digital whisper
Technologic Papers

Federated Identity

מאת ליאור בר-און

הקדמה

האינטרנט פורח - וזה דבר נהדר. במקום אתרים סטטיים יש לנו עכשיו "אפליקציות" שעושות דברים נפלאים. לעתים הולכות וקרבות אפליקציות משתפות פעולה זו-עם-זו בכדי לעשות משהו נפלא חדש.
בכדי לעשות את כל הדברים הנפלאים הללו, בצורה שתתאים לנו, האפליקציות שומרות פרטים עלינו. פרטים אישיים. פרטים חשובים.

בכדי להגן על הפרטים האישיים של המשתמשים, העולם משתמש במנגנוני-אבטחה, שאחד מיסודותיהם הוא מנגנון הAuthentication (זיהוי). כשמדברים על האינטרנט ואפליקציות המדברות זו-עם-זו, אנו מדברים לרוב על Authentication בעזרת Federated Identity (= זהות בקבוצה מבוזרת).

במאמר זה נסקור את עקרונות הבסיס של פרוטוקולי Federated Identity ונספק כמה מילים על הפרוטוקולים הנפוצים.

זהות

אם אנו מדברים על Federated Identity (בפוסט זה אשתמש מעתה בקיצור FI), אולי כדאי להתחיל בכמה מילים על המושג "זהות".

אני מניח שכולם יודעים מהי "זהות" של אדם ביום-יום (כל עוד לא נגרר למישור הפילוסופי): המסמך שמתאר את הזהות שלנו יכול להיות ת"ז, דרכון או רשומה במשרד הפנים, אבל גם נתונים פיזיים: מראה, טביעת אצבע או אופן ההליכה (מסתבר שהוא דיי ייחודי) - כל אלו מתארים את הזהות שלנו.

מרכיב חשוב בזיהוי "יום-יומית" היא שהיא מבוססת על אמון (Trust): מישהו מאמין למדינת ישראל וסומך על תעודת הזהות שהיא הנפיקה (על אף שהיא עלולה להיות שקרית או מזויפת). אתם יכולים להאמין למישהו (למשל חבר לעבודה) שאומר "אני מכיר אותו, זהו משה!". האמון יפחת אם החבר לא מכיר היטב את האיש השלישי, או אם אתם לא מכירים היטב את החבר. למשל: אדם זר שניגש ברחוב ומספר לכם

שאיש שלישי הוא משה, הוא לא מצב מעורר אמון (אפשר לפתוח ולומר: "מה בכלל רוצה ממני הבחור הזה?!")

אמון יכול להיות טרנזיטיבי: החבר ראה ת"ז של אדם שלישי. אתם סומכים על החבר שסומך על תעודת הזהות שראה.

זהות דיגיטלית, או זהות אינטרנט היא דומה - אך קצת שונה:

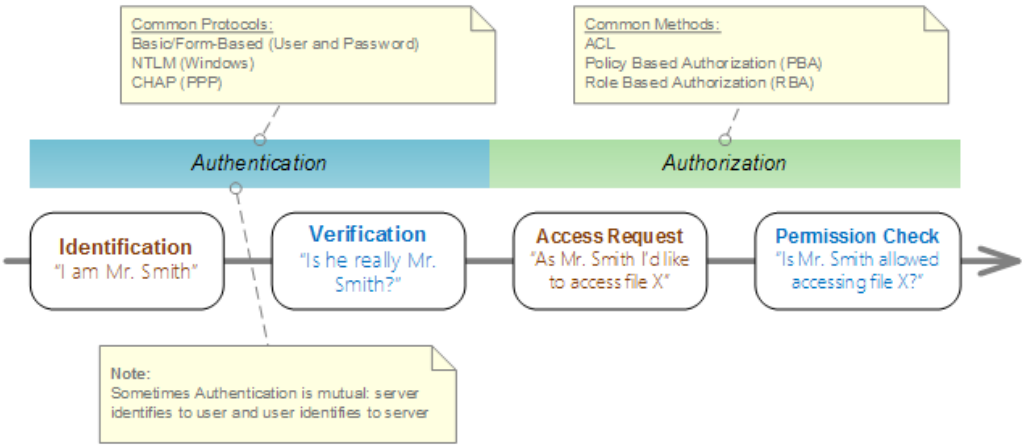
- בעולם הדיגיטלי אין באופן טבעי מאפיינים ייחודיים (מראה, קול, או אפילו דרך הליכה) המזהים אדם שלישי - כולם נראים "אותו הדבר" מלבד כמה פרטים שהצהירו על עצמם.
- מעניין לציין שיש עיקרון שנקרא "risk-based authentication" בו מאמתים זהות של משתמש (ליתר דיוק: מחשבים סיכון לגניבת זהות) ע"י איסוף ופענוח "ההתנהגות הדיגיטלית" הטיפוסית של המשתמש, למשל: מספר השניות שהוא מבלה בכל דף באתר וסדר הדפים שהוא ניגש אליהם. אם תתחברו לאתר הבנק ותנהגו בצורה שונה מאוד מבד"כ - יש סיכוי שתחסמו ותתבקשו לגשת לסניף להוציא ססמה חדשה.
- מיקום גאוגרפי הוא קל ל"זיוף": כיצד אני יודע שהבחורה הנחמדה מאשדוד שמופיעה בפייסבוק היא לא האקר רוסי שנמצא 2000 קילומטר משם?
- קל למדי לפורץ "לאמץ" מספר זהויות דיגיטליות, אפילו באותו האתר - דבר הרבה יותר מסובך בעולם הפיסי.
- כמות הישויות (חברות / אפליקציות) שאוספות עלינו מידע הוא רב יותר, והמידע מפורט יותר. קשה להאמין שביה"ס שלמדתם בו במשך שנים שמר מידע שווה ערך למה שאתר אינטרנט שומר עליכם בתוך מספר ימים של שימוש. מספיק שרק אתר אחד ייפרץ - בכדי שפרטים אישיים כלשהם שלכם יהיו נגישים לאישיות לא רצויה.
- אתרי אינטרנט יכולים לייצר בקלות יחסית חזות אמינה, או לפחות אנו נוהגים בפחות זהירות כלפיהם: לכמה אתרים נתתם את כתובת הדוא"ל שלכם והשתמשתם שם באותה הסיסמה כמו חשבון הדוא"ל? לכמה בתי עסק נתתם את הכתובת שלכם בבית, ומפתח?

עקרונות ה-FI אינם תקן או דרך-יחידה לבצע את הדברים, אולם אם נתבונן במנגנונים הנפוצים:

- (Kerberos)
- SAML 2.0
- OAuth
- OpenID
- Claims-Based Authentication

נראה שיש בניהם המון דמיון. במשך שנים לארגונים ומערכות שונות היו מימושים פרטיים לאותם עקרונות. כל זה השתנה בשנות האלפיים שאפליקציות החלו יותר ויותר לדבר זו עם זו. עברו עוד מספר שנים עד שהארגונים הגדולים הצליחו להסכים על תקן (תקנים) אחידים ולהתיישר לפיהם. שלושת התקנים החשובים (SAML, OAuth ו-OpenID) מציגים שוני עקרוני בפונקציונלית שלהם שמצדיק קיום 3 פרוטוקולים שונים.

Kerberos ו-CBA הם פתרונות מוצלחים, שנפוצים כיום כמעט ורק בעולם של מייקרוסופט. מכיוון שההגמוניה של מייקרוסופט בסביבת המחשוב נפגעה מאוד במעבר לענן - ניתן להתייחס כיום ל-2 תקנים אלו כתקנים בעלי חשיבות משנית.



[תהליך אפשור גישה למשאב. אנו נתמקד בפוסט זה בשלב ה-Verification]

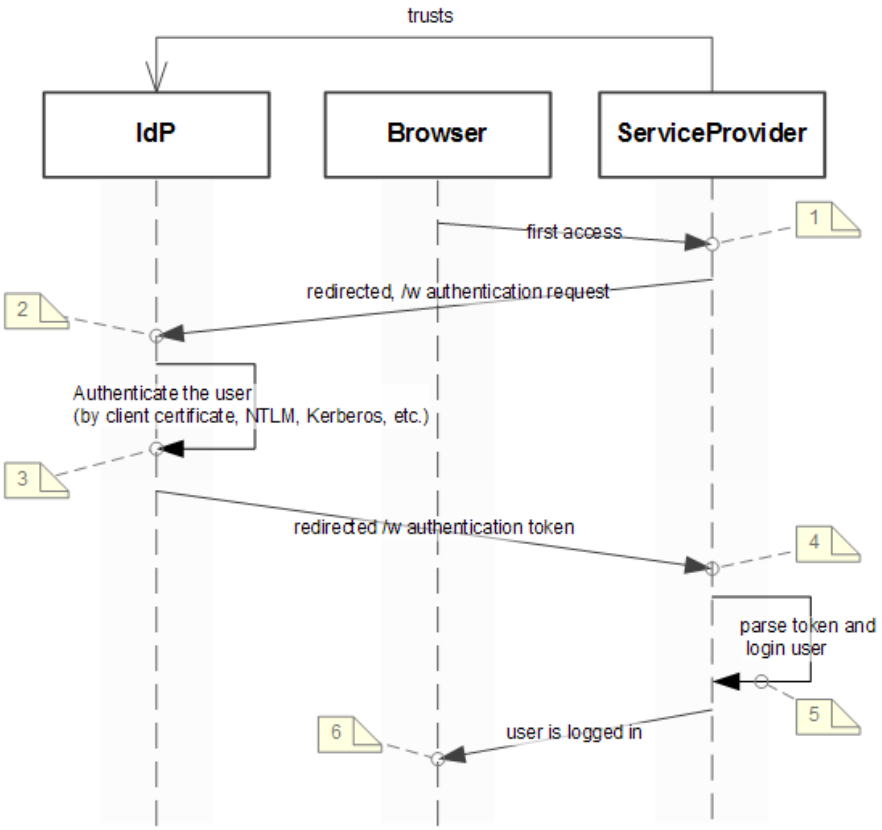
ניתן לקרוא עוד על שלב ה-Authorization בפוסט קודם בנושא.

תהליך ה-FI מנצל את העובדה שאימות (Authentication) וניהול הרשאות (Authorization) הם, מאז-ומעולם וע"פ תכנון, שלבים נפרדים בתהליך הגישה למשאב - בכדי להפוך את תהליך האימות לתהליך מבוזר שמתרחש בכלל על שרת אחר.

בשפה של FI מגדירים את המונחים הבאים:

- **Service Provider** - את הישות נותנת את השירות, למשל שרת פייסבוק.
- **Identity Provider** (בקיצור: IdP) - את הישות שמאמתת את זהות המשתמש (שלב Verification). ייתכן וה-IdP הוא לא המערכת בה מנהלים את המשתמשים (קרי LDAP / Active Directory) אלא שירות חיצוני שנמצא ביחסי אמון עם ה-User Repository.
- **Credential Store** (לחלופין Authentication Store) - היכן ששומרים את ההרשאות מה משתמש מורשה לעשות (שלב ה-Permission Check), לרוב זהו נותן השירות (למשל: פייסבוק), אך תאורטית זו יכולה להיות מערכת צד-שלישי שנותן השרות סומך עליה. פיזית, נתונים אלו נשמרים לרוב ב-Directory Service או בסיס-נתונים.

בדומה לעולם הפיסי, המפתח ל-FI הוא אמון (Trust) בין השרתים השונים. אמון זה נוצר פעמים רבות תוך כדי החלפת מפתחות-הצפנה בין 2 השרתים. ברגע שיש אמון, האמון הוא "מוחלט"[א]: אין וידוא נוסף מעברת לאימות זהות השרת עליו סומכים (על בסיס הצפנה) ווידוא שמה שנשלח מהשרת המרוחק לא שונה (modified) בדרך. להלן תרשים שמתאר בקירוב את האופן בו פרוטוקולים של FI עובדים:



תרשים A

הנחה מקדימה: ה-SP מכיר וסומך על ה-IdP (נעשה בעזרת קונפיגורציה ידנית).

1. **המשתמש** פותח דף בדפדפן ומנווט ל-SP.
2. ה-**SP** לא מזהה את המשתמש ומפנה אותו ל-IdP. אופן ההפניה שונה מפרוטוקול לפרוטוקול, וכן זהות ה-IdP.
3. ה-**IdP** מאמת את זהות המשתמש בדרכים שעומדות לפניו: הוא מכיר את המשתמש וצריך להיות מסוגל לאמת אותו.
4. ה-**IdP** מפנה את הדפדפן חזרה לדף המקור (פרטים הופיעו בבקשת האימות, בד"כ) ומייצר "מסמך" (token או ticket) המתאר פרטים שהוא יודע על המשתמש: id, מיקום, קבוצות שהוא שייך אליהן וכו'. פרטים אלו נקראים לרוב Assertions או Claims וה"מסמך" שמכיל אותם הוא מין וריאציה דיגיטלית של דרכון או ת"ז.

   ה"מסמך" נחתם ב-חתימה דיגיטלית כדי לוודא שצד שלישי לא יוכל לעשות בו שינויים.
5. ה-**SP** מקבל את המסמך - הוא מאמת, לרוב בעזרת החתימה הדיגיטלית, את זהות ה-IdP ואת שלמות/תקינות (integrity) המסמך.
6. במידה והמסמך נמצא תקין, הוא מבצע log in **למשתמש** ע"פ הפרטים שבמסמך, כלומר: בד"כ יוצר session שמתאר את המשתמש.

בסיכום מהיר ניתן לציין ל-FI את היתרונות והחסרונות הבאים:

<u>יתרונות:</u>

1. **משתמש**: כאשר משתמשים באותה סיסמה על ריבוי מערכות, ברגע שמערכת אחת נפרצת הפורץ יכול לנסות את הסיסמה ב-100 האתרים הנפוצים - אולי יתמזל מזלו. בעזרת FI אין צורך לנהל מספר-רב של ססמאות: אם מערכת (Service Provider) נפרצה - אין עליה את הסיסמה שלי.
2. **משתמש**: חוויית משתמש טובה. מעבר לכמה שניות המתנה בזמן ה-login, המשתמש לא מודע ש FI היה בכלל מעורב.
3. **מפתח ה-SP**: לא צריך להתעסק עם הנושא המורכב שקרוי Authentication. אנשי שיווק היו כבר ממציאים: "Authentication as a Service".
4. **מפתח, Admin ומשתמש**: היכולת לספק SSO (Single Sign-On) בצורה אלגנטית (למשל: פרוטוקול SAML 2.0), הרבה יותר אלגנטיים ממשפחה נפוצה אחרת של פתרונות SSO שנקראת " Credential Storage" בה שרתים / מכונת המשתמש שומרת שם וסיסמה לשרתים השונים. SSO הוא טוב כי:
   - המשתמש - לא צריך לזכור הרבה ססמאות / לבצע Login שוב כאשר הוא מופנה למערכת אחרת.
   - ה-Administrator - לא צריך לנהל מנגנוני Authentication כפולים.
   - המפתח (של SP) - חוסך לעצמו התעסקות עם Authentication. שהמפתח של ה-IdP - יעשה את זה!

5. **מפתח SP**: מקבל אינטגרציה קלה בין מערכות, שכעת רק צריכות להסכים על פרוטוקול ה-FI.
6. **מפתח, Admin**: מנגנוני FI לרוב גמישים למדי, ומאפשרים ל-IdP לספק כל סט של נתונים שהאפליקציה דורשת, למשל: מקום גאוגרפי של המשתמש, שפה מועדפת וכו'. זהו תחליף חלקי לשמירת מידע personalized על המשתמש, ויותר מזה - ניהול כפול שלו במספר מערכות (אני רוצה לכוון את השפה עברית במערכת אחת ולא בעשרה).

חסרונות:

1. החשש שפריצה ל-IdP תספק לפורץ גישה לכל האפליקציות של המשתמש/ים על ה-IdP, מה שנקרא "מפתח יחיד לממלכה". הסיכון קיים, אבל מכיוון שניהול של 20 סמאות מוביל לרוב לסיסמה אחת על 20 שרתים שפחות מאובטחים מה-IdP הממוצע - אין אלטרנטיבה טובה יותר לסיכון הזה, עדיין.
2. פתרון מורכב להקמה. למרות שהרעיון אינו חדש, יש מחסור בבסיס ידע / חומר כתוב [ב] / מומחים בתחום ה-FI, במיוחד כאשר מדובר ב-deployments שאינם בסיסיים. לאחרונה צצים פתרונות של "IdP as a Service" (לדוגמה Azure ACS) - לא אתפלא לגלות שהם מצליחים לפשט רבות את מלאכת ההגדרה.
3. אין סטנדרט יחיד ל-FI (בעצם יש 3-4 נפוצים), מה שמחייב מערכות לתמוך בכמה תקנים / לא לתמוך ב-FI עבור כל המשתמשים.

הפרוטוקולים

כמה מילים על ההבדלים בין הפרוטוקולים הנפוצים:

OpenID

מקצר / מפשט את תהליך ה-Trust בין ה-SP ל-IdP. בעזרת OpenID, אתר (SP) יכול לסמוך על IdP מבלי שה IdP יכיר אותו. כלומר: אין צורך בהחלפת מפתחות / certificates. מתאים לאתרי אינטרנט שפתוחים לכולם, ושבעיקר רוצים לזהות את המשתמש מבלי להכביד עליו. לרוב חיבור של OpenID נראה למשתמש הקצה ככפתור "התחבר באמצעות... <שם של IdP>"

IdPs של OpenID כוללים את: Facebook, Google, Yahoo ועוד.

OpenId מאפשר שתי דרכים שונות לבצע Authentication:

1. המשתמש נרשם ל-IdP ומקבל OpenId. כשהוא מתחבר ל-SP עליו להזין את ה-OpenId שבתוכו מקודד URL ל-IdP וכך ה-SP יודע להיכן להפנות את הדפדפן לצורך Authentication (שלב 2 בתרשים A).
2. (כנראה יותר נפוצה) כשהמשתמש ניגש ל-SP, ניתן לו ב-UI מבחר של IdP והוא בוחר אחד מבניהם אליו ה-SP יפנה את הדפדפן לביצוע תהליך ה-Authentication.

OAuth

הייחוד ב-OAuth הוא שהוא כולל גם Authorization, Authorization שעושה משתמש הקצה לאתר מסוים - לגשת לפרטים שלו (קריאה ו/או כתיבה). נגדיר את "אתר המקור", כאתר שמנהל פרטים אישיים של המשתמש ואתר ה-Service Provider כנותן שירות מסוים נוסף. התצוגה למשתמש הקצה תהיה הפנייה לאתר המקור ושאלה: "אתר SP מבקש לגשת ל... <נתונים/פעולות על נתונים> שלך, האם אתה מאשר?" האישור הוא כן/לא, לעתים עם יכולת לאשר רק חלק מהפעולות. לעתים השאלה מופיעה בתוך iFrame

(יכולת שנחסמה בגרסה 2.0 של הפרוטוקול) בכדי לא לגרום למשתמש לאבד אוריינטציה (אבל מאפשר התקפות clickjacking).

כשהמשתמש נותן אישור ל-SP לגשת למידע, ה-token שמותיר את הגישה מקודד בפירוש את שם ה-SP כך שאם ה-token נחטף - אתר אחר לא יוכל לעשות בו שימוש. בנוסף, OAuth token ניתן לזמן מוגבל ולאחר שיפוג - יהיה צריך האתר לבקש את אישור המשתמש פעם נוספת.

SAML

פרוטוקול SAML (בעצם SAML 2.0, אף אחד לא משתמש בגרסה 1 בימנו...) הוא פרוטוקול FI "קלאסי". על ה-IdP וה SP לייצר trust ע"י החלפת מפתחות ומשתמש בעיקר תסריטים של Enterprise בהם חשוב להגביל גישה ל-SP ממשתמשים לא רצויים. SAML 2.0 גם תומך ב-SSO. SAML, כדרך אגב, מתבסס על XML/SOAP (טכנולוגיות כמעט "מוקצות" בימנו) כבסיס.

על המחבר

ליאור בר-און עובד כארכיטקט תוכנה (בכיר) בחברת SAP ומתמחה בתחומים של web ו-middleware ב-Enterprise.

בנוסף, ליאור כותב בלוג עברי על הנדסת / ארכיטקטורת תוכנה:
http://www.softwarearchiblog.com.