקטיבי מאד מחלקה שלמה של חולשות המתבססות על הרעלה של משתני סביבה.

2. אפל הוסיפה `Launch Constraint` על `systemmigrationd` כך שלא ניתן לטעון אותו באמצעות `Launch Daemon` שונה מההגדרה המקורית. הסיבה לכך היא שקבצי `plist` המגדירים `launch daemons` מכילים גם משתני סביבה הניתנים להגדרה מלאה, והתוכנית של הצוות הייתה להגדיר משתנה סביבה עבור `systemmigrationd`. זהו איננו מאמר על `Launch Constraints` - לרקע מומלץ לקרוא [סיכום ראשוני על ידי Csaba Fitzl](#). באופן דומה, גם פתרון זה פותר מחלקה גדולה של חולשות פוטנציאליות.

3. המימוש של `systemmigrationd` כעת מסיר את יכולות עקיפת ה-SIP לפני הרצת סקריפטים (בדומה לדרך שבה `drop_sip` עובד). זהו תיקון טקטי אך משמעותי - הצוות הצליח עדיין לעקוף את הפתרונות האחרים על ידי שינוי `dependencies` של `perl` ללא שינוי משתני סביבה, אך כאמור - עקב תיקון טקטי זה הצוות לא הצליח לבצע עקיפת SIP.

צוות המחקר מאמין שהחולשה נסגרה באופן הרמטי על ידי אפל, יחד עם צמצום משמעותי של משטח התקיפה על SIP.

## על הכותב

יונתן בר אור (Jonathan Bar Or, "JBO") חוקר בחברת מייקרוסופט תחת Microsoft Defender ובאופן רשמי ארכיטקט המחקר למערכות הפעלה שאינן Windows.

כותב פה ושם בטוויטר תחת [@yo\\_yo\\_yo\\_jbo](#).



# 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 ובשמה 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

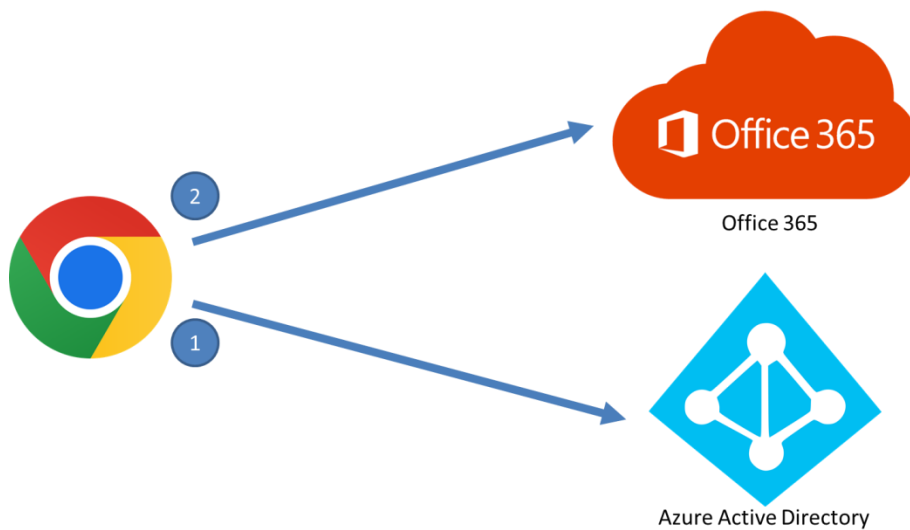
2. אימות המשתמש Authentication

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.

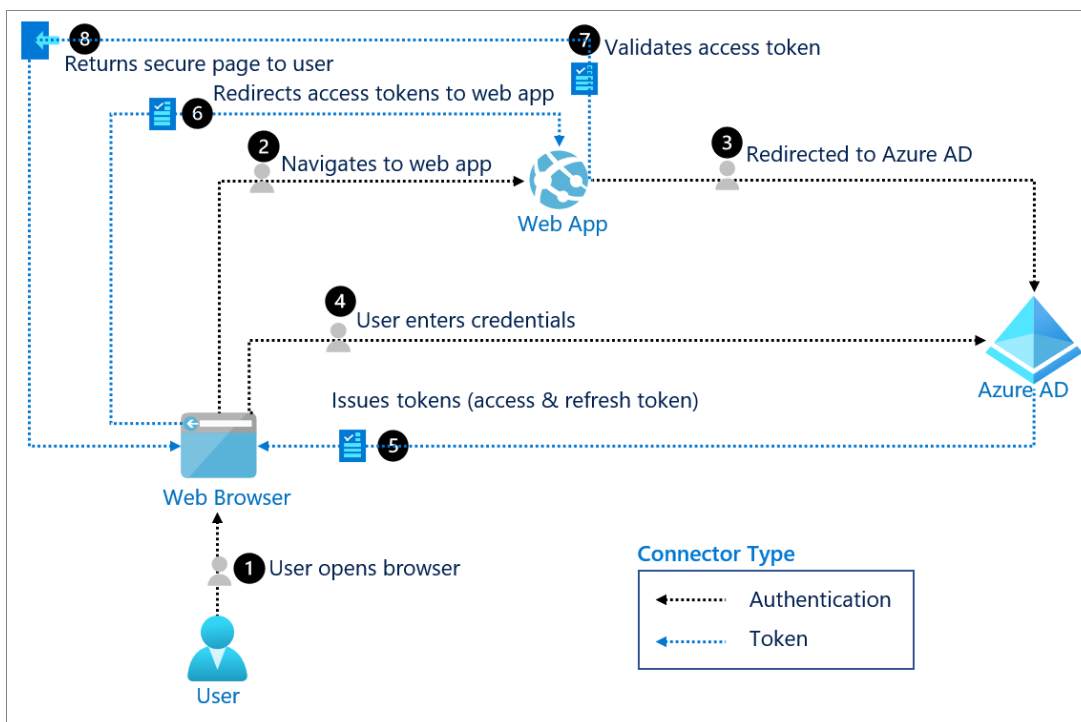
בואו נבחן את התרחיש הזה רגע:



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. Identity provider, ה-Azure AD

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



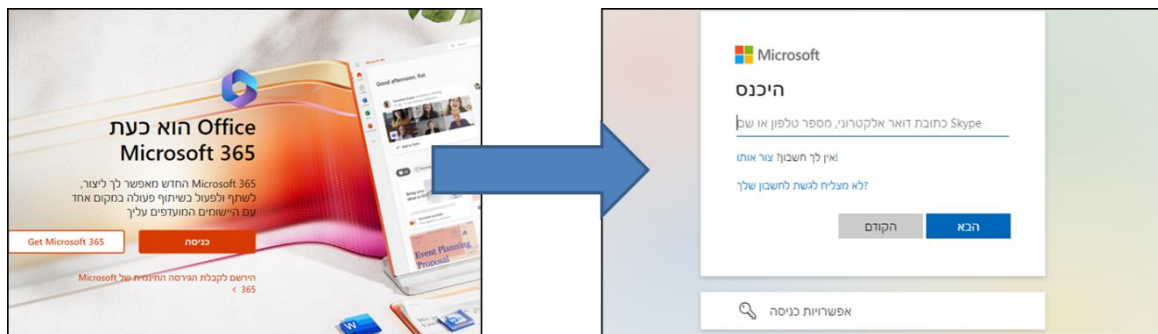
**פירוט השלבים**

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.

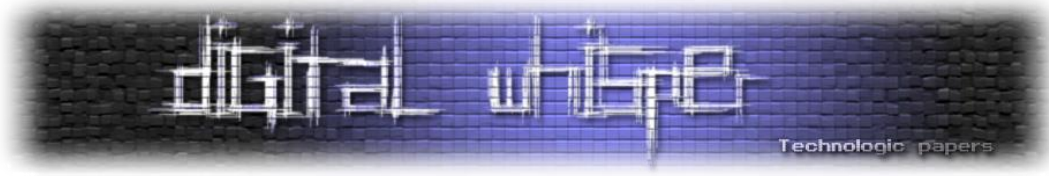
לאחר מכן הדפדפן יפנה אותנו בחזרה לעמוד האפליקציה, שתבצע ולידציה ל-Token שלנו ובמידה והוא תקין תחזיר לנו את העמוד המבוקש. פשוט, לא? אז אפשר לצלול קצת יותר לעומק.

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.





ניתן לראות אותו כאן: Overview > Identity > Home

Home >

## Default Directory

+ Add Manage tenants What's new Preview features Got feedback?

Azure Active Directory is becoming Microsoft Entra ID. [Learn more](#)

Overview Monitoring Properties Recommendations Tutorials

Search your tenant

### Basic information

Name	Default Directory	Users	1
Tenant ID	5c93733a-13df-4f58-abc7-fd9226f855c3	Groups	0
Primary domain	sapxfedoutlook.onmicrosoft.com	Applications	0
License	Azure AD Free	Devices	0
Workload License	Azure AD Workload Free		

התשובה שמתקבלת בבקשה הזו היא:

```

1 {
-  "token_type": "Bearer",
-  "scope": "https://management.core.windows.net//user_impersonation https://management.core.windows.net",
-  "expires_in": 4969,
-  "ext_expires_in": 4969,
-  "access_token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiIsIng1dCI6Ii1LSTNROW50UjdiUm9meG11w9YcwjIiwkdldyIS",
-  "refresh_token": "0.Aa4A0n0TXN8TWE-rx_2SjvhVw4NAS8Swo8FJtH2XT1PL3zyuAD8.AgABAAEAAAAtyolD0bpQQ5Vt1I4uG",
-  "id_token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiIsImtpZCI6Ii1LSTNROW50UjdiUm9meG11w9YcwjIiwkdldyIj9.eyJ",
-  "client_info": "eyJ1aWQlOiIwMDAwMDAwMCAwMDAwLTAwMDA0YQYyZS1jZDQyYmNINTRmODAlLCJ1dG1kIjoioTE4ODAwMGQ0N"
- }
    
```

לאחר שלקוח יש את ה-Access token הוא יכול להשתמש בו ב-Header הבקשות שלו ובכך להציג שהוא מאומת ואת ההרשאות שלו.



## 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
```

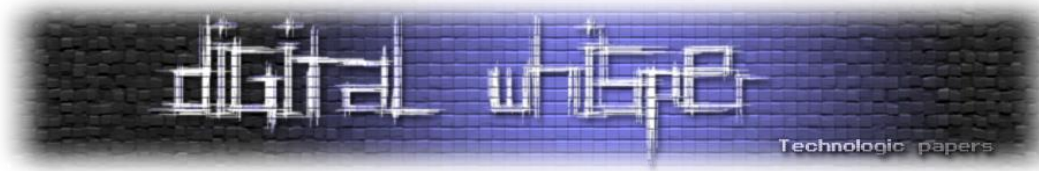
חשוב להדגיש ש-JWT יכול להיות כמה דברים:

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

### JWS

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

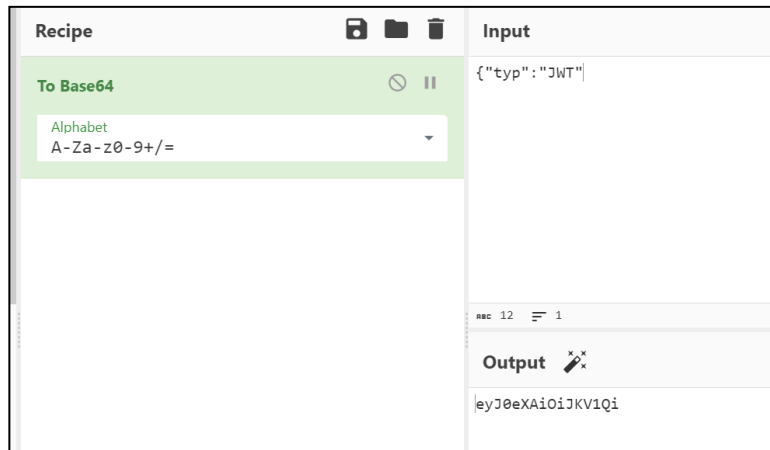
- JOSE - JavaScript Object and Encryption Header
- Payload (בדרך כלל הסטרינג שמזהה את הלקוח בפורמט JSON)
- Signature (כביכול אופציונאלי אבל למזלנו ב-Azure זה תמיד ממומש)



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:



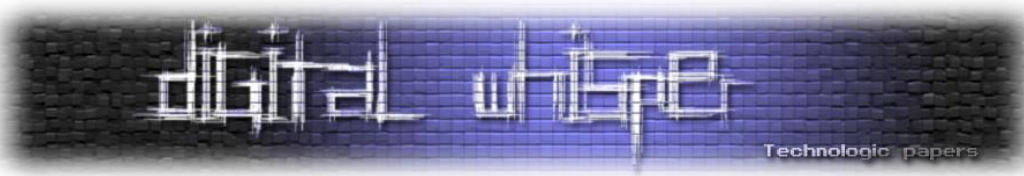
אולי זה לא נראה משמעותי עכשיו, אבל היכולת לזהות סטרינגים כ-Access Tokens היא ממש שימושית, ונראה את זה בהמשך.

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/0aGxk9fLFkt03gRp531pTyuIcCjwjWfdgGQ=",
  "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_2S3vhVw0ZIf3kAutdPukPawfj2MBOuAD8.",
  "scp": "user_impersonation",
  "sub": "vVn52NPPke7jpaCAqF-XgHXfNdq0iNX56w4W3Nk8j24",
  "tid": "5c93733a-13df-4f58-abc7-fd9226f855c3",
  "unique_name": "live.com#sapxfed@outlook.com",
  "uti": "8_RONa2CmUjPaVw4gFAUAA",
  "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
  - exp/nbf - תאריך התחלה וסיום של תוקף ה-Token
  - scp - ה-Scope של ה-Token, הרשאות
- הנה דוגמא יותר מעניינת לשדה הזה מזו שבתמונה:

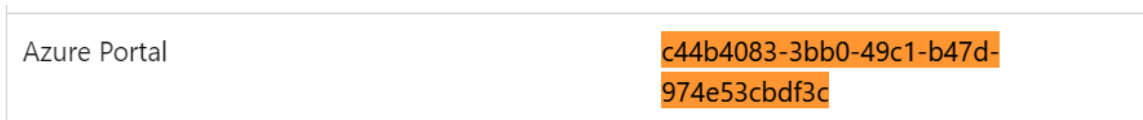
```
"scp": "AccessReview.ReadWrite.All AuditLog.Read.All
ConsentRequest.Create ConsentRequest.Read
ConsentRequest.ReadApprove.All
ConsentRequest.ReadWrite.All
CustomSecAttributeAuditLogs.Read.All
Directory.AccessAsUser.All Directory.Read.All
Directory.ReadWrite.All Directory.Write.Restricted
DirectoryRecommendations.Read.All
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

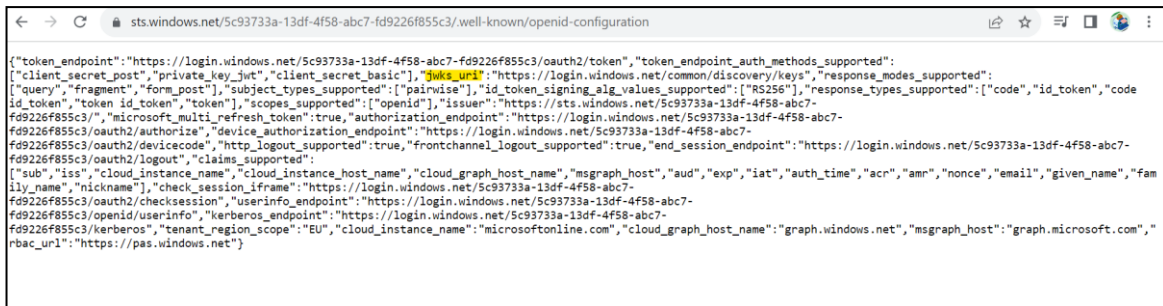
בסוף אנחנו כאן בשביל אותנטיקציה, אז איך מתבצע תהליך אימות החתימה? נכנסים ל-Openid Configuration שימצא ב-URL הבא:

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

במקרה שלנו זה יהיה הנתביב:

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

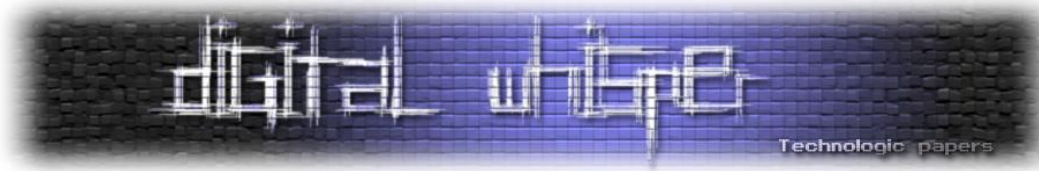
שם נמצא את ההגדרה הבאה - `jwtks_uri`:



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": "w...nR7bRofmeZoXqbHZGew", "e": "AQAB", "x5c": "IF9k3KjpTyOyY25-9_DrDvPGjY1Qk0iCjBfjyQAL-pBec9r-XkAS-C4zTn1ZRw--GELabuo8U-U6s3TKj42xPDEP-R5Rp0GshcC9S1k1USteuuh4F8M3XFR2GB8c..."}, {"kty": "RSA", "use": "sig", "kid": "220p3UupbjAYKYGaXE3181V0TOI", "x5t": "220p3UupbjAYKYGaXE3181V0TOI", "n": "w...nR7bRofmeZoXqbHZGew", "e": "AQAB", "x5c": "IF9k3KjpTyOyY25-9_DrDvPGjY1Qk0iCjBfjyQAL-pBec9r-XkAS-C4zTn1ZRw--GELabuo8U-U6s3TKj42xPDEP-R5Rp0GshcC9S1k1USteuuh4F8M3XFR2GB8c..."}]
```

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/>

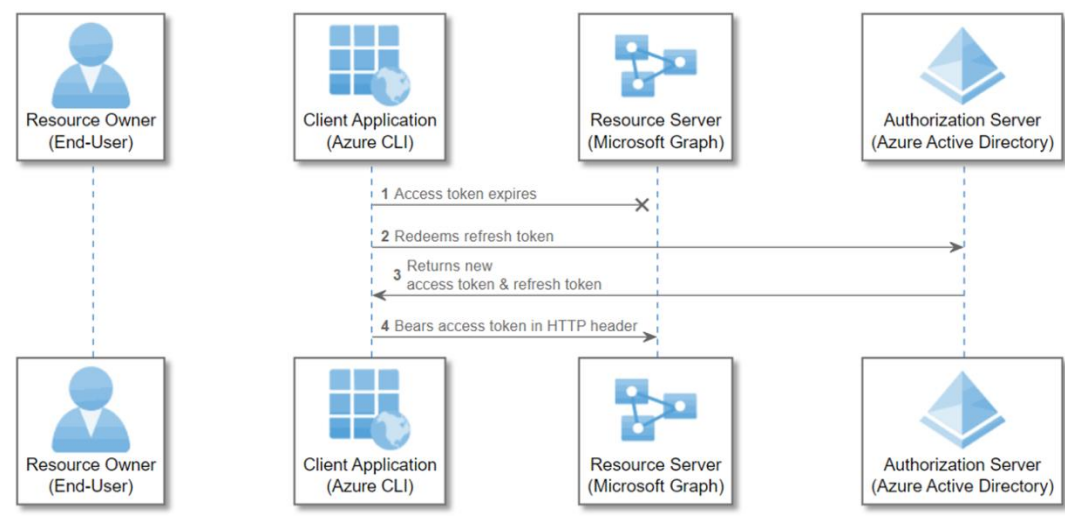
או לחילופין באמצעות הפקודה:

```
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:





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 ולא את 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.

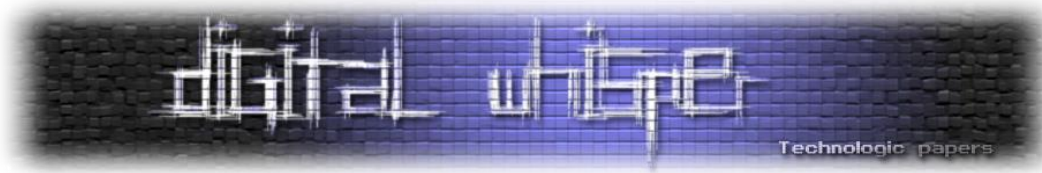
במחקר מצאו סך הכל 15 אפליקציות שניתן להפיק מה-Token שלהן Token חדש לאפליקציות אחרות(מתוך ה-15). אז מה הן האפליקציות האלו? ולמה רק איתן ההתנהגות הזו מתקיימת?

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+blwH-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-AADIntAccessTokensForAADGraph -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 פייסינג שעושה בקשת POST ל-URI הזו עם הקוד, וכל מה שהקורבן צריך לעשות הוא רק ללחוץ ולאשר. השיטה הזו נורא נפוצה בימים אלו.

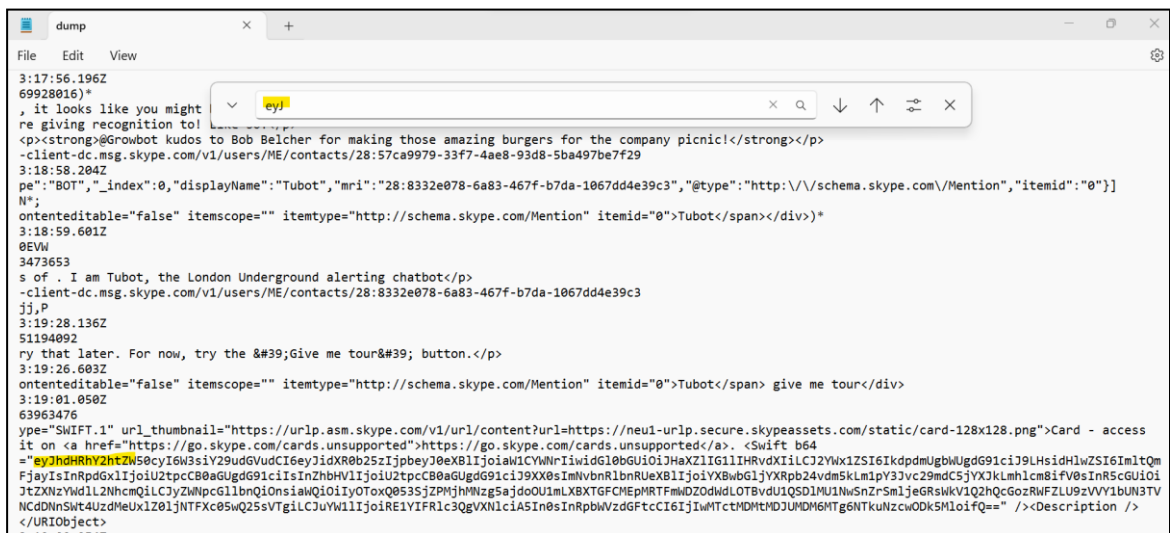
לאחר האישור, התוקף בבית מקבל את ההרשאות שהיה צריך וכעת הוא יכול להשתמש ב-Access token שקיבל ולבצע באמצעותו קריאות API!

### 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!

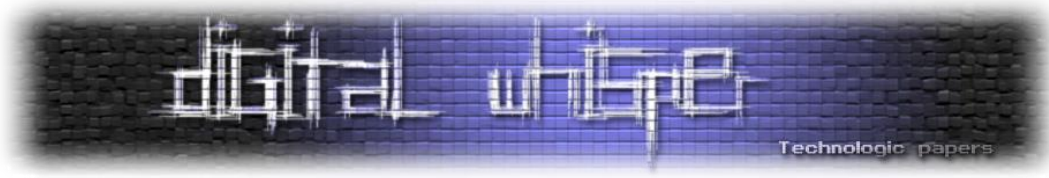
זוכרים שאמרתי שזה שימושי שכל Access token תמיד מתחיל באותה צורה? זה הרגע להוכיח את התועלת שבדבר.

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:

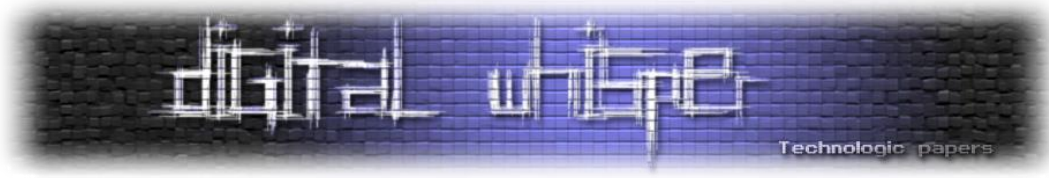
GetAccessTokenForResource

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

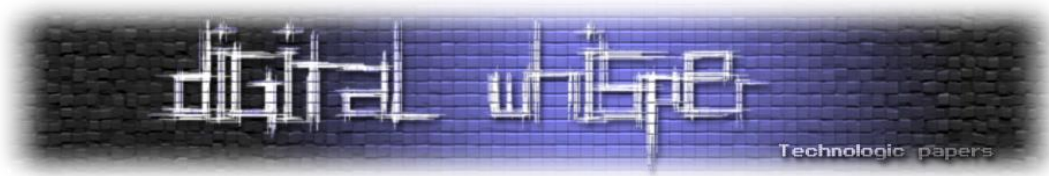
<https://aadinternals.com>

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>

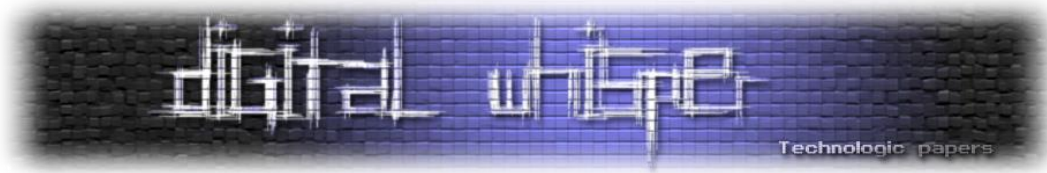
RFC-ים:

<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/>



# מבוא למחקר אפליקציות

מאת עידן שכטר

## הקדמה

מחקר בעולם הסייבר הינו עולם דינמי, מגוון ומרגש, המציב בפני חוקרים אתגרים המתחדשים בהתאם לעולם טכנולוגי הדוהר קדימה. אחד מעולמות התוכן הרלוונטיים ביותר בימינו, הוא עולם ה-Mobile, מעצם הפיכתם של מכשירים סלולריים לחלק בלתי נפרד מחיינו.

במאמר זה נחשף לעולם מחקר האפליקציות בסביבת Android, נכיר כלי מחקר פרקטיים ונכנס לראשו של חוקר בתהליך מציאת פתרון לבעיה. תוכן המאמר מהווה בסיס מעולה לכל המעוניין לצלול אל עולם מחקר האפליקציות בסביבת Android ולפרוח ממנו לתחומים נוספים...

חשוב לציין, כי המאמר רלוונטי לכל המתעניין בתחום! אך לצורך חוויה מיטבית, אמליץ על היכרות עם שפת Java ועקרונותיה הבסיסיים וכך גם לגבי JavaScript, כמו כן ידע בפרוטוקולי תקשורת ושליטה במערכות מבוססות Linux.

## הקמת סביבה

על מנת לעסוק באתגר, עלינו להקים סביבת מחקר שבין היתר תאפשר לנו להתקין ולהריץ את ה-APK הרלוונטי. לשם כך, נצטרך מכשיר המריץ Android עם הרשאות Root המאפשר USB Debugging, או לחילופין, Emulator כלשהו המספק את אותו מענה.

במאמר זה לא נעסוק בהשגת הרשאות Root על מכשיר Android, אך נוכל למצוא שלל מדריכים באינטרנט בהתאם למכשיר שבידינו. ע"מ להדליק את פיצ'ר ה-USB Debugging במכשיר נפעל ע"פ הצעדים הבאים:

1. Settings ← About Phone ← Software Information ונלחץ על Build number מס' פעמים, עד אשר נראה הודעה המציינת כי הדלקנו בהצלחה את ה-Developer mode במכשיר.
2. נחזיר אל ה-Settings של המכשיר, כעת נראה לשונית חדשה בשם "Developer options", נכנס אל הקטגוריה החדשה, נגלול עד ל-Debugging, ונדליק את אופציית ה-USB debugging.

או במידה ומכשיר שכזה אינו בהישג יד, ניתן להשתמש באחד מהפתרונות הבאים:

1. יצירת מכשיר וירטואלי מבוסס ARM ב-[Android Studio](https://www.digitalwhisper.co.il)



2. יצירת מכשיר וירטואלי באמצעות Genymotion, והתקנת ARM Translation (מאחר וה-Image ש-Genymotion מייצר מחכה ארכיטקטורת x86, בעוד ה-APK מיועד לארכיטקטורת ARM)
- a. מורידים Genymotion [מהקישור](#), מתקינים, ויוצרים מכשיר וירטואלי עם Android 9.
- b. מתקינים [ARM Translation](#) על המכשיר הוירטואלי (גוררים את ה-Zip הרלוונטי (for\_9.0...)) אל המסך של המכשיר הוירטואלי, או לסירוגין משתמשים ב-adb install.
- תוכנה נוספת שנצטרך היא [Android Studio](#), בו נשתמש כדי לקרוא ולחקור את קוד ה-APK הרלוונטי.

## האתגר

כעת, כשיש לנו Setup מוכן, נוכל להתקין את ה-APK הרלוונטי על המכשיר שברשותינו, ובהזדמנות זו גם נכיר את הכלי הראשון שנפגוש במאמר זה, והוא: ADB.

### Android Debug Bridge (ADB)

מדובר בממשק המספק נתיב תקשורת אל מול מכשירי Android המאפשר ביצוע פעולות מגוונות, כגון דיבוג מרחוק, צפייה בלוגים, הנגשת Shell וכו'.

במאמר זה, נשתמש בכלי בעיקר כדי לבצע פעולות בסיסיות כמו הנחה ומשיכה של קבצים, אך אמליץ בחום לחקור את הכלי לעומק ולהכיר את היכולות השונות שלו!

ADB מגיע כחלק מ-Android SDK Platform Tools, אותו נוכל להוריד ולהתקין בהתאם למערכת ההפעלה שלנו [מהקישור כאן](#).

הפקודה הראשונה אותה נכיר, היא adb devices, אשר מציגה בפנינו רשימה של כל מכשירים ה-Android המחוברים למחשב שלנו (פיזיים ווירטואליים כאחד):

```
user@user:~$ adb devices
List of devices attached
RFCR10H6BDA    device
```

נוכל לראות כי ישנו מכשיר מחובר אחד בעל המזהה RFCR10H6BDA. מאחר ושינו רק מכשיר אחד מחובר, השימוש במזהה אינו רלוונטי, לעומת זאת, במקרה בו ישנם מספר מכשירים מחוברים, נוכל להשתמש בדגל -s על מנת לציין את ה-UUID של המכשיר עליו נרצה לבצע את הפעולה המבוקשת.

לאחר שראינו שהמכשיר שלנו מחובר, נרצה להתקין את ה-APK של Shazam Lite על ידי שימוש בפקודה install:



```
user@user:~$ adb install com.shazam.android.lite_1.1.0-170321-101040_minAPI9\(\arm64-v8a\)\(nodpi\)_apkmirror.com.apk
Performing Streamed Install
Success
```

במידה והיו מספר מכשירים מחוברים, היינו מריצים את הפקודה על דגל -s בצירוף ה-UUID של המכשיר הרלוונטי:

```
user@user:~$ adb -s RFCR10H6BDA install com.shazam.android.lite_1.1.0-170321-101040_minAPI9\(\arm64-v8a\)\(nodpi\)_apkmirror.com.apk
Performing Streamed Install
Success
```

במידה והאפליקציה הותקנה בהצלחה, נוכל לראות את האייקון שלה בדף הבית / רשימת היישומים של המכשיר.

פקודות adb שימושיות נוספות:

- קבלת ממשק Shell על המכשיר:

```
adb shell
```

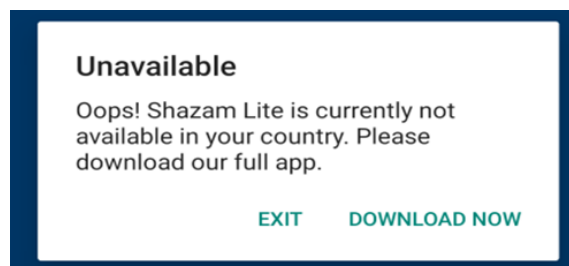
- "העלאת" קובץ למכשיר:

```
adb push <source> <destination>
```

- "משיכת" קובץ מהמכשיר:

```
adb pull <source> <destination>
```

כשנססה להריץ את האפליקציה, נתקל בשגיאה הבאה:



ובכן, נראה שהאפליקציה לא נתמכת במדינה שלנו - באסה. אבל מאחר ואנחנו חוקרים, לא ניתן לשגיאה הזו לעצור אותנו, המטרה שלנו - לזהות את המנגון הרלוונטי ולמצוא דרך לעקוף אותו!



## הבנת הבעיה

אנו מבינים כי קיים מנגנון מסויים באפליקציה, שמטרתו לזהות את המדינה בה נמצא המכשיר, ולאנוף את השימוש באפליקציה בהתאם, מכאן עולה השאלה - כיצד המנגנון עובד?

אפשרות אחת היא שה"אכיפה" מבוצעת בצד הלקוח, כלומר, האפליקציה מבצעת את הבדיקה בעצמה, ומציגה את ההודעה הרלוונטית בהתאם.

אפשרות נוספת, היא שהבדיקה מבוצעת בצד שרת, כלומר, האפליקציה אוספת נתונים מסויימים אודות המכשיר, שולחת אותם לשרת, והוא זה שמחליט איך צריך לפעול בהתאם לנתונים. לבסוף, השירות ישיב לאפליקציה עם ההחלטה שהתקבלה, וזאת תציג את ההודעה בהתאם.

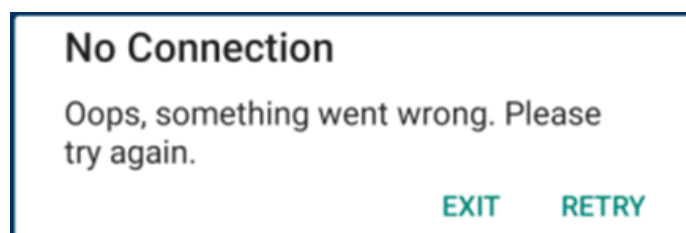
היתרון המשמעותי של האפשרות השנייה, בה השרת הוא זה שמבצע את הבדיקה, היא שאופן הפעולה של המנגנון אינו חשוף למשתמש, אלא רק המידע הנשלח \ מתקבל מהמנגנון, דבר המקשה על הבנת המנגנון, ולכן עשוי להקשות על מציאת פתרון.

נתבונן בפתרונות העומדים בפנינו:

- מנגנון ממומש בצד לקוח ← נזהה את המנגנון באפליקציה ← נשבש את אופן הפעולה שלו לטובתנו ← ניצחון
- מנגנון ממומש בצד שרת ← נזהה את המידע הנשלח \ מתקבל מהמנגנון ← ננסה להבין מה המידע ש"מפליל אותנו" (במידה ולא נצליח לזהות את המידע, נצטרך למצוא דרך להבין כיצד "מידע לגטימי" [כזה שעבורו המנגנון יחזיר תשובה חיובית] נראה, ולהשתמש בו, או לחילופין, לייצר אותו בעצמנו) ← נשנה את המידע המפליל הנשלח לשרת ונחליף אותו במידע לא מפליל ← ניצחון.

רגע לפני שננסה להבין מול איזו סיטואציה אנו נמצאים, נתעכב על נקודה חשובה - כחוקרים, נרצה לחקור כמה שפחות ונשאף למצוא פתרון הפשוט \ מהיר ביותר (אם הדבר אכן אפשרי). לכן, רגע לפני שנרוץ לפתרונות קצת יותר "מורכבים" כמו לרוורס את ה-APK (אל דאגה, גם לשם נגיע!) נחשוב על דרך פשוטה לענות על השאלה הבאה - האם המנגנון ממומש בצד שרת, או בצד לקוח?

פעולה פשוטה שיכולה לעזור לנו לענות על שאלה זו, היא לנסות לפתוח את האפליקציה כשאנחנו לא מחוברים לאינטרנט, מה שיכול להכניס אותנו ל-"Flow" שונה באפליקציה, כלומר, לגרום לאפליקציה להתנהג אחרת (לדוגמא, להציג הודעה שונה):





ואכן נגלה שנקבל שגיאה שונה. אבל רגע! האם זו הוכחה מספיק טובה? מה אם קיים באפליקציה מנגנון שקודם כל בודק אם אנחנו יכולים תחילה לגשת לשירות, ורק אז מבצע את הבדיקה הרלוונטית? (כלומר, הבדיקה תבוצע בצד הלקוח, אבל רק אחרי שווידא שהוא יכול לגשת לשירות) - זהו בהחלט תרחיש אפשרי!

על מנת לפתור את הסוגיה, נפעל בצורה הבאה - נבדוק האם המנגנון ממומש בצד שרת, ובמידה ונראה שלא, נוכל להסיק שהוא ממומש בצד הלקוח.

על מנת לבצע בדיקה זו, נצטרך למצוא דרך להאזין לבקשות שהאפליקציה שולחת לשירות, וכאן נכנס לתמונה HTTP Proxy.

## MITMProxy

פעמים רבות, כחלק מהליך מחקרי, נרצה לנתח את ההתנהגות הרשתית של אפליקציה מסויימת, בין היתר, כמו במקרה שלנו, על מנת להבין מנגנון העומד בדרכנו בצורה טובה יותר. את יכולת זו נשיג על ידי שימוש בשרת Proxy, זהו למעשה שרת ה"עומד" בין האפליקציה לבין השירות שלה, כך שהוא מקבל את ההודעות ישירות מהאפליקציה, ושולח אותן בשם האפליקציה לשרת, בדרך זו, נוכל לראות את הבקשות הנשלחות לשרת ואת התשובות להן (גם מעל HTTPS!) (כמובן, שגם פעולה זו לא בדיוק פשוטה, ותדרוש מאיתנו מספר מניפולציות נוספות).

למזלנו, לא נצטרך לכתוב שרת כזה בעצמנו ונוכל להשתמש בכלים קיימים, הכלי בו נשתמש הוא MITMProxy. MITMProxy הינו HTTP Proxy חינומי, מבוסס קוד פתוח, אינטראקטיבי, המציע ממשק (CLI & Web) קל (Lightweight) ופשוט לשימוש.

נוכל להשתמש בו כדי לבחון את הבקשות (HTTPS) הנשלחות מהאפליקציה לשירות, ולחפש בהן מידע שעשוי לעזור לנו בהבנת המנגנון (כמו כן, הרגישו חופשי להשתמש בפרוקסי המועדף עליכם).

([התקנת MITMProxy](#), הרצתו על ידי הפקודה `mitmproxy / mitmweb`, אמליץ להשתמש ב-`mitmweb` מאחר והממשק שלו חברותי יותר). על מנת לנתב את התעבורה שיוצאת מהמכשיר שלנו דרך MITMProxy, נצטרך לחבר את המכשיר איתו אנחנו עובדים לשרת שהקמנו, ולשם כך נצטרך שהמחשב איתו אנחנו עובדים והמכשיר בו אנחנו משתמשים ישבו על אותה רשת (במידה ומדובר באימולטור, זהו המצב הדיפולטי איתו אנחנו עובדים, ולכן כל מה שנצטרך לעשות זה להגדיר את כתובת ה-Proxy בהתאם ל-Host הרלוונטי) במידה ואנחנו עובדים ממכשיר פיזי, נוכל ליצור נקודת גישה אלחוטית במחשב איתו אנחנו עובדים, ולהתחבר אלייה ע"י המכשיר הרלוונטי.



הדבר האחרון שעלינו לעשות הוא [להתקין את ה-certificate של mitmproxy](#) על המכשיר איתנו אנחנו עובדים. לאחר שקינפנו והרצנו את MITMProxy בהצלחה, נריץ שנית את האפליקציה שנית.

רגע, מוזר! נראה שאנחנו שוב מקבלים שגיאת התחברות, כזו הזזה לשגיאה שקיבלנו כשאר פתחתנו את האפליקציה ללא חיבור לאינטרנט, אז מה בעצם קרה כאן?

## SSL Certificate Pinning

מנגנון אבטחה נפוץ בקרב אפליקציות מובייל, שבא לחזק את השימוש ב-SSL Certificates, הינו SSL Pinning. בפועל, המנגנון מוודא שה-Certificate שהאפליקציה מקבלת מהשירות איתו היא מנסה ליזום סשן מאובטח, הוא אכן ה-Certificate האותנטי של השירות, או במקרה שלנו, Certificate השייך לשירות של Shazam Lite, מאחר וחיברנו את האפליקציה לשרת Proxy, מנגנון ה-SSL Pinning המוטמע באפליקציה ימנע ממנה ליצור סשן מאובטח, מאחר וה-Certificate שתקבל הוא לא ה-Certificate של השירות. כלומר, "נועץ" את ה-Certificate המקורי של השירות (ומכאן השם).

אז איך נתגבר על מנגנון שכזה? בפועל, המנגנון מבצע בדיקה פשוטה - השוואה של ה-Certificate שקיבל מהשירות איתו הוא מנסה ליצור סשן מאובטח, אל ה-Certificate המקורי של השירות, המוטמע בקוד בדרך כלשי (לדוגמא, משווה ערכי Hash) פעולה זו צפויה לחזיר שתי תוצאות אפשריות - ה-Certificates זהים, או שלא, כלומר - True / False (כלומר, שחרור הנעץ - Unpinning).

כל מה שעלינו לעשות הוא לזהות את המימוש של המנגנון, ולגרום לו להחזיר את התשובה הרצויה עבור כל קלט שיקבל, אך איך נעשה זאת? במעמד זה נכיר כלי רב עוצמה, שבין היתר יפתור עבורנו את הבעיה בקלות, הלו הוא Frida!

## Frida

פרידה הינה כלי רב עוצמה המאפשר ניתוח ושינוי של יישומים בזמן ריצה (Dynamic Analysis) על ידי ביצוע Hook לפונקציות, (שינוי מימוש של פונקציה) קריאת וכתובה של ערכים בזיכרון ועוד. מדובר בכלי מחקרי לעולמת ה-iOS ו-Android כאחד (כולל תמיכה בספריות Native) אף על פי שראוי לכתוב על Frida מאמר שלם בפני עצמו, לא נצלול לעומק יכולות הכלי, אלא נשאר בגבול הגזרה שרלוונטי תרגיל זה.

בפועל, Frida מורכבת מ-2 חלקים, האחד הוא שרת הנמצא על המכשיר (קובץ בינארי) אשר מבצע מניפולציות בתהליך הרלוונטי. והשני, הוא הקליינט המנגיש ממשק סקריפטינג עשיר התומך במספר שפות, וכמו כן כלים נוספים (frida-ps, frida-trace, etc) (בפועל, הקליינט והכלים שצויינו הם ישויות נפרדות, אך הם חלק מחבילת ה-frida-tools). הקליינט והסרבר מבצעים תקשורת דו צדדית על מנת לבצע פעולות שונות, בפועל, הקליינט ישלח לשרת את הפעולה שנרצה לבצע (לדוגמא, סקריפט שנרצה להריץ), השרת יריץ אותו, ויחזיר לקליינט את הפלט הרלוונטי.

**הערה:** השם Frida מגיע מצירוף המילים Free IDA - יוצר הכלי, Ole André Vadla Ravnås, שאף ליצור אלטרנטיבה חינומית ל-IDA, התשלב יכולות ניתוח סטאטיות ודינאמיות כאחד, אך עם הימים הפוקוס השתנה לניתוח דינאמי בלבד. כיום, הכלי מתוחזק על ידי חברת NowSecure / אמליץ לקרוא על הכלי באתר הבית: <https://frida.re/docs/home>.

כדי להתקין את Frida נריץ את הפקודה:

```
pip install frida-tools
```

נוודא שההתקנה אכן הצליחה על ידי:

```
frida --version
```

בימים של כתיבת מאמר זה, גרסאת ה-Frida העדכנית ביותר הינה 16.0.7, והיא זו שתשמש אותי בפתרון האתגר, אך השימוש בכלי רלוונטי גם לגרסאות ישנות יותר, וכמו כן לגרסאות עתידיות!

כעת, נתקין את החלק השני, הסרבר. נכנס לדף ה-releases ב-repository של Frida ב-Github, ונחפש את קובץ השרת:

```
frida-server-{version}-android-{arch}.xz
```

כך ש-arch מייצג את הארכיטקטורה של המכשיר בו אנחנו משתמשים (אם מדובר במכשיר פיזי, נבחר ב-arm64, אך אם מדובר באימולטור, נבחר ב-x86). את version נתאים לגרסאת ה-client שהתקנו, זו שנקרא בהרצת הפקודה:

```
frida --version
```



**כמה מילים על תאימות:** נוכל להשתמש בצמד שרת + לקוח, כל עוד הם חולקים את אותה גרסת major, כלומר, שרת בגרסא 16.x.x יעבוד אל מול קליינט מגרסא 16.x.x

כפי שהבנו, Frida מאפשרת לנו בין היתר לבצע Hook לפונקציות, וזה בדיוק מה שאנחנו צריכים! במידה ונוכל לזהות את הפונקציה שמבצעת את בדיקת ה-Certificate, נוכל לשנות את ערך החזרה שלה בהתאם!

מאחר ומדובר בפעולה דיי נפוצה הממומשת על ידי ספריות מוכרות, לא נצטרך למצוא את הפונקציה בעצמנו, ונוכל להשתמש בסקריפטים קיימים (אל דאגה, נבצע Hook באמצעות Frida בהמשך)

סקריפט המבצע SSL Unpinning לאפליקציות Android נוכל למצוא ב-Codeshare של Frida (שבין היתר מכיל מגוון סקריפטים נוספים):

<https://codeshare.frida.re>

על מנת להריץ את הסקריפט הלוונטי, נשתמש בפקודה הבאה:

```
frida -U -f com.shazam.android.lite -l <script_name>
```

Frida תריץ את האפליקציה עבורנו תוך כדי ביצוע Hook לפונקציות המממשות מנגנוני SSL Pinning בספריות רלוונטיות מוכרות, נוכל לזהות בהדפסות הסקריפט שזוהי שימוש ב-TrustManager והותאם מעקף בהתאם (מאחורי הקלעים, Frida תבצע Hook לפונקציות הרלוונטיות).

אם נבחן את הסקריפט, נראה כי ה-Hook מבוצע על ה-Constructor של המחלקה "SSLContext" המשמשת את האפליקציה ליצירת חיבור מאובטח (TLS). אותו Constructor מקבל כפרמטר אובייקט מסוג TrustManager המכיל את ה-Certificate של השירות (אותו אנחנו "נועצים"), שבו יוכל להשתמש כדי לבצע את הבדיקה הרלוונטית בהמשך הריצה. על מנת "להתחמק" מתרחיש זה, נספק ל-Constructor אובייקט TrustManager ריק, לחילופין, נוכל להחליף את אובייקט ה-TrustManager בערך null לקבל תוצאה זהה.

הופה! אנחנו מקבלים את השגיאה הראשונה שוב! פתרנו את הבעיית ה-SSL Pinning. אם נביט ב-mitmproxy, נזהה בקשת HTTPS הנשלחת לשרתי האפליקציה, ומכילה JSON מעניין מאד ב-Body:

```
POST https://amp.shazam.com/configuration/v1/configure
...
```

נשים לב כי ה-Status code של התשובה שאנחנו מקבלים מהשרת הוא 403, כלומר, Forbidden, מה שמהווה רמז ענק לעובדה שהבקשה בה אנו דנים, היא הבקשה ש"מפלילה אותנו".

השגיאה שאנו מקבלים, זו שאומרת כי האפליקציה לא זמינה במדינה הנוכחית שלנו, היא רמז ענק לחלק המפליל בבקשה, נוכל לראות כי נשלחים 3 פרמטרים מעניינים, שקשורים באופן ישיר למדינה בה אנחנו נמצאים: mobileNetworkCode-I Country, mobileCountryCode.



משהו אומר לי שהפתרון שלנו קשור סביבם.

לדוגמא, ניתן לראות כי השדה "country" מחזיק את הערך "il" כלומר Israel, בנוסף השדה "[mobileCountryCode](#)" מחזיק את הערך 425, חיפוש מהיר בגוגל ילמד אותנו שמדובר במספר המזהה מדינה בעולם תקשורת המובייל, ובהתאם למקרה שלנו, מדובר במספר המזהה של ישראל.

אם נצא מנקודת הנחה שערכים אלה "מפלילים" אותנו, כל שנצטרך לעשות הוא להבין עבור אילו ערכים הבדיקה עוברת בהצלחה, נשלח אותם לשרת, וניצחנו!

חיפוש מהיר בגוגל מלמד אותנו ש-Shazam Lite נתמכת במספר מצומצם של מדינות, בין היתר בהודו. נחפש בזריזות את ה-country code של הודו, וכמו כן את ערך ה-mobileCountryCode. הם "in" ו-"404" בהתאמה (נוכל להשתמש ב-404, 405, 406 עבור mobileCountryCode).

נשתמש בפיצ'ר שימושי ב-mitmproxy (וכמו כן ב-proxies מוכרים נוספים) המאפשר לנו לערוך בקשות, ולשלוח אותן לשרת בשנית, באמצעות פיצ'ר זה נוכל לערוך את השדות החשודים ולראות את תגובת השרת.

כיצד נערוך את הבקשה? על ידי סימן העיפרון שנמצא בצד ימין של הבקשה: נשנה את ערכי "country" ו-"mobileCountryCode" בהתאם, ונשלח את הבקשה בשנית ע"י לחיצה על כפות ה"Replay" בצד שמאל למעלה, או לחליפין על ידי לחיצה על R. וואלה! אנחנו מקבלים 200 מהשרת! **ניצחנו!**

אבל רגע, אומנם הבנו מה הערכים שמפלילים אותנו והצלחנו לקבל תשובה חיובית על הבקשה שלנו, אך מדובר בבקשה לה ביצענו Replay ולא בבקשה עצמה הנשלחת ע"י האפליקציה. בשלב זה נחזור אל כלי עוצמתי שהכרנו לפני מספר דקות, הלוא היא Frida.

אחד הדברים המגניבים ש-Frida מאפשרת לנו לעשות, הוא היכולת לבצע Hook לפונקציות בפשטות ובמהירות, על ידי כתיבת סקריפטים פשוטים ב-Javascript (בדומה ל-Script ה-SSL Unpinning שבו השתמשנו). ביצוע Hook לפונקציה למעשה מאפשר לנו לשנות את המימוש של לחלוטין, לדוגמא במקרה שלנו,

פונקציה היוצרת JSON עם ערכים מסויימים - נוכל לשנות את הדרך בה הפונקציה יוצרת את ה-JSON הרלוונטי על ידי הכנסת ערכים לבחירתנו.

כאן אפשר למצוא מדריך מעולה לשימוש ב-Frida בעולמות ה-Android:  
<https://node-security.com/posts/android-hooking-in-frida>



בפועל, על מנת ליצור Hook לפונקציה, כל מה שנצטרך הוא לדעת מה שם ה-Class שמממש אותה, מה השם שלה בתוך אותו Class, ומה החתימה שלה (הפרמטרים וערך החזרה). אז איך נוכל למצוא את הפונקציה הזו? איך בכלל נוכל לקרוא את הקוד של האפליקציה? לצורך כך נכיר כלי נוסף, והוא jadx.

## Jadx

Jadx הינו כלי חינמי מבוסס קוד פתוח המסוגל לבצע די-קומפילציה לקבצי APK, ואף מספק ממשק UI לצפייה בקוד שעבר די-קומפילציה. במקרה שלנו, נשתמש ב-Jadx ע"מ לחלץ את קבצי ה-Java המרכיבים את האפליקציה, ונטען אותם ב-Android Studio על מנת לבחון אותם ולאתר את הפונקציה הרלוונטית.

את Jadx נוכל להוריד מה-Repository הרשמי ב-Github ולהתקין אותו ע"פ ההסבר שנמצא ב-Repository: <https://github.com/skylot/jadx>

לאחר שהתקנו jadx בהצלחה, נבצע די-קומפילציה ל-APK של האפליקציה כך:

```
jadx -e -deobf <path_to_apk>
```

כמה מילים על הדגלים בהם נשתמש:

“-e” - מייצא את הקבצים בצורה שבה כל Class נמצא בקובץ משלו, דבר המדמה בצורה טובה יותר את קוד המקור (מבחינת קבצים), ומאפשר לנו לבצע Import אל תוך Android Studio בתור פרוייקט.

“-deobf” - נבקש מ-Jadx לנסות לשחזר שמות של מחלקות ופונקציות שעברו אובפוסקציה (בפועל, jadx ייתן לפונקציות ומחלקות שמות קצת יותר "קריאים", דבר שעשוי להקל עלינו בקריאת הקוד)

נטען את הקוד ה-Decompiled ע"י File << Open Android Studio, כעת אנחנו ערוכים ומוכנים לריוורס האפליקציה.

## Static Analysis

כאשר אנו ניגשים למחקר סטטי של אפליקציה, בדגש על מצב בו אנו מחפשים מרכיב נקודתי וספציפי, או פונקציה במקרה שלנו, עולה השאלה - איך נמצא את הפונקציה? הרי מדובר בכל כך הרבה קוד! כדי שנוכל לדייק את עצמנו, או לפחות לצמצם את "משטח" החיפוש, נחשוב על Point Entry, כלומר, נקודה בקוד שיהיה לנו דיי קל למצוא, שקשורה באופן כזה או אחר לפונקציה שאנחנו רוצים למצוא.

לדוגמא במקרה שלנו, האפליקציה שולחת בקשה לשרת, בקשה זו מכילה ב-Body שלה JSON עם המידע אותו נרצה לשנות, כנראה שיש פונקציה שאחראית על בניית ה-JSON הזה, אולי נוכל לאתר אותה על ידי חיפוש מחרוזות שמופיעות באותו JSON, כמו לדוגמא "mobileCountryCode" (דוגמא נוספת היא לחפש את קטע הקוד שמרכיב את הבקשה הרלוונטית על ידי חיפוש ה-URL או אחד מה-Headers).

**נקודה חשובה לגבי מחלקות פונקציות ומשתנים שעברו אובפוסקציה:** בפועל, אחרי ש-jadx מבצע decompilation ל-APK, הוא ייתן שמות פשוטים וחסרי משמעות למחלקות פונקציות ומשתנים, בדרך כלל מדובר באותיות בודדות. כאשר נשתמש בדגל ה-debof, השמות יהפכו למעט יותר קריאים, שילוב של ספרות ואותיות, אך מאחר ושמות אלה עשויים להשתנות בין תוצאות דיקומפילציה שונות, נתבסס על השמות המקוריים, אותם ניתן לזהות ע"י שורת ההערה הנמצאת מעל לכל משתנה \ פונקציה \ מחלקה.

נלחץ ctrl+shift+f על מנת לפתוח את דיאלוג החיפוש, ונחפש את הסטרינג הרלוונטי. מעולה! נראה שיש רק תוצאה אחת! לחיצה כפולה על התוצאה תביא אותנו ל-Class בו ה-String הרלוונטי מופיע. נראה כי אותו String מזהה Attribute של מחלקה בשם:

com.shazam.android.lite.e.a.b.a.a.c0686c

```
/* renamed from: a */  
public final String f1654a;
```

בנוסף לאותו שדה, נוכל לראות שדות נוספים עם השמות "model", "mobileNetworkCode", "os" נראה קצת מוכר לא? מדובר בשדות שמופיעים ב-JSON שאנחנו מחפשים!

עושה רושם שמחלקה זו מייצרת אובייקט המכיל את הערכים שמעניינים אותנו, אם נמצא את החלק בקוד שמייצר את האובייקט הרלוונטי, ומזין אותו בערך "mobileCountryCode", נוכל למצוא את מקור הערך הרלוונטי!

```
public C0685b(String str, String str2, String str3, String str4, C0684a c0684a, C0686c c0686c) {  
    this.f1654a = str;  
    this.f1655b = str2;  
    this.f1656c = str3;  
    this.f1657d = str4;  
    this.f1658e = c0684a;  
    this.f1659f = c0686c;  
}
```



```

package com.shazam.android.lite.p054e.p055a.p057b.p058a.p059a;

import com.p017b.p018a.p019a.InterfaceC0263c;
/* renamed from: com.shazam.android.lite.e.a.b.a.a.b */
/* loaded from: classes.dex */
2 usages
public final class C0685b {
    1 usage
    @InterfaceC0263c(m566a = "inid")

    /* renamed from: a */
    public final String f1654a;
    1 usage
    @InterfaceC0263c(m566a = "platform")

    /* renamed from: b */
    public final String f1655b;
    1 usage
    @InterfaceC0263c(m566a = "language")

    /* renamed from: c */
    public final String f1656c;
    1 usage
    @InterfaceC0263c(m566a = "country")

    /* renamed from: d */
    public final String f1657d;
    1 usage
    @InterfaceC0263c(m566a = "application")

    /* renamed from: e */
    public final C0684a f1658e;
    1 usage
    @InterfaceC0263c(m566a = "device")

```

נשים לב כי Attribute זה מוגדר על ידי ה-Constructor, וליתר דיוק, מדובר בפרמטר השני. נלחץ ctrl+left click על ה-Constructor של המחלקה, ואוטמטית נקפוץ לשימוש היחיד שיש בו בקוד (במידה והיה נעשה בו שימוש יותר מפעם אחת, היינו מקבלים את רשימת כל המופעים).

נשים לב כי נגיע לשורת דיי ארוכה, המכילה שרשור של מספר פונקציות ומשתנים בעלי שם לא אינפורמטי (גם כאן מדובר באובפוסקפציה), אך לנו זה לא משנה, נוכל "להתעלם מהרעש" ולהתרכז באובייקט שמעניין אותנו - C0686c:

```

(), new C0686c(this.f14881.mo176e(), this.f14881.mo174g(), this.f14881.mo173h(), new C0687d(str: "Android", String.valueOf(this.f14881.mo175f()))));

```

כפי שראינו, הערך שיקבל ה-Attribute "mobileCountryCode" הוא השני ב-Constructor, אותו ערך המגיע מפונקציה בשם "mo174g" מאובייקט מסוג "com.shazam.android.lite.e.a.l InterfaceC0720l /". אם ניכנס לקובץ ה-Class של אותו אובייקט, נראה כי הקובץ מכיר חתימות לפונקציות ללא המימוש שלהן וזאת מאחר ומדובר בממשק שיש לממש.



אז איך נמצא את ה-Class שמממש את הממשק הרלוונטי? נחזור לדיאלוג החיפוש (ctrl+shift+f) ונחפש את הטקסט הבא:

“implements InterfaceC0720I”

כך נוכל למצוא את כל המחלקות המממשות את אותו ממשק, ולשמחתנו, יש רק אחת כזו! “C067b \ com.shazam.android.lite.d.b” נוכל לראות ב-Constructor של המחלקה שימוש ב-API מעניין מאד, נראה שאנחנו קרובים לפתרון!

```
1 usage
public C0607b(Application application) {
    TelephonyManager telephonyManager = (TelephonyManager) application.getSystemService( name: "phone");
    if (telephonyManager.getSimState() == 5) {
        this.f1525b = telephonyManager.getSimOperator();
        this.f1527d = telephonyManager.getSimCountryIso();
    } else {
        this.f1525b = null;
        this.f1527d = null;
    }
    this.f1526c = Locale.getDefault();
}
```

נריך חיפוש זריז על TelephonyManager ונמצא את הדף האינפורמטיבי [הבא](#).

נקרא על הפונקציות "getSimOperator" ו"getSimCountryIso" ונראה שהן מחזירות בדיוק את מה שאנחנו מחפשים. נוכל לראות כי הפונקציה "b" מחזירה את התוצאה של "getSimCountryIso", כלומר את הערך "country" ב-JSON הרלוונטי, והפונקציה "g" מחזירה את הערך "mobileCountryCode" שמגיע כחלק מהקריאה ל-getSimOperator.

נבצע בדיקה ונכתוב סקריפט Frida פשוט שמבצע Hook ל-2 הפונקציות הרלוונטיות, ומדפיס את ערך החזרה שלהן:

```
Java.perform(function()
{
    console.log("Script loaded")
    var theClass = Java.use("com.shazam.android.lite.d.b")

    theClass.b.implementation = function()
    {
        console.log("b ret - " + this.b())
        return this.b()
    }

    theClass.g.implementation = function()
    {
        console.log("g ret - " + this.g())
        return this.g()
    }
})
```

אז מה הסקריפט עושה? (האובייקט הנקרא Java בו אנו משתמשים כדי לקרוא לפונקציות כמו perform או use הוא למעשה reference לסביבת הריצה ה-Java-ית של האפליקציה הרלוונטית), תוכן הסקריפט שלנו נמצא בתוך פונקציית perform, שהיא חלק חשוב מה-Javascript API של Frida, פונקציה זו למעשה יוצרת את Context הריצה שלנו, בפועל היא יוצרת Thread בתהליך האפליקציה הרלוונטית, המיועד להרצת הסקריפט שיצרנו.

נתחיל בהדפסה פשוטה שתודיע לנו כי הסקריפט נטען בהצלחה, ונתחיל ביצירת משתנה שייצג את המחלקה הרלוונטית על ידי שימוש בפונקציה use, המאפשרת לנו "לגשת" ליישויות בעולם ה-Java, וכמו במקרה שלנו, יצירת Reference למחלקה כרצוננו.

כעת, האובייקט theClass הוא למעשה Reference למחלקה הרלוונטית, ונוכל להשתמש בו כדי לשנות מימוש של פונקציות השייכות ל-Class.

פעולה זו מיושמת על ידי "פנייה" לפונקציה הרלוונטית (נתייחס לאובייקט theClass כאל אובייקט רגיל לחלוטין, המכיל Attributes שמייצגים פונקציות ומשתנים) במקרה שלנו, b ו-g (גם אליהן נתייחס כאובייקטים) כך שנדרוס את ערך ה-Implementation שלהן (Attribute המייצג את המימוש של הפונקציה) ע"י מימוש פונקציית JavaScript אנונימית אשר תהווה את המימוש החדש.

נדפיס את ערך החזרה של המימוש המקורי של הפונקציה ע"י קריאה לפונקציה הרלוונטית מתוך האובייקט this, אשר כמו ב-Java, מהווה רפרנס לאובייקט הנוכחי, ולבסוף, נחזיר את ערך החזרה של המימוש המקורי של הפונקציה (כמובן שיוכלנו להכניס את ערך החזרה למשתנה ולוותר על 2 קריאות, אך המימוש הרלוונטי מאפשר לנו לגעת שוב במשמעות של האובייקט this ב-Context של Frida)

נשמור את הסקריפט ונריץ אותו (כפי שראינו במהלך המאמר), ונראה את ה-Output הבא:

```

zam.android.lite
┌───┐
│ ( ) │
│ > │
│ /_/_ │
│ . . . │
│ . . . │
│ . . . │
│ . . . │
│ . . . │
│ . . . │
│ . . . │
│ . . . │
│ . . . │
└───┘
Frida 16.0.7 - A world-class dynamic instrumentation toolkit

Commands:
  help           -> Displays the help system
  object?       -> Display information about 'object'
  exit/quit     -> Exit

More info at https://frida.re/docs/home/

Connected to SM G980F (id=RF8N21R4EBP)
Spawned `com.shazam.android.lite`. Resuming main thread!
[SM G980F::com.shazam.android.lite ]-> Script loaded
g ret - 425
b ret - il

```



מעולה! נראה שהפונקציות שמצאנו מחזירות בדיוק את הערכים שחיפשנו! כל מה שנשאר לנו לעשות הוא לשנות את המימוש של ה-Hook שכתבנו, כך שבמקום להדפיס את ערך החזרה של המימוש המקורי, נחזיר את הערכים שמצאנו.

נערוך את הסקריפט בהתאם:

```
Java.perform(function()  
{  
    console.log("Script loaded")  
    var theClass = Java.use("com.shazam.android.lite.d.b")  
  
    theClass.b.implementation = function()  
    {  
        console.log("Return fake value for b")  
        return "in"  
    }  
  
    theClass.g.implementation = function()  
    {  
        console.log("Return fake value for g")  
        return "404"  
    }  
})
```

צ'אקה! עקפנו את המנגנון וכעת נוכל להשתמש ב-Shazam Lite על אף ההגבלות ©

## סיכום

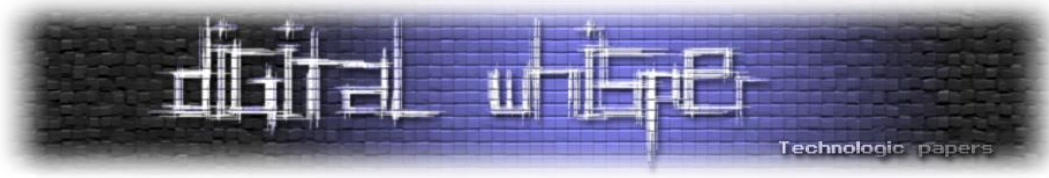
במאמר זה עקפנו מנגנון באפליקציית Android מוכרת תוך שימוש במספר כלים בעלי עוצמה רבה, התחלנו מלהתגבר על מנגנונים המקשים עלינו בביצוע מחקר עד לכדי מחקר סטאטי של קוד האפליקציה וזיהוי של הפונקציה הנקודתית אשר מהווה מקור שאלת המחקר שלנו.

ישנם נושאים רבים ורלוונטים בהם לא נגענו במאמר זה, אך הכלים שהכרנו מהווים Toolbox משמעותי ולגמרי מספק כדי להמשיך ולצלול אל עולם המחקר בסביבת Android.

מקווה שנהנתם!

## על המחבר

עידן שכטר, בן 25, אוהב בעלי חיים ואת הים, חוקר בקבוצת NSO.



---

## דברי סיכום

---

בזאת אנחנו סוגרים את הגליון ה-154 של Digital Whisper, אנו מאוד מקווים כי נהנתם מהגליון והכי חשוב: למדתם ממנו. כמו בגליונות הקודמים, גם הפעם הושקעו הרבה מחשבה, יצירתיות, עבודה קשה ושעות שינה רבות כדי להביא לכם את הגליון.

ניתן לשלוח כתבות וכל פניה אחרת דרך עמוד "צור קשר" באתר שלנו, או לשלוח אותן לדואר האלקטרוני שלנו, בכתובת [editor@digitalwhisper.co.il](mailto:editor@digitalwhisper.co.il).

על מנת לקרוא גליונות נוספים, ליצור עימנו קשר ולהצטרף לקהילה שלנו, אנא בקרו באתר המגזין:

[www.DigitalWhisper.co.il](http://www.DigitalWhisper.co.il)

*"Silk'n' Bout a r3vo7u7ion 5ounds like a whi5p3r"*

הגליון הבא ככל הנראה לא ייצא בסוף ספטמבר, אלא בסוף אוקטובר.

אפיק קסטיאל,

31.08.2023