Digital Whisper Technologic Papers

Windows Authorization & Security Internals 101

מאת שקד אילן (shackrack)

הקדמה

כיצד מנגנון ההרשאות עובד ב-Windows? מה קורה מאחורי הקלעים כשביצענו Logon? מה מתרחש מאחורי הקלעים כאשר אנחנו מנסים למחוק קובץ ומקבלים Access Denied? איך אפשר להעלות הרשאות לתהליך או (בצורה פחות לגיטימית ונחמדה) לגנוב זהות של משתמש אחר ולהשתלט על דומיין באמצעות גניבת טוקנים (Impersonation)? מה זה אומר להיות Elevated? או שה-Integrity Level של התהליך שלי הוא Medium? אילו מנגנוני בידוד אבטחתי והכלה של תהליכים קיימים ואיך הם עובדים? למשל, כיצד Microsoft מונעים מחולשה בדפדפן להתפרץ לכל מערכת ההפעלה?

מהו ה-Security Descriptor ומה ההבדל בין SACL ל-DACL שחיים בתוכו? (וכמובן למה צריכים אותם בכלל?), מי הם משתמשי ה-Builtin שבאים בכל התקנת Windows ולמה הם שם? על הכל אנסה לענות ולעשות סדר במאמר הבא. כל מה שחשוב לדעת על מאחורי הקלעים של מנגנוני האבטחה, אימות וזהות ב-Windows באמצעות דוגמאות לשימושים מחיי היומיום.

עד סוף המאמר תכירו היטב את מונחי ה-Windows Internals הבאים: ACL, SACL, DACL, ACE, SRM, IL SID, MIC, UAC, Integrity Level, Access Rights, Access Token, Full/Primary/Filtered/Restricted Token, Impersonation Token, Impersonation Level, Builtin Users ועוד. בהחלט רשימה מכובדת. אז שנתחיל?

הערה: המאמר מבוסס על Windows מגרסאת 10 ו-11 (בשונה מהמאמר הקודם) ומיועד גם לקוראים בעלי ידע בסיסי ב-Windows (אין צורך בידע low level ו\או תכנות Win32 קודם) וכאלו המעוניינים לצלול לעומק הדברים.

Builtin Users

בכל התקנה של Windows מגיעים חבילה של משתמשים, כאשר לכל אחד יש ייעוד משלו, נכסה את העיקריים שבהם:

- **Administrator**: משתמש אדמין מקומי בעל הרשאות גבוהות, disabled כברירת מחדל והפורטה שלו זה שאינו נכלל תחת הפוליסה של **UAC**: משמע שאם נאפשר ונעבוד דרכו לא נצטרך לעבוד דרך UAC כאשר נרצה להריץ תהליכים Elevated - מה שאומר שכל פקודה שנריץ (גם בלי להשתמש באופציה של "Run as administrator") תרוץ בהרשאות גבוהות (ממש כמו ב-Windows XP). את המשתמש הזה לא ניתן להסיר.
  - מאחורי הקלעים מה שקורה זה שהוא מקבל full access token עם high integrity level (ולא medium כמו בהתחברות רגילה)
  - הוא בעל Relative Identifier של 500, לדוגמא:

```
SID: S-1-5-21-DomainID-500
```

  - כדי לאפשר אותו נערוך את הערך רג'יסטרי הבא ל-"1":

```
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Policies\System\FilterAdministratorToken
```

- **Guest**: משתמש בעל הרשאות נמוכות, disabled כברירת מחדל

- **DefaultAccount**: חדש החל מ-Windows 10, משתמש פנימי בשימוש ע"י מ"ה, בעל הרשאות נמוכות.

- **WDAGUtilityAccount**: חדש החל מ-Windows 10, בשימוש ע"י Windows Defender Application Guard ונועד לעזור לדיפנדר להגן מפני malwares וכד'

- **SYSTEM**: המשתמש הכי חזק במערכת, בעל הרשאות יותר גבוהות מ-Administrator ובשימוש ע"י מ"ה לשירותים שלה בעיקר.

- **LocalService** & **NetworkService**: משתמשים חזקים אך בעלי הרשאות מוגבלות (לדוגמא ל-local יש גישה רשתית מוגבלת), נקראים גם service accounts והמ"ה משתמשת בהן על מנת להריץ Windows Services.

  על מנת לא לתת לכולם לרוץ תחת ההרשאות הכי חזקות של המשתמש SYSTEM ולרדד את משטח התקיפה, שירותים מסויימים רצים תחת שני החבר'ה האלו

- קבוצת **NT AUTHORITY\Authenticated Users**: כל המשתמשים שעשו לוגין ועברו אותנטיקציה בהצלחה (למשל לא כוללת את המשתמש Guest)

- קבוצת **NT AUTHORITY\INTERACTIVE**: כל המשתמשים שכרגע מחוברים (RDP או מקומי)

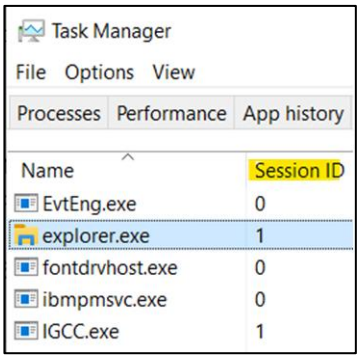
- קבוצת **NT AUTHORITY\SERVICE**: כוללת את כלל השירותים אשר רצים תחת המשתמשים LocalService ו-NetworkService
- קבוצת **NT AUTHORITY\SYSTEM**: מכילה את כל התהליכים שרצים עם הרשאות SYSTEM. בדומיין המשתמש NT Authority\System שקול בהרשאותיו ל-computer account עצמו.

כיצד המשתמשים האלו באים לידי ביטוי? כאשר מפתחים בונים את ה-Windows Service שלהם, הם יכולים להחליט תחת איזה משתמש הוא ירוץ בהתאם לצרכים שלם. ככל שיבחרו במשתמש המותאם ביותר (לדוגמא LocalSystem אם אין צורך בפעולות רשתיות) כך במידה ותוקף יצליח להריץ קוד בתוך התהליך שלהם הוא יהיה מוגבל.

דוגמא נוספת תהיה ה-web server המפורסם IIS, שרץ (או מריץ אתרים מסוימים) תחת המשתמש המוחלש IUSR/DefaultAppPool אשר מוגבל מאוד ביכולות שלו (בלי יכולת להריץ תהליכים Elevated, לגעת ברג'יסטרי, או להתקין תוכנות. בגדול מסתכם באפשרות להגיש קבצים מהתיקייה של האתר, בלי יכולת EXECUTE מעבר לכך וכו') כשגם פה הרעיון או שבמידה ותוקף הצליח להשחיל webshell הוא ישאר contained and isolated בסביבה של האתר הזה בלבד.

Windows Stations and Desktops

אולי כבר שמתם לב לשדה הזה בשם Session ID המשויך לכל תהליך ותהיתם מה הוא אומר:



הרעיון בפשטות: ניהול ויצירת בידוד אבטחתי בין משתמשים מחוברים ושירותים שונים במערכת. במידה ויש מספר משתמשים מחוברים לאותו מחשב נרצה שהם יהיו מופרדים לחלוטין נכון? עם GUI משלהם, clipboard שלא יהיה משותף ותהליכים משלהם שלא ישפיעו אחד על השני לדוגמא לא יוכלו לקרוא את הזיכרון אחד של השני) - בבסיס זה הרעיון.

נסביר: כאשר אנחנו עושים Interactive Logon ל-Windows (בין אם באמצעות RDP או פיזי) נוצר **Session** עבור ההתחברות שלנו, ה-Session-ים ממוספרים באופן עוקב, החל מ-0 כאשר **Session 0** משמש ל-Windows Services. לכל Session מוקצה זיכרון ייחודי ומאובטח משלו.

כל Session מכיל מספר **Window Stations**: אובייקטים מאובטחים, המשמשים בעיקר לצורך תחימת האובייקטים שהם מכילים תחתיהם לצורכי אבטחה. Window Stations יכולים להיות אינטראקטיביים ולא אינטראקטיביים (קרי: יש\אין GUI) והשם של כל Window Stations הוא ייחודי בתוך כל Session.

עבור כל Session, קיים Window Station מיוחד בשם **winsta0**, והוא ה-Window Stations האינטראקטיבי היחיד.

תחת ה-Window Stations ניתן להגדיר מספר **Desktops**: כל Desktop הוא אובייקט מאובטח בעל "משטח תצוגה" לוגי, עליו אפליקציות יכולות לרנדר אלמנטי UI בצורת חלונות, הוא בעצם האובייקט שמאפשר ל-GUI במ"ה להתקיים. לכל interactive windows station (כמו 0winsta) יש כברירת מחדל Desktop בשם Default.

תחת Window Station אחד יכולים להיות מספר Desktops בעלי אלמנטי UI אך רק אחד מהם יכול להיות מוצג בכל רגע נתון. תחת Winsta0 נוצרים באופן דיפולטי שלושה Desktops:

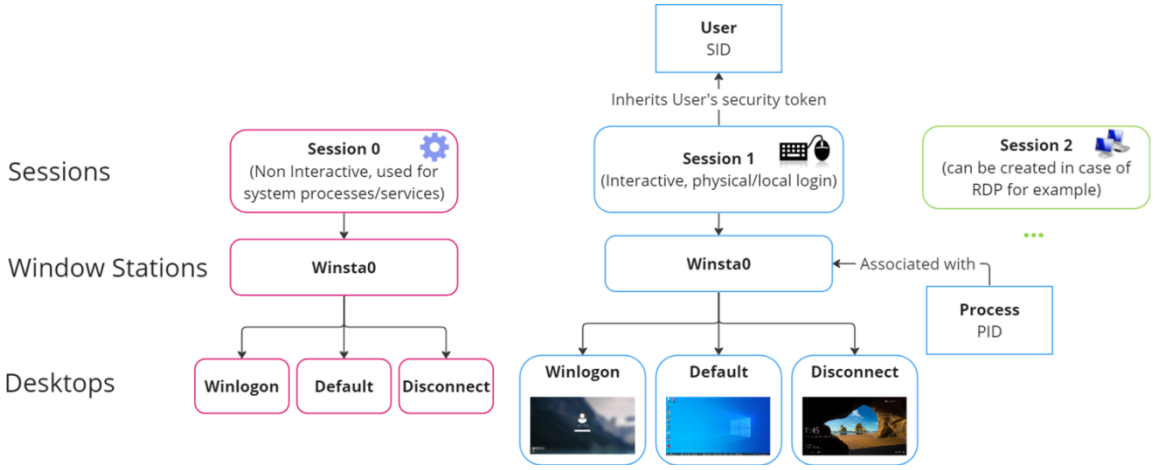
1. **Winlogon**: מכיל את מסך ההתחברות, ולאחר ההתחברות משמש כ-Desktop בו מוצגים חלונות דיאלוג של ה-UAC לבקשת העלאת הרשאות (בהנחה ש-Secure Desktop מופעל) ולכן לעיתים קוראים לו גם ה-Secure Desktop. זהו גם ה-Desktop אשר מוצג למשתמש כאשר לוחצים על Del+Alt+Ctrl. מה שמייחד את ה-Desktop הזה הוא שרק תהליכים שרצים כ-System יכולים לגשת אליו.

2. **Default**: ה-Desktop אשר מוצג לאחר ההתחברות. זהו ה-Desktop העיקרי אשר יוצג למשתמש, ובו יצוירו כל החלונות הקשורים ל-Session שלו (אלא אם כן המשתמש יבחר ליצור Desktops חדשים).

3. **Disconnect**: ה-Desktop בו מופיע ה-Saver-Screen.

כאמור, מעבר ל-Desktop-ים הללו, ניתן ליצור גם Desktops נוספים באופן תכנותי או בעזרת כלים מובנים ולא מובנים במערכת.

הכנתי אילוסטרציה קטנה שתעזור להסביר את ההיררכיה:



כעת בחזרה לשדה Session שראינו: אם תסתכלו שוב תוכלו להבחין שאכן כל התהליכים הרצים ב-Session 0 משוייכים לשירותי מערכת כמו lsass.exe ולחלופין יש לנו את explorer.exe (בכל ה-GUI) הרץ ב-Session 1 האינטראקטיבי:

Process	Session	Integrity	CPU	Private Bytes	Working Set
lsass.exe	0	System		11,552 K	17,716 K
fontdrvhost.exe	0	AppContainer		1,464 K	2,336 K
csrss.exe	1	System	< 0.01	3,636 K	4,548 K
winlogon.exe	1	System		3,140 K	7,148 K
fontdrvhost.exe	1	AppContainer		4,220 K	4,444 K
dwm.exe	1	System			
explorer.exe	1	Medium			
SecurityHealthSystray.exe	1	Medium			
rundll32.exe	1	Medium			
chrome.exe	1	Medium			
chrome.exe	1	Medium			
chrome.exe	1	Low			
chrome.exe	1	Medium			
chrome.exe	1	Untrusted			

lsass.exe:96 Properties

Image | Performance | Performance Graph | Security

User: NT AUTHORITY\SYSTEM
SID: S-1-5-18
Session: 0 Logon Session: 3e7
Virtualized: No Protected: No

טכנית ל-Session 0 כן יש מאפיינים אינטראקטיביים, אך אין לשירותים הרצים בו אפשרות לייצר חלונות (לרוב הדרך בה שירות יגיע ל-GUI של משתמש יקרה באמצעות taskbar notifications וכד').

למי שמעוניין בצלילה לקרנל על האובייקטים הנ"ל ממליץ על המאמר של יובל עטיה Kernel Exploitation Using GDI Objects בו נעזרתי לכתיבת פסקה זו.

Authentication Types וההבדל בין משתמש מקומי לדומייני

בעבר הרחוק מאוד יכולנו להתחבר רק פיזית (עם מקלדת ועכבר) למחשב Windows שלנו, כיום ישנן אפשרויות רבות שחשוב להכיר אותן ואת השוני ביניהם על מנת להבין את המשך המאמר, נסקור את המרכזיות:

רק לפני שנתחיל חשוב להבחין בין אותנטיקציה אינטראקטיבית לכזו שאינה:

- **Interactive Authentication**: מצריך התערבות של המשתמש, כמו לדוגמא הכנסת שם משתמש וסיסמה. מסך הלוגין המפורסם שניכנס דרכו כשמערכת ההפעלה עולה הוא דוגמא מצוינת לסוג זה של התחברות
- **Non-Interactive Authentication**: ללא התערבות של המשתמש, לרוב יבוצע ע"י Windows Services הרצים ברקע (ללא GUI) וצריכים לבצע אותנטיקציה ברקע ללא קבלת input מהמשתמש

ואלו ה-Types השונים:

- **Local Authentication**: בדיקה של המשתמש אל מול ה-database המקומי שנמצא על המחשב עצמו (Security Account Manager). אותנטיקציה מסוג זה אינה פונה למשאב רשתי כלשהוא על מנת לאמת את הזהות, אלא משתמשת רק במידע הנמצא על המחשב. סוג זה ישתמש בפרוטוקול NTLM ולרוב יבוצע כשנתחבר פיזית את המחשב שלנו וניכנס אליו. הפרטים של משתמשים מקומיים נשמרים במסד נתונים קטן בשם SAM במ"ה תחת הקובץ:

```
%SystemRoot%\System32\Config\SAM
```

- **Remote Authentication**: כאשר נתחבר מרחוק (לדוגמא ב-RDP) למחשב ברשת. כאן זה מתפצל לשני תרחישים:
  - כאשר נשתמש בפרטים של המשתמש המקומי: לדוגמא כאשר ברשת הביתית שלי (שאין בה דומיין) ארצה להתחבר למחשב של אשתי ב-RDP, אקיש את השם משתמש והסיסמה המקומיים שלה (ויתבצע ע"ג פרוטוקול NTLM לרוב)
  - כאשר נשתמש בפרטים של משתמש מהדומיין

- **Domain Authentication**: אימות של המשתמש אל מול ה-Active Directory אשר מנהל את המשתמשים, קבוצות והרשאות בדומיין (לרוב שרת ה-Domain Controller). לדוגמא כאשר אתחבר באמצעות המשתמש שלי מהדומיין, סוג זה ידרוש לרוב גישה רשתית ויתבצע ע"ג פרוטוקול Kerberos וכ-fallback ישתמש ב-NTLM.

- **Physical Authentication**: אמצעי פיזי כמו smart card \ ביומטרי.

הערה: חשוב להדגיש שבתרחישים מסוימים מספר שיטות יכולות להשתלב, לדוגמא כאשר משתמש מתחבר פיזית לעמדה שלו ומבצע זאת עם פרטי משתמש השייך לדומיין (ואז יש לנו שילוב של Domain & Interactive), או לחלופין כאשר משתמש מתחבר מרחוק (Remote Authentication) עם פרטי משתמש מהדומיין (Domain Authentication).

לעת הזו מה שכיסיתי יספיק לכם, אך למי שמעונין לאחר המאמר הזה לצלול לעומק הפרוטוקולים ושיטות האימות תרשמו לעצמכם לקרוא את המאמר ניהול סמאות וזהויות ברשתות מיקרוסופט של יהודה גרסטל, ועל Windows Lateral Movement from Scratch של יהונתן אלקבס.

כיצד השוני בין המשתמשים והשיטות אימות משמעותי לתקיפה? דוגמה לקריטיות בהבנת המשתמשים והאפשרויות השונות תהיה במידה והתפרצנו לשרת (שהוא חלק מדומיין המחובר ל-AD) באמצעות חולשה באיזה Web App וכעת אנחנו רצים עם ה-shell שלנו כאיזשהוא משתמש מקומי של אותו שירות שאנחנו רצים דרכו, זה אחלה ומגניב אבל זה לא מאפשר לנו להגיע לסביבה הדומיינית (לא לשרתי קבצים מעניינים ובטח לא ל-Domain Controller - בהמשך המאמר נבין למה) ובעצם אנחנו תקועים בשרת אליו פרצנו.

כתוקפים נחפש לראות איזה תהליכים רצים במערכת שהם תחת משתמש דומייני כלשהוא ע"מ שנוכל לגנוב את הזהות וההרשאות שלו: במידה ומשתמש מהדומיין כמו איש ה-IT התחבר ב-RDP או לחלופין ממש מחובר פיזית לשרת, יהיה לנו תהליכים רצים עם ה-Access Token שלו, כמו לדוגמא explorer.exe האהוב).

כעת רק חשוב להבין את הבסיס הזה ולמה הוא חשוב לתהליך התקיפה, בהמשך המאמר אסביר ונדגים איך אפשר לבצע את הגניבה הזו בפועל.

מאחורי הקלעים כשפותחים קובץ (Access Tokens, Access Rights & SRM)

אוקיי אוקיי, עשינו Logon לעמדה, ואנחנו נותנים בדאבל קליק על קובץ - מי מאשר לנו לקרוא את הקובץ ומה המנגנון קבלת החלטות שלו? בשביל לענות על זה נצטרך ללכת כמה שלבים אחורה ולהסביר מה קורה מאחורי הקלעים:

כשמשתמש מתחבר (מקומית-פיזית \ מרחוק \ משתמש דומייני) ל-Windows, נוצר בשבילו **Access Token**.

Access Token מכיל בין היתר את הנתונים הבאים:

- **Security Identifier**[1] (ובקיצור **SID**) של המשתמש (מזהה ייחודי למשתמש\קבוצה, לדוגמא *S-1-5-21-3623811015-3361044348-30300820-1013*). רוצים לדעת מה שלכם? הריצו ב-CMD את הפקודה:

```
wmic useraccount where name='%username%' get sid
```

כך יראה הפלט:

```
C:\Users\shaked ilan>wmic useraccount where name='%username%' get sid
SID
S-1-5-21-287014810-1449389800-2189324088-1001
```

- ה-SIDs של הקבוצות שהמשתמש נמצא בהן
- Logon SID שהוא מזהה אבטחתי ייחודי (Security Identifier) לאותו Session התחברות (כל עוד המשתמש מחובר למערכת):

```
C:\Users\shaked ilan>whoami /logonid
S-1-5-5-0-414220
```

[1] https://learn.microsoft.com/en-US/windows-server/identity/ad-ds/manage/understand-security-identifiers

- רשימה של כלל ההרשאות (**Privileges**) שיש למשתמש ולקבוצות בהן הוא חבר המתארות מה הוא יכול לעשות:

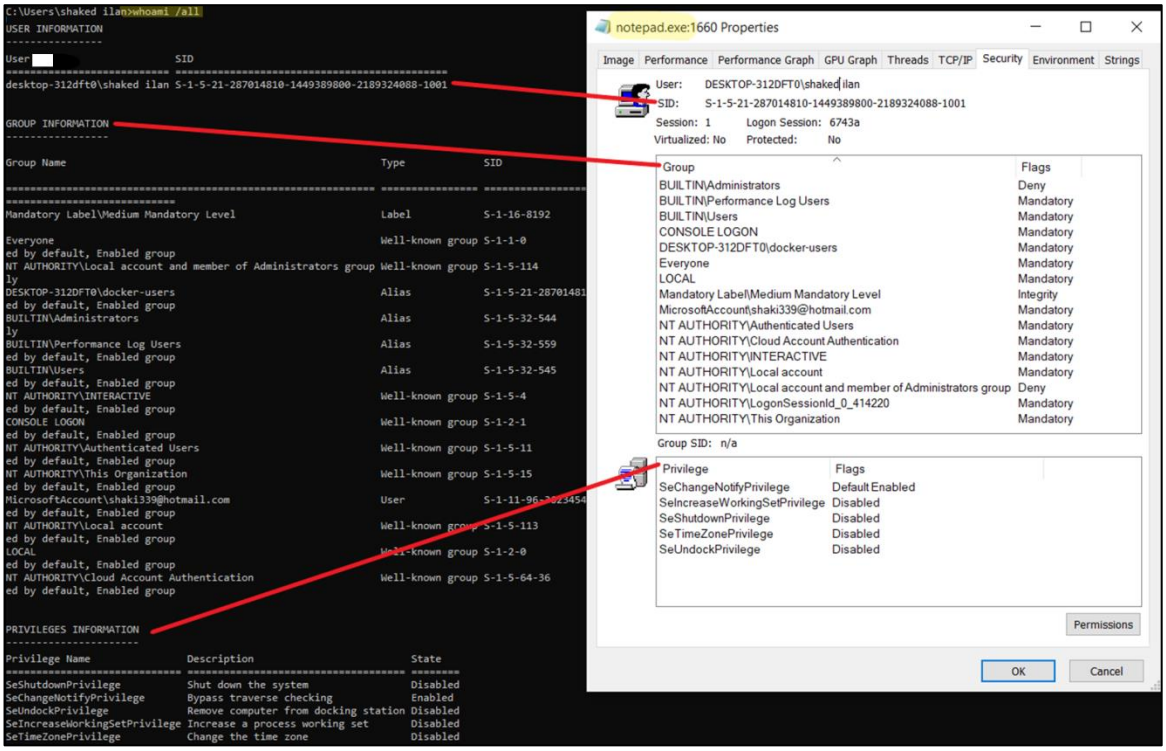
```
C:\Users\shaked ilan>whoami /priv

PRIVILEGES INFORMATION
----------------------

Privilege Name                Description                          State
============================= ==================================== ========
SeShutdownPrivilege           Shut down the system                 Disabled
SeChangeNotifyPrivilege       Bypass traverse checking             Enabled
SeUndockPrivilege             Remove computer from docking station Disabled
SeIncreaseWorkingSetPrivilege Increase a process working set       Disabled
SeTimeZonePrivilege           Change the time zone                 Disabled
```

- **DACL** המשמש כברירת מחדל כאשר המשתמש יוצר אובייקטים חדשים (נרחיב בהמשך, בקצרה: מי יכול לעשות מה לאובייקט, לדוגמא אם משתמש מסוים לא\יכול לקרוא מהקובץ)
- **Type** המכיל מידע על אם זה טוקן ראשי או impersonation token (נרחיב בהמשך)
- ועוד..[2]

חשוב לציין שכל אובייקט אשר נוצר ע"י המשתמש יורש את ה-**Access Token** שלו. שימו לב לדוגמא הבאה בה רואים כיצד הפרטים מתוך ה-Access Token של המשתמש שלי זהים לתהליך notepad.exe שיצרתי:



[באמצעות Process Explorer ניתן לראות כיצד ה-Access Token של תהליך זהה לזה של המשתמש שיצר\הריץ אותו]

[2] https://learn.microsoft.com/en-us/windows/win32/secauthz/access-tokens

LSASS: על המנגנון שמייצר את ה-Access Token

מי שאחראי על יצירת ה-Access Token היא מערכת LSA ה-Local Security Authority אשר מונגשת על ידי התהליך lsass.exe. הוא מוסיף לטוקן שמייצג את המשתמש את ה-SIDs שהמשתמש שייך אליהם (הקבוצות).

הוא עושה זאת באמצעות בדיקה אל מול Local Group Policy (אשר בסביבה דומיינית מתעדכן אל מול פוליסות הדומיין של ה-DC) ולפי כל רשומות ה-SIDs שהמשתמש מקושר אליהם מקבל token access שמייצג את ההרשאות ומשאבים שאותו משתמש זכאי אליהם.

לאחר יצירת הטוקן, LSASS משכפל אותו ויוצר handle שניתן להעביר אל התהליך Winlogon.exe ביחד עם הודעה שהאימות התרחש בהצלחה. מאותו רגע, לכל התהליכים של המשתמש, יהיו את ההעתקים של אותו ה-Access Token. קרי, מה מותר ומה אסור לתהליכים של המשתמש לעשות.

כדי ללמוד על הקרביים של LSASS תרשמו לעצמכם לקרוא את המאמר המצוין של חברי יהונתן אלקבס Inside LSASS.

בחזרה לדוגמה שלנו: כשאותו משתמש מנסה כעת לפתוח קובץ, נוצרת בקשה (access request) בה רשום בנוסף גם מה המשתמש רוצה לעשות (**Access Right** - לקרוא\למחוק\להריץ) במקרה שלנו מדובר בקריאה בלבד ולכן הערך יהיה **GENERIC_READ**.

Access Right (מכל סוג: standard/specific/generic) הוא דגל (bit flag) שמסמל את סוג הגישה לאובייקט מסוים, לדוגמא אם תהליך רוצה לקרוא מפתח ברג'יסטרי הוא צריך Access Right Specific מסוג **KEY_READ**[3], אחרת הוא לא יקבל את הגישה.

בקוד זה יראה כך לדוגמא:

```
RegOpenKeyExA(HKEY_LOCAL_MACHINE,
"SOFTWARE\\Microsoft\\Windows\\CurrentVersion", 0, KEY_READ, &hKey);
```

ניתן לשרשר בבקשה אחת מספר Access Rights שונים (לקרוא ולערוך לדוגמא), התוצר המתקבל נקרא **Access Mask**: ערך בגודל 32 ביט שמכיל סט של Access Rights. בקוד שלנו במקום KEY_READ יהיה לנו פשוט KEY_READ | KEY_WRITE כאשר נרצה לקרוא ולערוך.

[3] https://learn.microsoft.com/en-us/windows/win32/sysinfo/registry-key-security-and-access-rights

ישנם 3 משפחות שונות של הרשאות אותן ניתן להגדיר ב-Access Mask. כל משפחה מאופיינת על פי scope ההרשאות שהיא מאפשרת.

Specific Access Rights: סט מאוד מסוים (ספציפי) של הרשאות עבור פעולות שמתאימות באופן ייחודי לאובייקטים מסויימים. למשל הרשאה לקרוא או לכתוב לאובייקט של קובץ, הרשאה לתשאל את הערך שב-Registry Key וכד'.

Standard Access Rights[4]: בנוסף להרשאות ספציפיות שרלוונטיות לסוג האובייקט (רג'יסטרי, קובץ וכו'), ישנן הרשאות יותר סטנדרטיות שרלוונטיות לכלל האובייקטים, לדוגמא ההרשאה למחוק את האובייקט (DELETE), או לשנות בעלים (WRITE_OWNER)

Generic Access Rights[5]: כפי שבטח ראיתם, ישנן המון הרשאות ספציפיות שונות, אז כדי לפשט את המנגנון ושמפתחים לא יצטרכו לזכור ולעבוד עם כ"כ הרבה specific access rights, מיקרוסופט יצרו לנו את ההרשאות הגנריות (כלליות) יותר: GENERIC_READ, GENERIC_EXECUTE, GENERIC_ALL ו-GENERIC_WRITE, זהו ☺

לדוגמא כשאנחנו משתמשים ב-GENERIC_READ על מנת לפתוח קובץ, מערכת ההפעלה מתרגמת אותו לסט שלם של הרשאות ספציפיות (FILE_READ_DATA, FILE_READ_EA and FILE_READ_ATTRIBUTES) וסטנדרטיות (READ_CONTROL and SYNCHRONIZE).

בחזרה לדוגמה שלנו: כעת שיש למשתמש Access Token (המתאר את ההרשאות שלו) ונוצרה לו בקשת גישה לקובץ (Access Request), הבקשה מועברת אל מנגנון קרנלי בשם **Security Reference Monitor** (ובקיצור **SRM**) שבתורו מסתכל על הטוקן והבקשה ומשווה אותם אל מול המבנה הרשאות של האובייקט (הקובץ בדוגמה שלנו) ומחליט האם לתת לפעולה אור ירוק או לא.

הבדיקה הזאת ב-SRM מתבצעת בפונקציה שנקראת SeAccessCheck.

המבנה הרשאות הזה של האובייקט נקרא **Security Descriptor** והוא שאחראי לתאר בפרוטרוט מי הבעלים של הקובץ (משתמש וקבוצה המפורטים ב-SID) ו-SACL ו-DACL (מי יכול לעשות מה ומה אסור למי לעשות על האובייקט הזה - עוד על שני החבר'ה הללו בהמשך).

הערה: לכל אובייקט (אשר בשפת מיקרוסופט מוגדר כ-***Securable Object***) יש Security Descriptor - כולל מפתחות רג'יסטרי, קבצים ותיקיות.

[4] https://learn.microsoft.com/en-us/windows/win32/secauthz/standard-access-rights
[5] https://learn.microsoft.com/en-us/windows/win32/secauthz/generic-access-rights

דוגמא לאחד שכזה של קובץ טקסט שהמשתמש שלי יצר (את כל השדות נבין עד סוף המאמר לא להיבהל):

```
PS C:\WINDOWS\system32> Get-Acl -AllCentralAccessPolicies -audit C:\temp\READaMEfds.txt | Format-List | Out-Host
Path   : Microsoft.PowerShell.Core\FileSystem::C:\temp\READaMEfds.txt
Owner  : DESKTOP-312DFT0\shaked ilan
Group  : DESKTOP-312DFT0\shaked ilan
Access : DESKTOP-312DFT0\shaked ilan Allow  FullControl
         BUILTIN\Administrators Allow  FullControl
         NT AUTHORITY\SYSTEM Allow  FullControl
         BUILTIN\Users Allow  ReadAndExecute, Synchronize
         NT AUTHORITY\Authenticated Users Allow  Modify, Synchronize
Audit  : DESKTOP-312DFT0\shaked ilan Success  DeleteSubdirectoriesAndFiles, Modify, ChangePermissions, TakeOwnersh
Sddl   : O:S-1-5-21-287014810-1449389800-2189324088-1001G:S-1-5-21-287014810-1449389800-2189324088-1001D:AI(A;;FA;
         ;;BA)(A;ID;FA;;;SY)(A;ID;0x1200a9;;;BU)(A;ID;0x1301bf;;;AU)S:AI(AU;SA;CCDCLCSWRPWPDTLOCRSDRCWDWO;;;S-1-5-
```

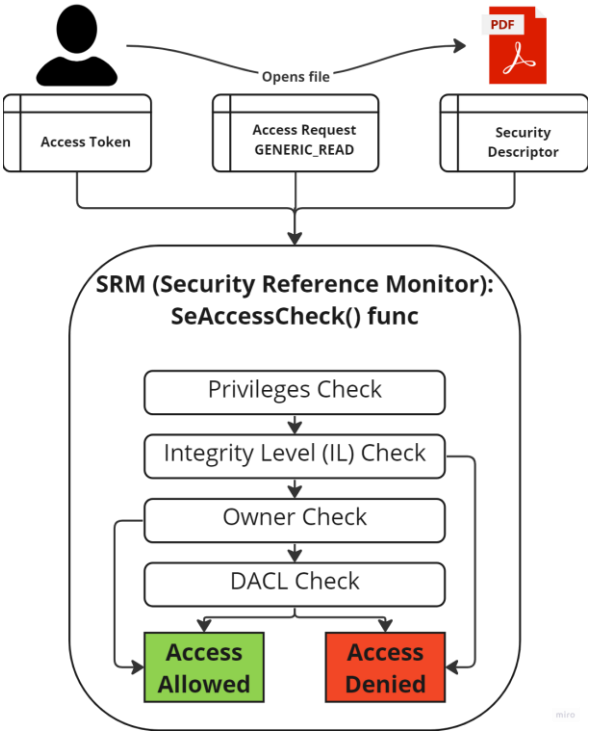
[Security Descriptor של קובץ טקסט שיצרתי]

השדה Sddl המוזר בתחתית נקרא Security Descriptor Definition Language ואין מה להיבהל ממנו - הוא רק פורמט לייצוג כל המידע שמוצג בשורות מעליו.

סיכום התרחיש עד כה:

1. משתמש עושה Logon ולאחר התחברות מוצלחת, מערכת ה-LSA יוצרת ושולחת אליו את ה-Access Token המתאר את ההרשאות שלו
2. המשתמש לוחץ פעמיים ומבקש לפתוח קובץ טקסט רנדומלי
3. נוצרת בקשה (Access Aequest) המתארת את סוג הגישה המבוקש (Access Right)
4. מנגנון ה-SRM מפרסר את השדות השונים בטוקן להבנת קונטקסט ההרשאות של המשתמש, משווה את סוג הגישה המתבקשת אל מול ה-Security Descriptor של הקובץ ומחליט האם להתיר גישה או לא.

אלו הן הפעולות שמתרחשות בכל פעם שאתם רוצים לקרוא קובץ:



[קיבלתי השראה מ-https://www.youtube.com/watch?v=QRpfvmMbDMg]

שימו לב שבאיור ישנם שלבים שעוד לא עברנו עליהם, כעת נלמד לעומק על עוד כמה מנגנונים ונחזור לדוגמה שלנו בהמשך.

על Security Descriptors ולמה הם טובים

הערה: מכאן והלאה אשתמש בעיקר ב-*TokenUniverse* על מנת להדגים, ממליץ להוריד ולשחק איתה במקביל למאמר (יש גרסת *exe* מקומפלת ומוכנה ב-*Releases*)

אז כמו שציינו - לכל אובייקט (**Securable Object**) ב-Windows יש Security Descriptor, שמכיל שני מבנים חשובים: SACL ו-DACL, עליהם ה-SRM מסתכל כאשר הוא בא לקבל החלטה אם לאשר גישה או לא. בואו נבין אותם:

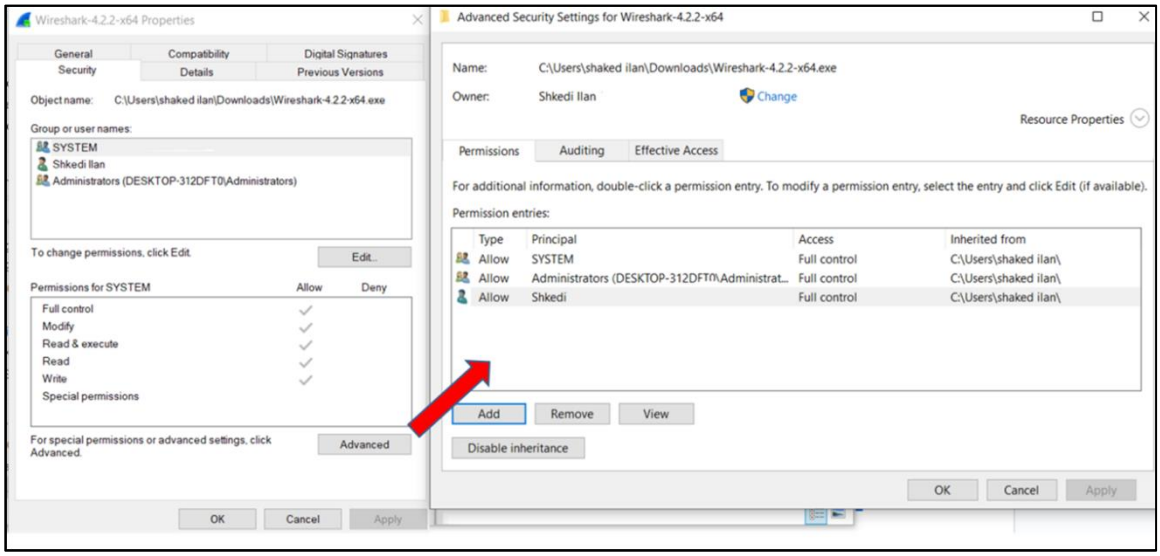
Discretionary Access Control List (DACL):

DACL הוא הרלוונטי יותר לתהליך ה-Authorization - הוא מגדיר את סוג הגישה שיש למשתמשים\קבוצות לאובייקט והוא בעצמו מוגדר ע"י ה-Owner של האובייקט (לכל DACL מוגדר Owner ו-Group).

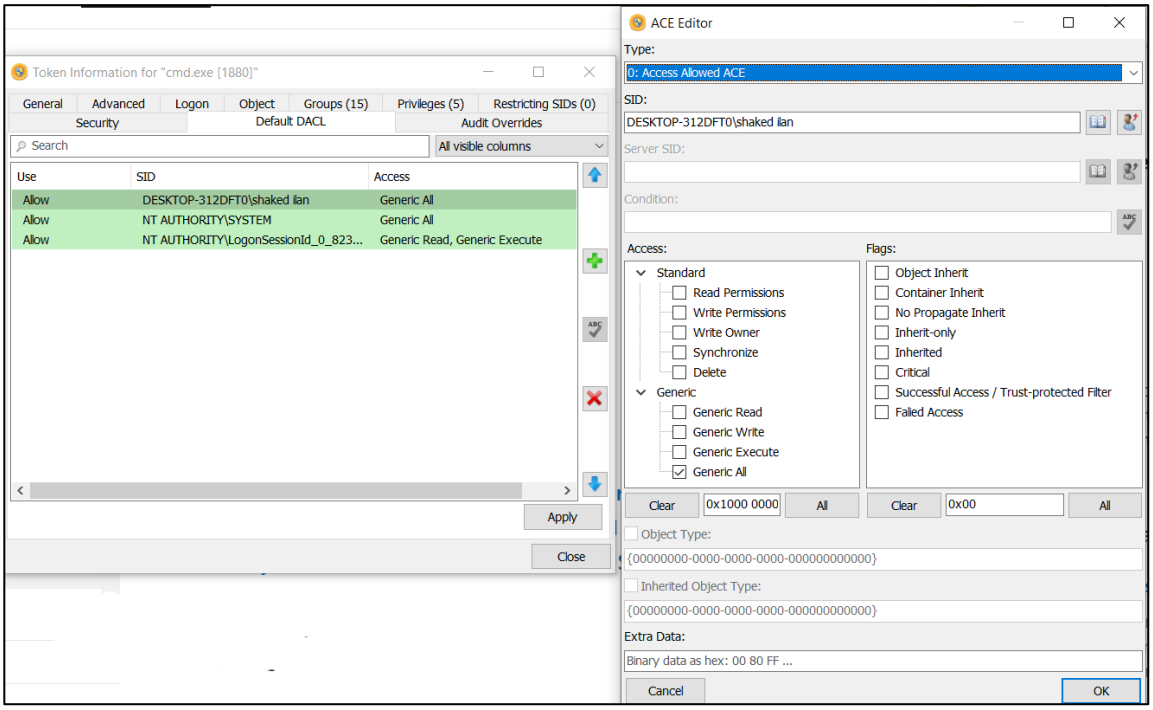
לכל DACL יכול להיות 0 או יותר רשומות הנקראות **Access Control Entries** (ובקיצור **ACE**) וישנן 6 סוגים מהן, כאשר שלושה הם ספציפיות לאובייקטים מסוג מסוים (לא נכנס אליהן) ושלושה עיקריות:

Access-denied ACE	רשומה שמי שיתואר בה יחסם מלקבל גישה לאובייקט לפי סוג הגישה שיתואר. **לדוגמא**: המשתמש משה לא יוכל להריץ את האובייקט *(GENERIC_EXECUTE)*
Access-allowed ACE	רשומה שמי שיתואר בה יקבל גישה לאובייקט לפי סוג הגישה שיתואר. **לדוגמא**: חברי קבוצת *X* יוכלו לקרוא את האובייקט (*GENERIC_READ*)
System-audit ACE	יירשם לוג כאשר מי שמתואר ברשומה מבצע את סוג הגישה המתואר.

דוגמא ל-DACL של קובץ התקנה של WIRESHARK שהורדתי:



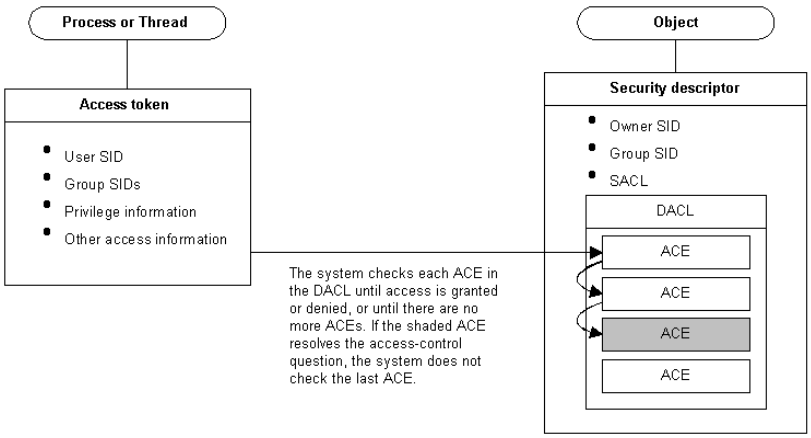
ובתמונת המסך הבאה ניתן לראות את שלושת ה-ACEs שנוצרים דיפולטיבית בתהליך cmd.exe שהרצתי, כאשר ה-ACE הפתוח בחלון הוא מסוג Access-allowed ACE:



[באמצעות TokenUniverse ניתן לראות את ה-ACEs ומבנה של אחד מהם ב-DACL של ה-cmd.exe]

אם אין DACL בכלל לאוביקט (*Null DACL*) Windows - נותנת גישה חופשית לכולם, אם יש אבל אין רשומות ACE (נקרא *Empty DACL*) - אין גישה כלל. כדי להבין לעומק את הלוגיקת קבלת החלטות המבוססת על DACL (סדר החוקים, מתי Windows נותנת גישה ומתי לא) ממליץ לפנות למאמר הבא.

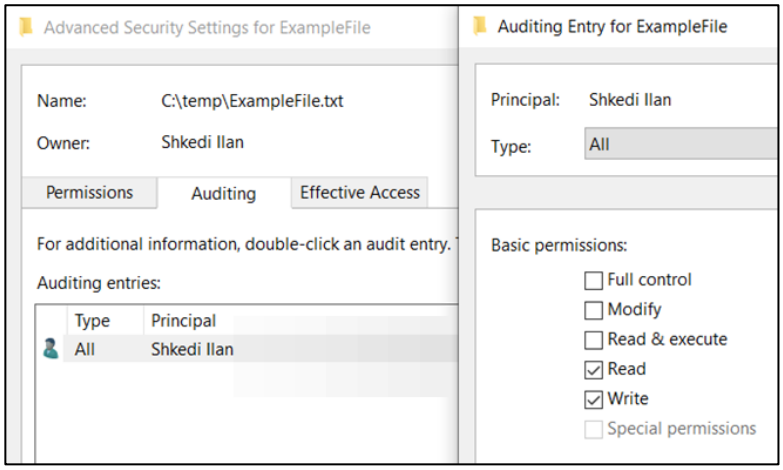
אז עכשיו אנחנו יכולים להבין מול מה בדיוק **SRM** משווה כשהוא מקבל בקשה: הוא מסתכל על ה-Requesting User SID (אותו הוא מוציא מה-Access Token) ועל ה-Access Right המבוקש, משווה את ה-SID של המבקש אל מול ה-Owner/Group הרשומים ב-DACL. ולאחר מכן עובר על כל אחת מהרשומות ACE (בסדר קנוני) ומשווה אותן ל-Access Right המבוקש עד אשר נמצאת התאמה המתירה או אוסרת את הגישה המבוקשת.



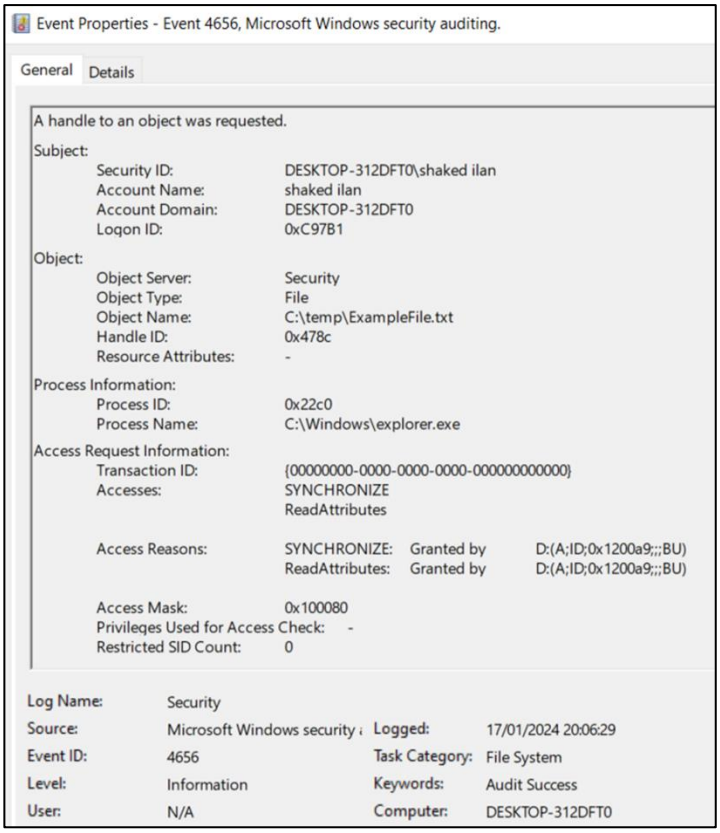
[מקור https://learn.microsoft.com/en-us/windows/win32/secauthz/interaction-between-threads-and-securable-objects]

System Access Control List (SACL)

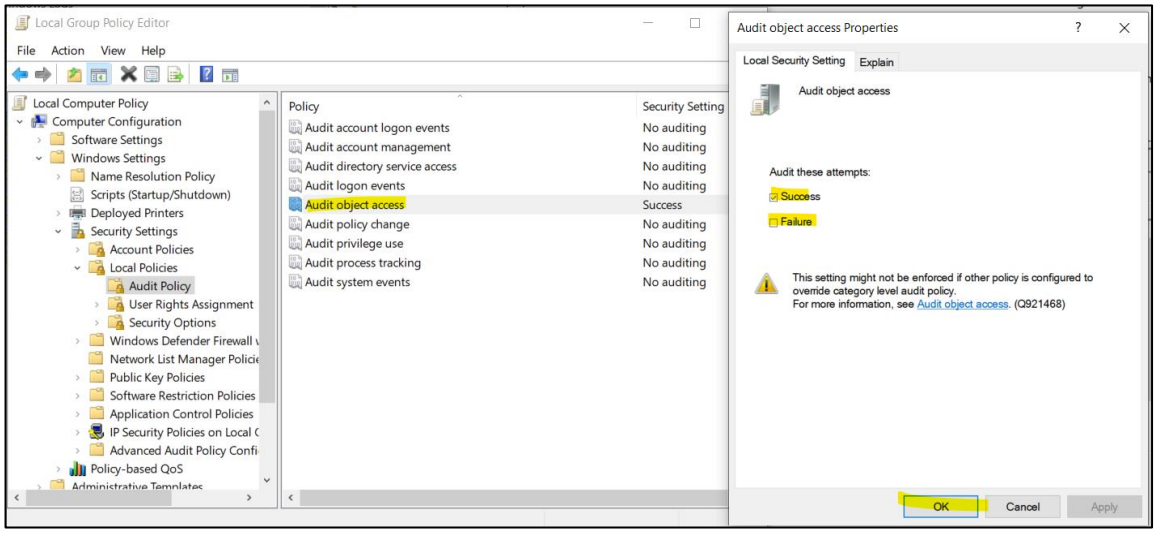
SACL מגדיר את ה-Integrity Level הנדרש (**MANDATORY_INTEGRITY_LABEL**, נרחיב בהמשך) ומתי ישמרו לוגים (audit messages) והוא בעצמו מוגדר ע"י מנהלי המערכת (לאו דווקא ה-Owner, בשונה מ-DACL). גם הוא בנוי על בסיס ACEs אך ההבדל היחידי הוא שכאשר יש match, במקום מתן הרשאת גישה תרשם שורה בלוג. לרוב ישתמשו בו למטרות Compliance ו-Auditing, לדוגמא בהגדרות SACL הבאות:



נוצר הלוג הבא כאשר פתחתי את הקובץ:



חשוב לציין שעל מנת להפעיל את המנגנון יש להדליק את ההגדרה ב-Local Group Policy Editor:



תוכלו לנסות בעצמכם בסקריפט שהכנתי כאן (שימו לב שנדרש להריצו כאדמיניסטרטור).

החלק הנוסף שיש ב-SACL מתייחס ל-integrity levels ויש לו בדיוק אותה משמעות: "תרשום לוג כאשר יש גישה מאובייקט בעל integrity level שונה מהאובייקט שלי" (לרוב תהליך בעל integrity level נמוך יותר מנסה לגשת לתהליך בעל רמה גבוהה ממנו).

אז מה למדנו? **SACL** לרוב ישמש למטרות **Auditing** והתראות על אובייקטים חשובים) תיקייה השייכת לצוות א' ועובד מצוות לא קשור (תוקף?) מנסה לגשת אליה, קובץ סיסמאות רגיש וכו') שגורמים מסוימים לא אמורים לגעת בהן.

ואתם צודקים - במקרה של הדוגמא שלנו הוא פחות רלוונטי, אבל כדאי מאוד להכיר את המנגנון הזה מכיוון שהוא ליבתי במערכת ההפעלה, במיוחד כשמדברים על פתרונות הגנה וניטור. מהצד הכחול, יצירת מספר אובייקטי decoy שמהווים מלכודות דבש (כדוגמת קובץ סיסמאות ש"נשכח" בתיקיית share) אשר מנוטרים בצורה צמודה עם SACL היא טקטיקה קונספטואלית נכונה. אי לכך מהצד האדום, בתור תוקפים, חשוב מאוד לשמור על היגיינת OpSec ראויה ולכן נרצה תמיד לדעת האם יש על אובייקטים רגישים (כדוגמת קבצים) ניטור מיוחד על ידי בדיקת שדה ה-SACL.

בשאילתה מאוד פשוטה ב-PowerShell:

```
Get-Acl -audit file.txt | Format-List | Out-Host
```

אפשר לדעת מה ה-SACL של אובייקט ולהבין האם כדאי לבצע את הפעולה הזדונית שלנו ☺

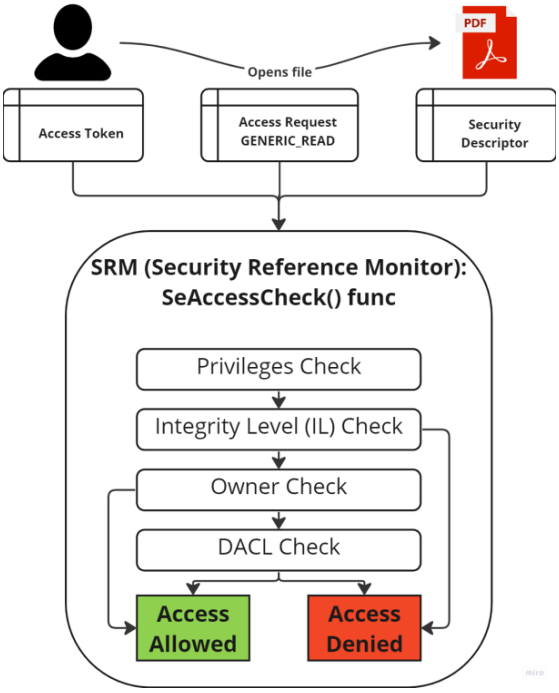
Integrity Level Check / Mandatory Integrity Control (MIC)

אז מה זה ה-integrity level (ובקיצור IL) הזה בכלל? מיקרוסופט החליטו שכל מה שתיארנו עד כה לא מספיק fine-grained בשביל לספק אבטחה ברזולוציה טובה במיוחד והחל מ-Windows ויסטה הוסיפו מנגנון חדש בשם **Mandatory Integrity Control**[6] (ובקיצור **MIC**) שהתווסף ל-Security Descriptor של כל אובייקט ב-Windows.

מטרת המנגנון היא לייצג את רמת המהימנות של האובייקט. הוא בנוי מ-4 רמות עיקריות שונות:

Low (or Untrusted), **Medium**. **High** and **System**

אם תסתכלו שוב באיור המתאר את תהליך הבדיקה של ה-SRM, תוכלו להבחין שה-SRM מבצע בדיקה אל מול MIC לפני שהוא בכלל מגיע לבדיקת DACL!:



לכן אם תהליך ב-Integrity Level Medium ינסה לגשת לתהליך מהימן יותר ממנו (לדוגמא עם רמת SYSTEM) - ה-SRM יעיף אותו כבר שם בבדיקה הזו. לדוגמא ב-cmd.exe שפתחתי (בלי UAC ובלי להיות Elevated - ככה סתם) הוא נראה כך (השתמשתי ב-*TokenUniverse*):

Caption	Granted Access	User Name	Token Type	Session	Elevation	Integrity
cmd.exe [1880]	Full Access	DESKTOP-312DFT0\shaked ilan	Primary	1	No (Limited)	Medium

[6] https://learn.microsoft.com/en-us/windows/win32/secauthz/mandatory-integrity-control

שימו לב שכברירת מחדל Windows מעניק Medium IL לכל תהליך שהמשתמש שלי (למרות שהוא אדמין מקומי) יצר. רק כאשר אצור את התהליך כאשר שאני **Elevated** (נתתי הרשאה ל-UAC) - התהליך יקבל High IL:

Caption	Granted Access	User Name	Token Type	Session	Elevation	Integrity
cmd.exe [16724]	Full Access	DESKTOP-312DFT0\shaked ilan	Primary	1	Yes (Full)	High

וגם פה ניחשתם נכון, כל אובייקט שיווצר ע"י המשתמש SYSTEM יקבל גם דיפולטיבית **System** Integrity level.

הבה וניצור קובץ מ-cmd.exe רגיל (לא **Elevated**) ונראה שכרגע עוד לא מוגדר לו IL (באמצעות ***icacls***, בינארי המגיע כברירת מחדל עם Windows ומאפשר לצפות ולערוך DACL/SACL של אובייקטים):

```
C:\temp>echo test> test.txt

C:\temp>icacls test.txt
test.txt BUILTIN\Administrators:(I)(F)
         NT AUTHORITY\SYSTEM:(I)(F)
         BUILTIN\Users:(I)(RX)
         NT AUTHORITY\Authenticated Users:(I)(M)
```

[את ההשראה להדגמה לקחתי מ-*hacktricks*]

כעת מ-cmd.exe **Elevated** נחיל עליו Integrity level **High**:

```
C:\temp>icacls test.txt /setintegritylevel(oi)(ci) High
processed file: test.txt
Successfully processed 1 files; Failed processing 0 files


C:\temp>icacls test.txt
test.txt BUILTIN\Administrators:(I)(F)
         NT AUTHORITY\SYSTEM:(I)(F)
         BUILTIN\Users:(I)(RX)
         NT AUTHORITY\Authenticated Users:(I)(M)
         Mandatory Label\High Mandatory Level:(NW)
```

ושימו לב שכעת כאשר ה-cmd.exe הרגיל שלי (שהוא medium level) ינסה לגשת שוב לקובץ (שהוא יצר אותו במקור!) הוא נכשל:

```
C:\temp>type test.txt
Access is denied.

C:\temp>move test.txt test2.txt
Access is denied.
        0 file(s) moved.
```

הערה: לא לכל הקבצים והתיקיות מוגדר IL, אך לכל התהליכים תמיד יוגדר IL.

בחזרה לדוגמה שלנו: כשה-SRM יבוא לבדוק את ה-integrity level של התהליך המבקש אל מול ה-integrity level של האובייקט המבוקש, הוא אמור להעיף אותו אם הוא יותר נמוך נכון? אז בגדול כן, אבל זה טיפה יותר מורכב:

ה-**IL** לכל אובייקט מוגדר ב-**SACL** תחת struct בשם **SYSTEM_MANDATORY_LABEL_ACE**[7] האחראי לתאר את **Integrity Level Policy** - המדיניות שקובעת מה יקרה במידה ואובייקט בעל IL נמוך יותר מנסה לגשת לאובייקט שלנו, המדיניות הזו שמורה תחת המשתנה **Mask**[8] ב-struct הנ"ל.

icacls שהשתמשנו בו מקודם בעצם מפרסר את המבנה הזה ומציג אותו.

למתקדמים: תחת SYSTEM_MANDATORY_LABEL_ACE ישנו struct נוסף בשם **MANDATORY_INTEGRITY_LABEL** שמכיל SID שמגדיר את ה-IL של האובייקט (low\medium\high\system).

ומי קובע מה ה-IL של המשתמש שלנו? את זה קובע מנגנון ה-LSA[9] כשאנחנו מתחברים למכונה! והוא מוכל כבר ב-Access Token של המשתמש. שאלה שכנראה עברה לחלקכם בראש היא אבל מה הרעיון? כאילו, למה בכלל צריך את המנגנון של **MIC** אם יש לנו את המנגנון של **DACL**? תשובה: אם תסתכלו ב-Process Explorer, כשאתם פותחים את כרום, הטאבים שלכם רצים תחת תהליכים ברמה הנמוכה ביותר: **Untrusted**:

Caption	Granted Access	User Name	Token Type	Session	Elevation	Integrity
chrome.exe [14968]	Full Access	DESKTOP-312DFT0\shaked ilan	Primary	1	No (Limited)	Untrusted

ולעומת זו, explorer.exe (או כל קובץ\אובייקט אחר ששייך למשתמש שלי) רץ תחת medium:

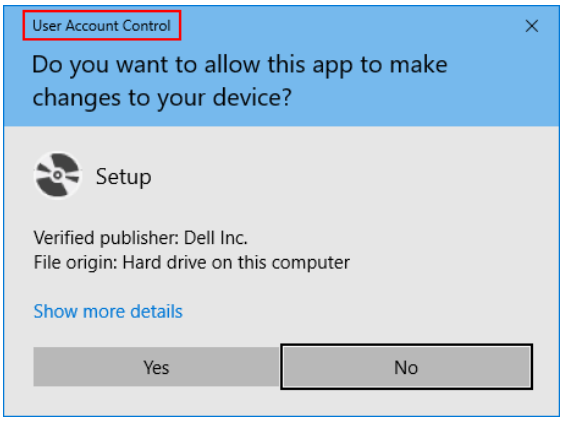
Process	Integrity
explorer.exe	Medium

והסיבה היא פשוטה: אם הרוסים או סינים הזריקו לכם חולשה והצליחו להריץ קוד בתוך הדפדפן שלכם, נרצה לשמור אותם שם, שלא יוכלו לגעת ב-Registry או במערכת הקבצים שלכם, בלי קשר בכלל להרשאות DACL - נרצה שה-SRM יעיף אותם עוד הרבה לפני. הגיון פשוט שטאב של דפדפן החשוף לאינטרנט חייב בהגדרה להיות לא מהימן (**Untrusted**) אל מול תהליך שהמשתמש שלי יצר. מ.ש.ל. ☺

[7] https://learn.microsoft.com/en-us/windows/win32/api/winnt/ns-winnt-system_mandatory_label_ace
[8] https://learn.microsoft.com/en-us/windows/win32/api/winnt/ns-winnt-system_mandatory_label_ace
[9] https://learn.microsoft.com/en-us/windows/desktop/SecGloss/l-gly

UAC - User Access Control

UAC הוא מנגנון אבטחה נוסף (הפופ אפ המעצבן שכולנו מכירים היטב) שמיקרוסופט הוסיפו החל מ-Windows ויסטה והוא נועד על מנת למנוע ממשתמשים להריץ בטעות תהליכים בהרשאות גבוהות אם הם לא התכוונו לכך או כשהן לא נדרשות:



[מקור: https://www.top-password.com/blog/3-ways-to-turn-on-off-uac-in-windows-10/]

זוכרים שסיפרתי לכם שבברירת מחדל כל התהליכים שלנו ירוצו תחת Integrity Level **Medium**? מעולה! עכשיו תארו לכם עולם (Windows XP) שבו כל התהליכים רצים בהרשאות גבוהות (כיף לסינים ולרוסים) - ויי זמיר! בדיוק בשביל זה הגיח לעולם UAC.

ולכן כיום, כשאנחנו עושים Logon בהצלחה, נוצרים בעצם 2 טוקנים למשתמש שלנו:

- **Full (Admin) Token** - נקרא גם Non-filtered Token והוא בעל High IL and Admin Privileges
- **Filtered (Admin) Token** - נקרא גם Standard/Restricted Token והוא בעל Medium IL and Reduced Privileges

כפי שראינו, בברירת המחדל, כאשר אנחנו עובדים במערכת בלי לבקש UAC, לדוגמא בעת יצירת תהליך cmd.exe חדש - התהליך ירוץ בהרשאות Medium זאת מכיוון ש-**Windows** משתמשת ב-**Filterd Token** ליצירת התהליך.

אך כאשר נעבור את מסך ה-UAC בלחיצת *Yes* - התהליך ירוץ באמצעות ה-**Full Token** ולכן יקבל **High IL**! (זה אגב מה שקורה כשאנחנו משתמשים ב-*Run as Administrator*).

לקריאה נוספת על המנגנון בדגש על התחברות מרחוק (RDP) ותקיפות בסגנון Pass-the-Hash ממליץ לקרוא את המאמר כאן.

Privileges Based Model

בתרגום חופשי מ-MSDN: הרשאה ב-Windows מגדירה אילו פעולות ניתן לעשות במחשב - כגון כיבוי, טעינת דרייברים, שינוי הזמן וכד'. הרשאות שונות מ-Access Rights שתיארנו מקודם בכך שהן מגדירות מה ניתן לעשות בכלל המערכת (והן ניתנות למשתמש או לקבוצה), בעוד ש-Access Rights מגדירות את סוג הגישה המותרים\אסורים על אובייקט ספציפי (ושעל בסיסים המערכת מחליטה האם לאשר את הגישה).

בכל מחשב יש DB קטן בשם SAM המחזיק את ההרשאות שיש לכל חשבון (משתמש\קבוצה) במחשב (אותו DB קיים גם ב-Active Directory הנקרא NTDS.dit והוא מעט יותר עשיר ומורכב), וכשמשתמש עושה logon הוא מקבל **Access Token** (כן כן, אותו אחד מתחילת המאמר) המכיל את רשימת ההרשאות שלו (ובהתאם ב-AD: למשתמש אחר המתחבר לאותו מחשב יכול להיות הרשאות שונות) וכאשר המשתמש מנסה לבצע פעולה כלשהיא, המערכת בודקת האם מותר לו לפי הרשימה הזו.

על מנת להבין אילו הרשאות יש לנו, נריץ *whoami /all*:

```
PRIVILEGES INFORMATION
----------------------

Privilege Name                Description                          State
============================= ===================================== ========
SeShutdownPrivilege           Shut down the system                 Disabled
SeChangeNotifyPrivilege       Bypass traverse checking             Enabled
SeUndockPrivilege             Remove computer from docking station Disabled
SeIncreaseWorkingSetPrivilege Increase a process working set       Disabled
SeTimeZonePrivilege           Change the time zone                 Disabled
```

[whoami /all from a **non-elevated** cmd.exe]

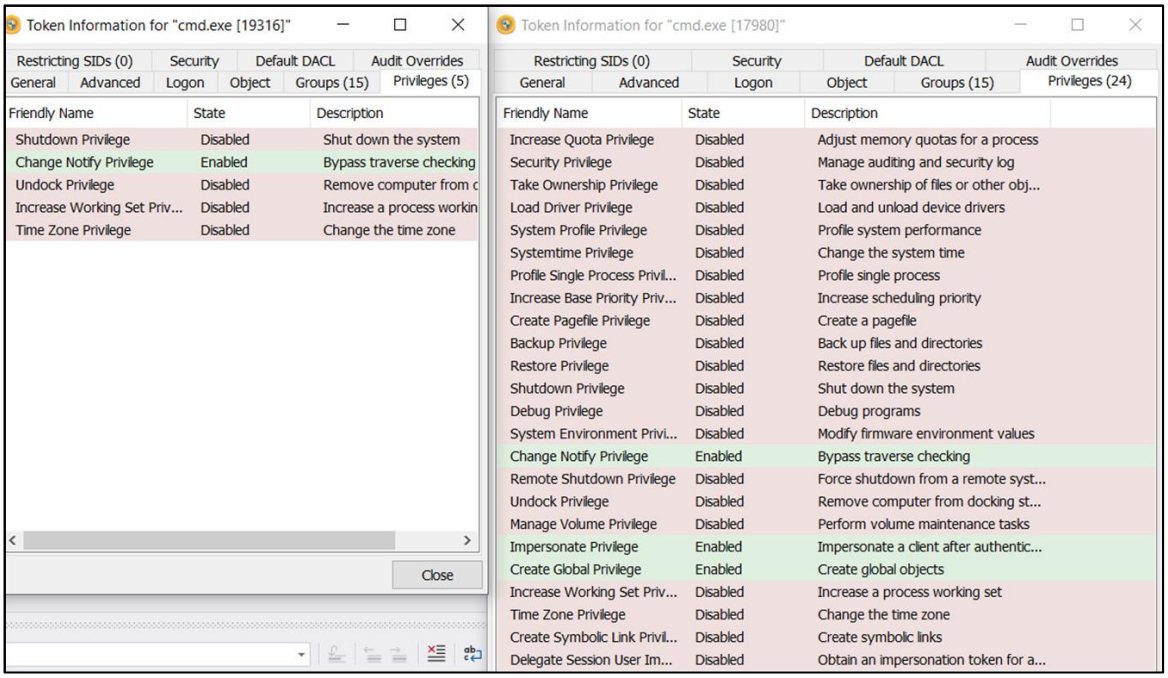
כעת, תראו איזה יופי של הרשאות קיבלנו כשאנחנו רצים מ-cmd בעל הרשאות אדמיניסטרטור (IL High):

```
PRIVILEGES INFORMATION
----------------------

Privilege Name                            Description                                                        State
========================================= ================================================================== ========
SeIncreaseQuotaPrivilege                  Adjust memory quotas for a process                                 Disabled
SeSecurityPrivilege                       Manage auditing and security log                                   Disabled
SeTakeOwnershipPrivilege                  Take ownership of files or other objects                           Disabled
SeLoadDriverPrivilege                     Load and unload device drivers                                     Disabled
SeSystemProfilePrivilege                  Profile system performance                                         Disabled
SeSystemtimePrivilege                     Change the system time                                             Disabled
SeProfileSingleProcessPrivilege           Profile single process                                             Disabled
SeIncreaseBasePriorityPrivilege           Increase scheduling priority                                       Disabled
SeCreatePagefilePrivilege                 Create a pagefile                                                  Disabled
SeBackupPrivilege                         Back up files and directories                                      Disabled
SeRestorePrivilege                        Restore files and directories                                      Disabled
SeShutdownPrivilege                       Shut down the system                                               Disabled
SeDebugPrivilege                          Debug programs                                                     Disabled
SeSystemEnvironmentPrivilege              Modify firmware environment values                                 Disabled
SeChangeNotifyPrivilege                   Bypass traverse checking                                           Enabled
SeRemoteShutdownPrivilege                 Force shutdown from a remote system                                Disabled
SeUndockPrivilege                         Remove computer from docking station                               Disabled
SeManageVolumePrivilege                   Perform volume maintenance tasks                                   Disabled
SeImpersonatePrivilege                    Impersonate a client after authentication                          Enabled
SeCreateGlobalPrivilege                   Create global objects                                              Enabled
SeIncreaseWorkingSetPrivilege             Increase a process working set                                     Disabled
SeTimeZonePrivilege                       Change the time zone                                               Disabled
SeCreateSymbolicLinkPrivilege             Create symbolic links                                              Disabled
SeDelegateSessionUserImpersonatePrivilege Obtain an impersonation token for another user in the same session Disabled
```

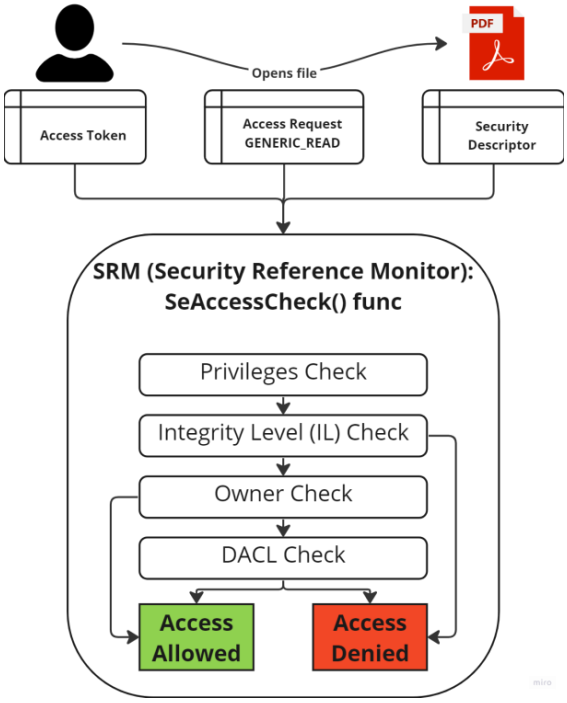
[whoami /all from an **elevated** cmd.exe]

ואותה הגברת ב-TokenUniverse:



מודל האבטחה של Windows מתבסס על העובדה שכל תהליך שרץ עם טוקן בעל הרשאות Debug (כמו זה שיש למשתמש שהוא Elevated כמו בתמונה מעלה; אגב שימו לב שבתמונה הוא Disabled אך אפשר לאפשר אותו) יוכל לבקש ולקבל גישה לכל תהליך הרץ במערכת ההפעלה: כלומר במילים פשוטות הוא יוכל לקרוא ולכתוב לזיכרון התהליך, להזריק קוד, להשהות ולחדש ריצה של threads, לתשאל מידע על תהליכים אחרים ובמקרה שלנו שנדגים בהמשך: **להתחזות**!

וכעת נוכל להבין את התהליך של גישה אל אובייקט מקצה לקצה:



על התחזות - Impersonation

כעת אנחנו מגיעים לחלק המעניין! היכולת להתחזות למשתמשים אחרים באמצעות שכפול (מישהו אמר "גניבת"?) טוקנים: יש פיצ'ר מדהים ב-Windows המאפשר לשכפל טוקנים של תהליכים (בהינתן ויש את ההרשאות המתאימות מהסוג שהזכרנו למעלה) על מנת להתחזות לכל משתמש אחר (כולל אפילו ל-SYSTEM/NETWORK SERVICE).

שימוש לגיטימי

נניח ויש לנו Windows Service שרץ כ-SYSTEM שמטרתו לגבות מסמכים של עובדים: בכל פעם שמשתמש בדומיין מתחבר לעמדה במחשב האישי שלו, השירות שלנו דואג לגבות את המסמכים מהמחשב האישי לתיקייה רשתית בשרת קבצים שרק לאותו משתמש יש גישה אליה (לטובת פרטיות ע"מ שמשתמשים אחרים לא יוכלו להיכנס לו לתיקייה).

העובד הוא משתמש בדומיין כמובן, אך המשתמש של השירות שלנו הוא SYSTEM ולכן אין לשירות יכולת להתחבר לתיקיה הרשתית כי הדומיין לא מכיר בו. מה עושים? אחת האופציות שלנו היא **Impersonation: via Token Duplication!** (אופציות אחרות הן הרצת ה-Service בהרשאותיו של המשתמש, או להשתמש בפיצ'ר בשם S4U-Based Impersonation אבל לא נכנס אל סוגי דלגציות במאמר זה).

השירות יעניק לעצמו (או לתהליך חדש שהוא יריץ) את הזהות של המשתמש המחובר לעמדה באמצעות שכפול את ה-Access Token של המשתמש, יזדהה ויעתיק קבצים לתיקייה הרשתית.

צוללים לפרטים

אז מצאנו דרך מעולה להתחזות למי שבא לנו ולבצע Privilege Escalation חופשי נכון? כמובן שלא, מיקרוסופט מגבילים את היכולת לבצע התחזות (לשכפל טוקנים במילים מדויקות יותר) ע"י הרשאות ספציפיות שנדרשות (מהסוג שהזכרנו למעלה) ורמת התחזות (שמגבילה את הפעולות שניתן לבצע באמצעות ה-Impersonated Token). נסביר:

לכל תהליך יש 2 סוגי טוקנים - **Primary** and **Impersonation**:

- **Primary Token**:
  - מייצג את ה-Security Context של התהליך עצמו: בתוכו רשום זהות המשתמש, קבוצות שהוא חבר בהן והרשאות
  - נוצר כשהתהליך עצמו נוצר, יש רק אחד כזה ונשאר איתו לכל אורך חייו עד שהתהליך נסגר
  - זה ה-Access Token עליו דיברנו לאורך כל המאמר עד כה, לפיו המערכת מחליטה האם לאשר גישה לאובייקט או לא

- **Impersonation Token**:
  - מאפשר לתהליך לרוץ <u>באופן זמני</u> עם טוקן של זהות אחרת (ולכן גם עם Security Context של אותה זהות)
  - בהתאם מאפשר את אותה גישה והרשאות כמו של הזהות המשוכפלת (במגבלת ה-Impesonation Level עליה נדבר עוד רגע)
  - משיגים אותו באמצעות קריאה לפונק' Windows API כמו:

DuplicateToken/ImpersonateLoggedOnUser/LogonUser

בנוסף, כחלק מה-Access Token של תהליך, יש שדה בשם **Impersonation Level** והוא קובע מה הרמה המקסימלית ששירות\תהליך אחר שמתחזה אליו יכול לאמץ - ישנן רמות שונות של Impersonation ולכל אחת הרשאות ואפשרויות שונות[10]:

- **Anonymous**:
  - ללא זהות ספציפית, הצד המתחזה (זה שיוצר את ה-Impersonation Token' נקרא לו מעתה השרת) מזדהה כאנונימי ולכן ניגש למשאבים כאנונימי (בתקשורת RPC / Named Pipes) בלי להזדהות
  - שמיש רק ב-Interprocess Communication (מוכר יותר כ-IPC) בהעברת מידע בין תהליכים, בעזרת Named Pipes לדוגמא
  - לא נדרשות הרשאות מיוחדות

- **Identification**:
  - ברירת המחדל, מאפשר לקבל פרטים על הטוקן שאליו מעוניינים להתחזות אך לא מאפשר לבצע פעולות בשמו (read/write/execute), רמה זו תשמש על מנת לבצע בדיקות ACL (למה בדיוק הזהות הזו נגישה)
  - דוגמא תהיה אם אנחנו database ולקוח מבקש מאיתנו (דרך IPC) לגשת לטבלה שאמורה להיות נגישה לו לפי ההרשאות שלו: נוכל לבדוק זאת באמצעות התחזות ברמה הזו על מנת לדעת מי הוא ולוודא שאכן יש לו גישה, בלי לבצע פעולות בשמו
  - נדרשת הרשאה מסוג *SeAssignPrimaryTokenPrivilege*

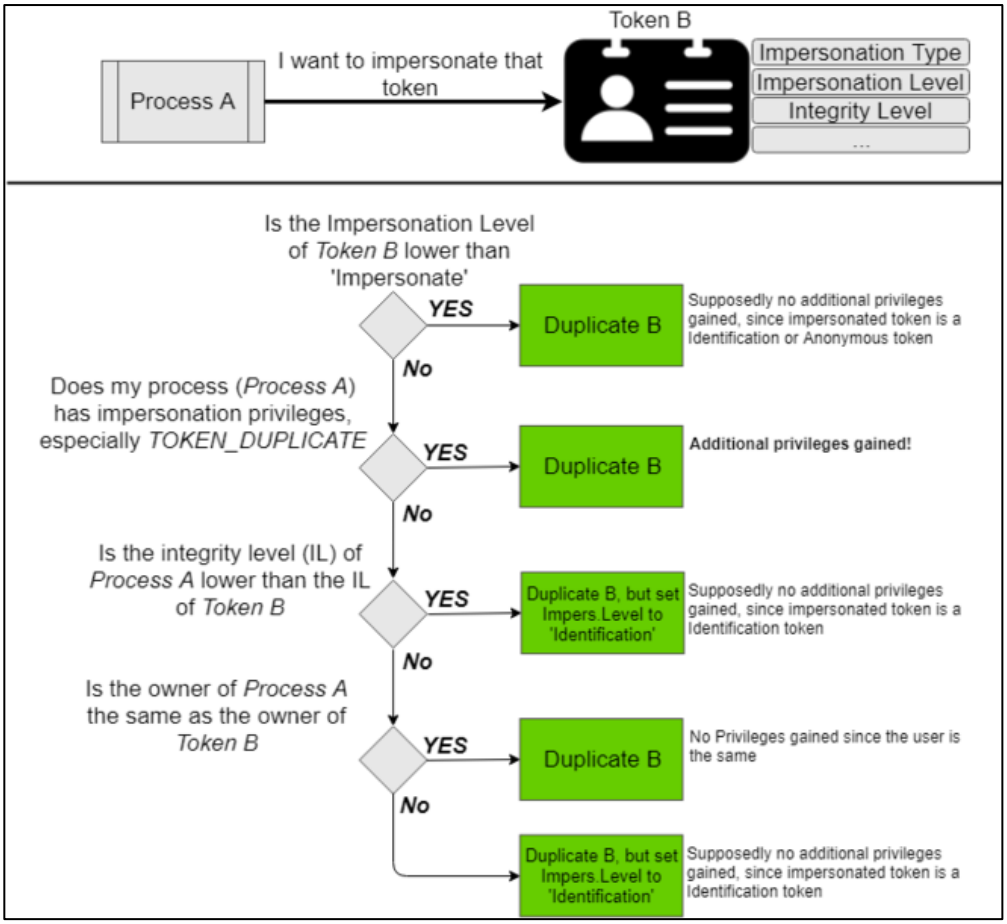
- **Impersonate**:
  - עכשיו זה מתחיל להיות מעניין: מאפשר לקבל security context של הלקוח (לדוגמא התהליך ממנו נרצה לגנוב את הטוקן) ולבצע פעולות בשמו באותו השרת (לא מרוחק)
  - נדרשת הרשאה מסוג *SeImpersonatePrivilege*

[10] https://learn.microsoft.com/en-us/windows/win32/com/impersonation-levels

- **Delegation**:
  - הכי חזק: כמו Impersonate level אך מאפשר גם להזדהות ולגשת לשירותים מרוחקים
  - נוצר כאשר משתמש מתחבר למכונה פיזית או באמצעות RDP/VNC
  - נדרשת הרשאה מסוג *SeDelegateSessionControlPrivilege*

על מנת לשכפל טוקן (לבצע התחזות) נצטרך הרשאה מיוחדת מסוג TOKEN_DUPLICATE, לקריאה נוספת[11]. למי שמעוניין לצלול לפרטים:

1. זה התהליך השלם שקורה כש-Windows מחליט האם ניתן לשכפל טוקן:



[מקור https://csandker.io/2018/06/14/AWindowsAuthorizationGuide.html#impersonation]

2. כפי שיהונתן ציין במאמרו[12] זה שיש למשתמש הרשאה מסויימת בטוקן לא אומר שהוא יכול להשתמש בה. לכל הרשאה יש משתנה בשם State ורק אם הוא במצב Enabled בטוקן עצמו אזי ניתן להשתמש בהרשאה. עם זאת, למרות שלא ניתן לשנות שדות בתוך הטוקן של משתמש מבלי שהוא יבצע התחברות מחדש, את אותו משתנה State אכן ניתן לשנות ועם קצת עזרה מהפונקציית API Win המתאימה (AdjustTokenPrivileges) לפעמים גם נוכל לבצע את זה ללא משתמש חזק.

[11] https://learn.microsoft.com/en-us/windows/win32/secauthz/access-rights-for-access-token-objects
[12] https://www.digitalwhisper.co.il/files/Zines/0x9C/DW156-3-LateralMovement-Part1.pdf

כיצד Impersonation בא לידי ביטוי כתוקפים?

כתוקפים נוכל להשתמש במנגנון על מנת להעלות הרשאות (Privilege Escalation) ולשנות הרשאות למשתמשים אחרים - גם במשתמשים מקומיים וגם במשתמשים דומיינים (לדוגמא ממשתמש מקומי שלא מחובר לדומיין - לדומיין אדמין). לכן ה-Impersonation Levels שמעניינים אותנו הם **Delegation** ו-**Impersonate** שמאפשרים לנו לבצע פעולות בשם הזהות המשוכפלת.

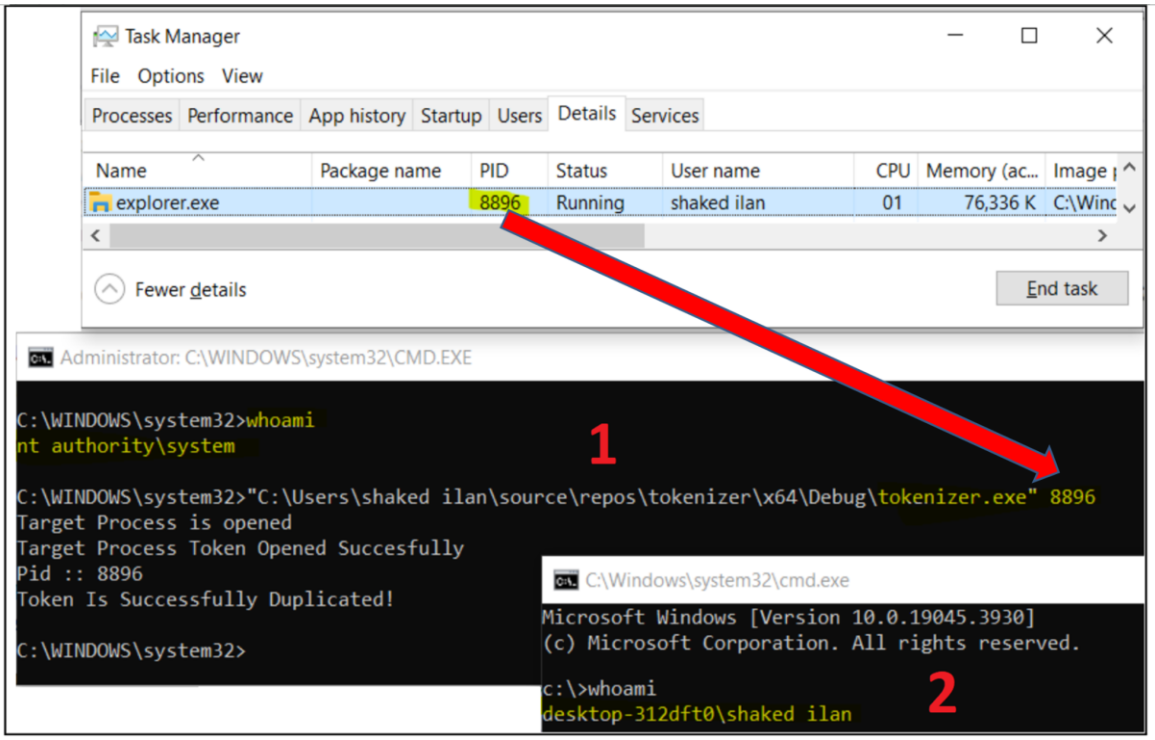
זוכרים את התרחיש מתחילת המאמר? נזכיר ונרחיב: במידה והתפרצנו לשרת (שהוא חלק מדומיין המחובר ל-AD) באמצעות חולשה באיזה Web App וכעת אנחנו רצים עם ה-shell שלנו כמשתמש SYSTEM או כ-Service אחר, זה אחלה ומגניב אבל זה לא מאפשר לנו להגיע לסביבה הדומיינית (לא לשרתי קבצים מעניינים ובטח לא ל-Domain Controller) ובעצם אנחנו תקועים בשרת אליו התפרצנו. פה בדיוק המנגנון רלוונטי בשבילנו!

כתוקפים נבקש למצוא תהליכים רצים במערכת שהם תחת משתמש דומייני כלשהוא ע"מ שנוכל לגנוב את הזהות שלו! (במידה ומשתמש מהדומיין כמו איש ה-IT התחבר ב-RDP או לחלופין ממש מחובר פיזית לשרת, יהיה לנו תהליכים רצים עם ה-Access Token שלו, כמו לדוגמא explorer.exe האהוב), התהליך יהיה:

1. למצוא תהליך שהורץ ע"י המשתמש הדומייני
2. לשכפל את הטוקן שלו (באמצעות OpenProcess ואח"כ DuplicateTokenEx)
3. לייצר תהליך חדש (לדוגמא cmd.exe, באמצעות CreateProcessAsUser) עם הטוקן הזה

וכעת יש לנו **cmd.exe** המזוהה לחלוטין כאותו משתמש ובעל אותם הרשאות המאפשרות לו לזוז בתוך הסביבה הדומיינית ולהתפרץ לשרתים נוספים!

זה עלול להיות קצת מסובך אז הכנתי לכם דוגמה קטנה: כתבתי תוכנה קטנה שמשכפלת טוקנים מתהליך נבחר, ראו דוגמא שהרצתי מתוך cmd.exe שרץ תחת SYSTEM (ומזכיר: ברצוני לעבור למשתמש אחר על מנת לגנוב את הזהות שלו ולעשות שימוש בהרשאות שלו):



שימו לב כיצד הצלחתי לעבור ממשתמש SYSTEM למשתמש אחר וכעת אני יכול לבצע פעולות בשמו תחת ההרשאות שלו (!). הקוד זמין לכם כאן.

סיכום

במאמר לקחנו תרחישים מהיומיום (פתיחת קובץ, ביצוע לוגין, התחזות לגיטימית) והסתכלנו מתחת למנוע של Windows להבין מה הגורמים שמאפשרים את כל הקסם הזה ואיך הם עובדים. למדנו את הנושאים הבאים:

1. מנגנון ה-**SRM** שנכנס לפעולה על כל בקשה הנוגעת באובייקט מאובטח במערכת ההפעלה ומחליט האם לאשר את פעולת הגישה או לא
2. מה הם מאפייני האבטחה של כל אובייקט מאובטח, תהליך ומשתמש במערכת (ע"ב מה המנגנון מקבל את ההחלטה): **User/Process Access Token** (**SID**, **Groups**, Integrity & Impersonation Level Session, **Privileges** and **Security Descriptors** (**SACL**, **DACL**)
3. הדגמנו כיצד ניתן לנטר לוגים על פתיחה של קובץ רגיש וכיצד ניתן לחסום גישה מעין זו באמצעות סקריפט המשנה את הגדרות ה-Security Descriptor

4. הדגמנו את ההשלכה של הרשאות של תהליכים: תחת משתמשים שונים ותחת Integrity Levels שונים ואיך תהליך שיצר קובץ כבר לא מורשה לגשת אליו
5. הדגמנו איך **Integrity Levels** מאפשרים להכיל התפרצות של תוקף מהאינטרנט לתוך התהליך של הדפדפן
6. איך **UAC** עובד מאחורי הקלעים ומי קיבל פטור ממנו
7. מהן הרשאות של תהליכים ואיזו הרשאה לדוגמא חשובה כאשר נרצה להתחזות
8. למה מיקרוסופט איפשרו פיצ'ר כל כך מסוכן של התחזות **(Impresonation)**, מה השימוש הלגיטימי בו וכיצד כתוקפים אנחנו יכולים להשתמש בו כדי להעלות הרשאות או להתפרץ לדומיין. כתבנו תוכנה המדגימה זאת
9. איך **Sessions**, **Windows Stations** ו-**Desktops** באים לידי ביטוי בבידוד אבטחתי של משתמשים ותהליכים שונים ובבד בבד מאפשרים לכמה משתמשים מרוחקים להתחבר ב-RDP במקביל ולעבוד בפרטיות.
10. מי הם משתמשי ברירת המחדל ואיך משתמשים בהם ובאחרים לטובת בידוד אבטחתי והכלה במקרה של פריצה

מקווה שהצלחתי לספק לכם תמונה מלאה והבנה על ה-Internals של המנגנונים הנ"ל, כעת יש לכם את כל הבסיס הנדרש כדי להגן באמצעותם או לעקוף אותם ;) כל שאבקש בתמורה הוא שתפיצו הבשורה ותתרמו מפי עטכם ושכלכם בחזרה לטובת קידום עם ומדינת ישראל.

זה גם המקום להודות ל**יהונתן אלקבס** ו**אפיק קסטיאל** על הביקורת עמיתים האיכותית וההארות המעולות.

ישנם נושאים נוספים שלא הספקנו לכסות (אולי במאמר הבא) כמו אותנטיקציה מעל RPC וחתימות באמצעות תעודות של תהליכים במערכת ועוד - מוזמנים להרחיב בעצמכם עד אז.

כשלב הבא ממליץ בחום לצלול לעומק בסדרת המאמרים של יהונתן על Lateral Movement: חלק א' וחלק ב', על הסדרה של Sonny, ולאחר מכן להכיר את lsass היטב וכיצד ניתן לתקוף אותו. ובעיקר לקרוא הרבה מהדוקומנטציה של מיקרוסופט.

על המחבר

שקד (שקדי) אילן, Offensive Security Researcher and Developer (כשאני לא בפרודקט), מוזמנים לפנות אלי לכל שאלה או הבהרה.