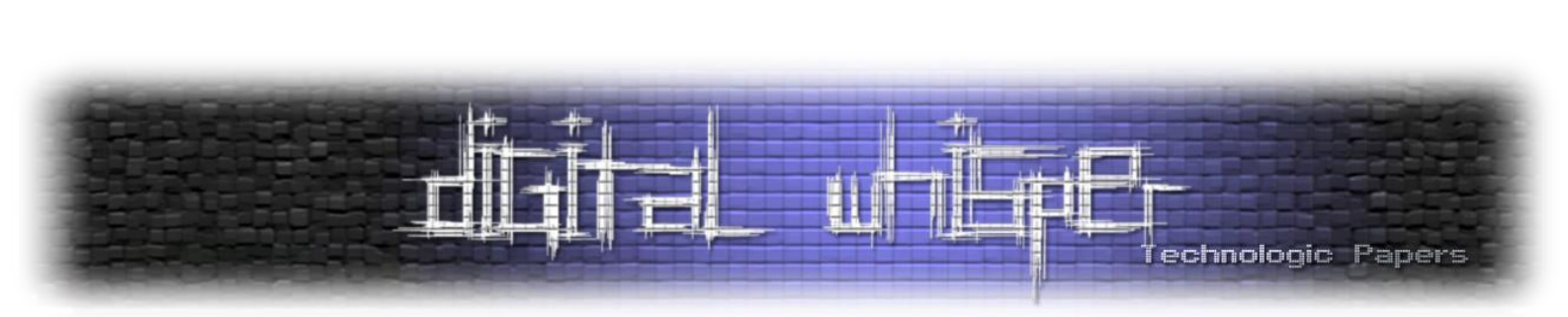


Windows Lateral Movement from Scratch Part 2 - One ACE to Rule Them All

מאת יהונתן אלקבס

הקדמה

הקונספט של רשימת מוזמנים וסלקציה עובד לא רע בעולם האמיתי. כמעט בכל אירוע חברתי גדול מבוסס מוזמנים נוכל לראות שעל מנת לעבור הרשאה ולקבל גישה לאירוע נצטרך לעמוד ב-2 תנאי סף - אחת, להיות הבעלים של כרטיס כניסה, ושתיים, הימצאות שֵׁמִית ברשימת המוזמנים.

מבחינת אופרציה, בעוד שישנן מערכות שמנפיקות את כרטיסי הכניסה (מבוססות web ו\או קופה רושמת), ישנן פלטפורמות הממוקמות בפרימטר בין השטח הציבורי לשטח האירוע שמאשרות את זהותו של בעל הכרטיס (לרוב את המשימה הנ"ל ימלא אדם פיזי). בצורה הזו, מנהלי האירוע יכולים לאשר ולחסום מוזמנים. מנגנוני **Whitelist** ו-**Blacklist** קלאסיים.

עקב הארכיטקטורה הפשוטה של כלל המנגנון, ברור לכל כי לא נדרש תחכום רב במיוחד בשביל לעקוף חלקים ממנו. ככל שנרצה להטיב עם אבטחת המכניזם ולהוסיף פתרונות הגנה מפצים כך יגדל משמעותית סך המשאבים הנדרשים לכלל המשימה.

למרות מגרעותיו, הרעיון הבסיסי והאבסטרקטי של רשימות בקרת גישה (Access Control List בלעז) מאפשר לממש פעולות מורכבות כגון מתן גישה וחסימת משתמשים במערכות גדולות בעלות נמוכה. לכן אין זה מפתיע שרשימות ACLs מצאו את דרכן גם אל ליבת מערכת ההפעלה והדומיין.

איך כל הפילוסופיה הזו רלוונטית לתזוזה רוחבית? בשביל להבין את ההקשר לסביבת Windows אנחנו בסך הכל צריכים לשנות את הטרמינולוגיה ולהוסיף מספר מילים ללקסיקון שלנו.

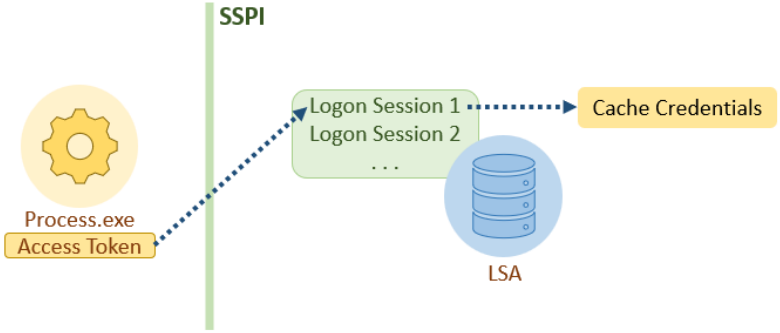
ישנם אובייקטים במערכת ההפעלה שכולם יכולים לגשת אליהם ויש כאלה שרק חלק מאוד ספציפי של משתמשים יכול ליצור עימם אינטרקציה. **אם המהות של Access tokens היא כרטיס הכניסה שצריך להציג בכניסה למסיבה הרי ש-DACL מאפיינת את רשימת הסלקציה ו-ACEs מייצגים את שמות המוזמנים.** כלומר במילים פשוטות, רק במידה והנתונים שמוטבעים ב-Token שלנו מתאימים לאלו שברשומת הגישה באובייקט נוכל לקבל הרשאה ולהשתתף במסיבה הפרעית של Windows.

בפרקים הקודמים של השיר שלנו

במאמר הקודם, Windows Lateral Movement from Scratch part 1 - Access Token Granted Right (כן אני מסכים, זה ללא ספק שם ארוך מדי) התחלנו לבנות את בסיס הידע שלנו על תזוזה רשתית מאבני הביניין שמרכיבות אותו. המאמר עסק ברובו ברכיבים ליבתיים של מערכת ההפעלה ובמודל האימות של Windows. סקרנו **6 מנגנוני אימות שונים** (פיזי, מרוחק, לוקאלי, דומייני, אינטראקטיבי ולא אינטראקטיבי) אשר חיים זה לצד זה במערכת ההפעלה וכיצד אלו משפיעים על יצירת ה-Logon sessions של משתמשים.

כוכב המאמר היה אובייקט ה-Access Token אשר נוצר על ידי תשתית ה-SRM בקרנל בזמן אימות אינטראקטיבי ועובר אל תשתית ה-LSA בסביבת ה-user-mode. באמצעותו, בכל פעם שתהליך משתמש ניגש לבצע פעולה אל מול אובייקט אחר, מערכת ההפעלה יודעת לפרסר את השדות בתוכו ולהבין את **קישוריות הגישה** (קרי, ה-security context) של session המשתמש ולהתיר או למנוע את הגישה.

כמו כן, נקודה חשובה שהבנו היא כי אימות רשתי ו-SSO מתאפשרים לאחר אימות אינטראקטיבי מכיוון שהמבנה של logon session מכיל מצביע אל ה-credentials של המשתמש. זו גם שורש הסיבה מדוע שימוש פחות תמים במנגנון יכול לזכות תוקף בקריאה של אותם credentials.



את המאמר עם השם הארוך סיימנו בהסבר על השיטות השונות בהן מבוצע **Impersonation** (התחזות בתרגום ישיר ושבור) הן ברמה התפעולית (הלגיטימית) לשם ניהול משאבים בדומיין והן ברמה ההתקפית לשם התפשטות רוחבית. בשביל כך, עברנו על קוד המקור של נוזקת **meterpreter** וראינו כיצד הכלי מצליח להזריק את ה-access token של משתמש אחר ל-thread של ה-session שלו. בצורה דומה, התבוננו מה קורה מאחורי הקלעים כאשר הנוזקה מעוניינת לפתוח תהליך חדש עם security context שונה.

עם כל הכבוד למאמר הקודם, ברור לכל כי תזוזה רוחבית באמצעות impersonation היא איך לומר במילים עדינות - לא שיא הפופולאריות. אולי רעיון נחמד ששווה לשחק איתו ברמה הלוקאלית בעת שמשיגים אחיזה על מכונה חדשה בדומיין אבל בבירור לא ה-bread & butter של האופרציה.

אז מדוע בכל זאת פתחנו (אני) דווקא עם הנושא הנ"ל? ובכן, מכניזם הגישה אל אובייקטים הוא אבן ביניין משמעותית מהבסיס של תזוזה רשתית והוא מספק לנו כַּר-פּוֹרֶה כשנעבור לדון ממש עוד מעט בנושאים כמו כוכבי המאמר הנוכחי - **DACL** (Discretionary Access Control List) ו-**ACEs** (Access Control Entries) ומה

מה שמרכיב אותם. כמו כן, הבנה טובה של אלו תוביל אותנו ישירות אל הצורך בקיום של מנגנוני הגנה נוספים כדוגמת Mandatory Integrity Control ו-User Access Control בשביל לאפשר שליטה ברזולוציה גבוהה יותר למשאבים רגישים.

All in all, במאמר הקרוב אנחנו הולכים להתקדם משמעותית בסך ההבנה שלנו בתזוזה רוחבית ולגלות כיצד עובד המנגנון שמאפשר מתקפות מבוססות ACLs אשר מרכיבות נתיבי התפשטות רשתיים בכלים כדוגמת **PowerView** ו-**BloodHound** ומדוע Microsoft בחרו לממש אותו כך.

הערה - לאלו שעדיין לא קראו את המאמר הקודם אני ממליץ בחום להתחיל ממנו. הוא מקור רפרנס מעולה וגם רפרוף חפוז עם דגש על האיורים בזמן הקפה של הבוקר עתיד לעשות את העבודה.

מאיפה הכל מתחיל

לקרנל משימות רבות - התניית גישה בין חומרה לתוכנה, ניהול משאבים (כדוגמת RAM, processes, files, threads וכד') וניהול הגישה אל אותם משאבים. בשביל לבצע את המשימות המורכבות הללו, הקרנל מסתמכת על שכבת אבסטרקציה שהיא בעצמה יצרה - מספר סוגים של אובייקטים אותם היא מנהלת בהתאם לצרכים של מערכת ההפעלה.

עם זאת, חשוב להבין שלא כל המידע ומבני הנתונים במ"ה מיוצגים על ידי אובייקטים. רק מידע שנדרש להגדיר, לשתף, להגן או לחשוף אל אפליקציות user-mode (על ידי System Services) נסגר בתוך אובייקט.

ההבדל העיקרי בין שם התואר הנכסף "אובייקט" אל "סתם מבנה נתונים" הוא העובדה שלא ניתן לעבוד ישירות על אובייקטים. מה הכוונה? בעוד שעל מנת לקבל מידע שמאוחסן במבנה נתונים בסך הכל צריכים לדעת את הכתובת שלו, על מנת לגשת אל מידע שמאוחסן באובייקט נדרש לקרוא לשירות שאחראי על משיכת המידע ממנו. זה אומנם נשמע כמו התפלספות מיותרת אבל השוני הנ"ל הוא מה שמבדיל בין האובייקטים לקוד שאשכרה עושה בהם שימוש. מדובר בטכניקה מוכרת בעולם התכנות שמספקת גמישות באימפלמנטציה של אובייקטים לאורך זמן וגרסאות.

על האקט של השגת גישה על ידי אובייקט אחד לאובייקט אחר נאמר כי האובייקט הראשון השיג **handle** לאובייקט השני. כלומר, בשפה פשוטה handle הוא רפרנס למאורע של אובייקט.

ב-Windows ישנם 3 קטגוריות של אובייקטים:

- **User** objects
- **GDI** objects
- **Kernel** objects

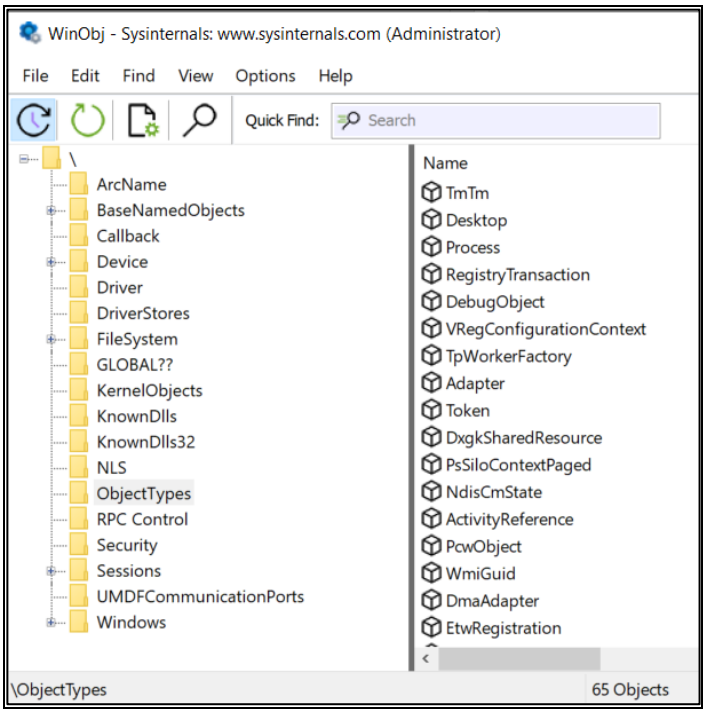
הסוג הראשון, במימד המשתמש, נועד לספק תמיכה לניהול מערכת ההפעלה ברמת המשתמש והם גמישים לשימוש. הממשק של מ"ה מאפשר רק **handle אחת לכל אובייקט** כזה ותהליכים לא יכולים לרשת או לשכפל את ה-handles עבור גישה אל user objects. עם זאת, אותם handles הם פומביים לכל התהליכים. קרי, כל תהליך יכול להשתמש ב-handle במידה ויש לו את יכולות הגישה (ה-security context) המתאימים. כל session של משתמש מוגבל למספר ה-handles שהוא יכול לנהל רק עקב אילוצי זיכרון.

הסוג השני (שתכל'ס לא כזה מעניין אותנו כרגע), הוא מסוג GDI (graphics device interface) ותומך בפעולות ממשק משתמש. מדובר באובייקטים פרטיים, כלומר רק התהליך שיצר את האובייקט יכול לעשות שימוש ב-handle שלו.

והסוג השלישי, וכנראה הכי חשוב, הם אובייקטים של הליבה עצמה. כל handle לאובייקט משוייכת לתהליך, כלומר כל תהליך מחויב או ליצור אחת או לפתוח אחת בעצמו בשביל לגשת לאובייקט קרנלי.

בשונה משני הסוגים שלפניו, **מספר רב של יישויות יכולות לגשת לאותו kernel object** (גם אם הוא נוצר על ידי תהליך אחר) בהינתן שאותה יישות יודעת את השם של האובייקט ויש ברשותה את ה- security access המתאים לאובייקט. מנגנון זה קורה באמצעות הנפקה של טבלת חוקות גישה (**Access Rights** בלעז) שמי שיצר את האובייקט הגדיר בזמן יצירתו.

נוכל להסתכל על כלל האובייקטים השונים באמצעות WinObj.exe מאת SysInternal:



בתוך כל אלו ניתן למצוא אובייקטים כדוגמת "Process" אותו הקרנל מצמידה לתהליכים, "Session" אשר על פיו ניתן לנהל סטאטוס חיבוריות של משתמשים במ"ה ואת "Token" הלא הוא כוכב המאמר הקודם שלנו שבאמצעותו המערכת מבינה את ה-security context של "process" או "thread".

כאמור, באמצעות Windows ניתן לבנות מארג של מכונות ומשתמשים עם חוקים מוגדרים מראש (זהו הדומיין) ולכן לסוגים האלו יתווספו קטגוריה שלמה של אובייקטים הקשורים ל-Active Directory (אותם הצגנו בתחילת המאמר הקודם). עם כל אלו אנחנו יכולים להתחיל לבנות את הסביבה הרשתית ולהכיל עליהם רשימה לניהול גישה. Indeed, one big party.

כיצד ניתן לקבל הזמנה למסיבה ב-Windows?

אובייקט מוגדר כ"אובייקט מאובטח" אם קיים ב-header שלו שדה בשם Security descriptor **שלפיו ניתן להגדיר את חוקות הגישה לאובייקט**. השדה מכיל מבנה נתונים בינארי שתמיד מכיל תחילית של control bits, את ה-SID של הבעלים של האובייקט וה-SID של הקבוצה הראשית של האובייקט.

```
typedef struct _SECURITY_DESCRIPTOR {
  BYTE                    Revision;
  BYTE                    Sbz1;
  SECURITY_DESCRIPTOR_CONTROL Control;
  PSID                    Owner;
  PSID                    Group;
  PACL                    Sacl;
  PACL                    Dacl;
} SECURITY_DESCRIPTOR, *PISECURITY_DESCRIPTOR;
```

כמו כן, כפי שניתן לראות בחתימה שלו, יהיו גם מבני נתונים בשם **discretionary access control list** (DACL) ומחזיק רשימות של **access control entries** (ACEs). ACE מייצג את זהות היישויות (ה-principals, או ליתר דיוק ה-trustees_בלשון הדוקומנטציה) ביחד עם ההרשאות שלהן ו-DACL מצייג את רמת האמון של בעל האובייקט באותן יישויות.

<u>בכל פעולה בין אובייקטים ה-DACL נבדק</u> - האם ה-ACE שמייצג את זהות המשתמש (כאמור, ה-SID שלו או של הקבוצות שלו) נמצא ב-DACL וכיצד צריך להגניש (או לא) עבורו את האובייקט. שני מצבי הקצה של DACL יוכלו ללמד אותנו על אופן הפעולה שלו בצורה אינטואיטיבית:

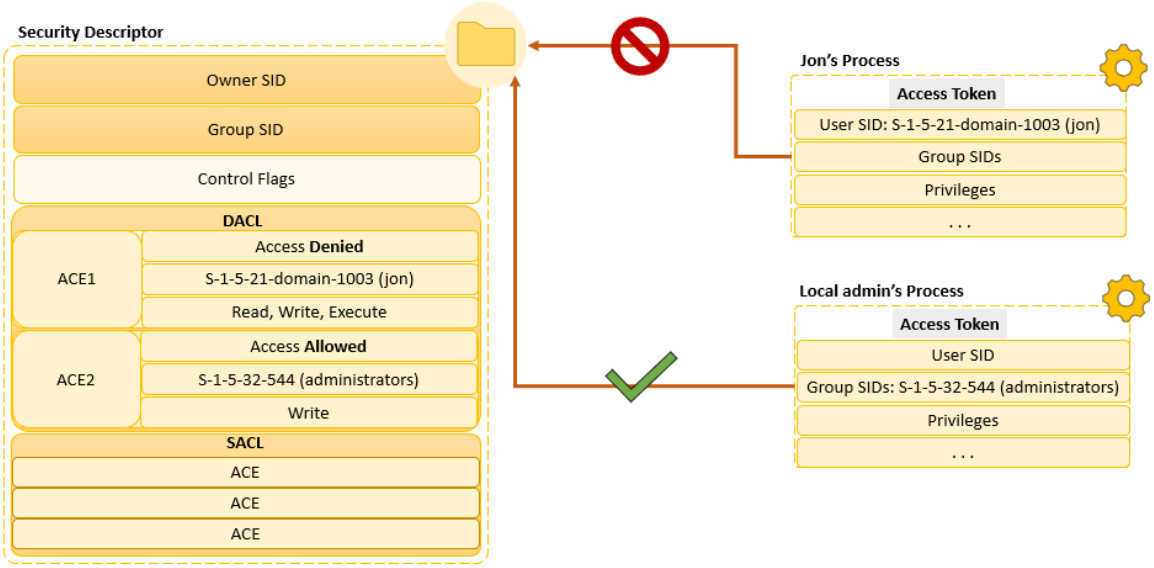
- **כאשר לאובייקט אין בכלל את שדה DACL** {ערך NULL} - כולם (ליטרלי קבוצת Everyone) יוכלו לבצע אינטרקציה עם האובייקט. באנלוגיית המסיבה שפתחנו עימה, אם עשינו אירוע ואין לנו רשימת מוזמנים - כל מי שיציג כרטיס כניסה יוכל להיכנס. מדובר במצב מאוד לא מאובטח כי אם נמשיך עם אותה אנלוגיה אז לא מדובר רק בהרשאה להיכנס למסיבה אלא גם בהרשאה של לקיחת בעלות על המסיבה, חסימה של אנשים אחרים ואפילו פיטורין של כלל העובדים בתוכה (מחיקה של האובייקט).
- **כאשר לאובייקט קיים שדה DACL אבל הוא <u>ריק</u>** - אף אחד לא יוכל לגשת אליו. עשינו מסיבה אבל כל אורח שירצה להיכנס לא יוכל לקבל גישה כי אין את השם שלו ברשימת מוזמנים.

בדומה למנגנון של FW סטנדרטי, כאשר תהליך רץ ב-security context של משתמש מסויים וינסה לפנות אל משאב בדומיין, מערכת ההפעלה תעבור אחד אחד על רשימת ה-DACL ותשווה את הפרטים ב- Access

token של התהליך אל רשומות ACE **לפי הסדר עד שתמצא רשומה המתאימה** לפרטים של ה-trustee (המשתמש, הקבוצה או ה-logon session) ב-Token ואז תפעל לפיה.

כלומר, במידה ורשומת "ACE deny X" תופיע לפני "ACE allow X" אזי מערכת ההפעלה תשלול את הגישה מ-"X" מכיוון שהופיעה לו קודם רשומת חסימה.

כתבתי כבר הרבה מילים אז נעבור לאיור שממחיש את כל הסיפור יותר טוב ממני:



במידה ויש לנו תיקיית share ושני תהליכים (או threads) ינסו לגשת אליה, מערכת ההפעלה תפרסר את השדות ב-primary token (שכאמור מדובר בסך הכל בפוינטר אל ה-access token של המשתמש שהריץ את התהליך) ותשווה את הערכים שנמצאים בו למערך של ACEs שנמצאים בתוך המערך DACL כחלק ממערך ה-security descriptor של התיקייה עד אשר תמצא ACE עם שדה SID שמתאים בפרטיו לזה של הטוקן.

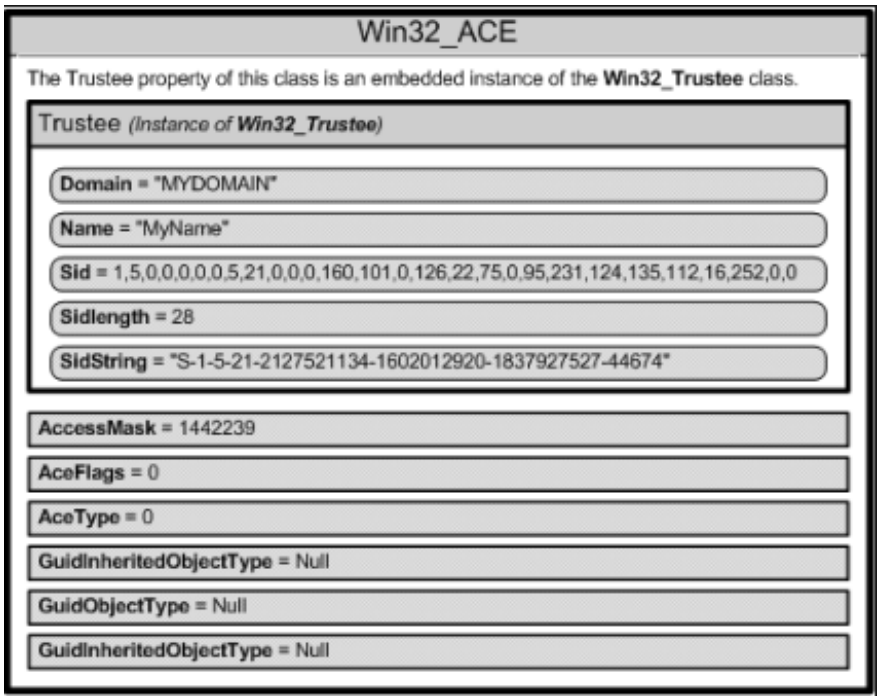
כפי שניתן היה לראות גם ממבנה הנתונים בתחילת הפרק וגם כעת מהאיור, למעשה ישנם 2 סוגים של רשימות המאכלסות ACEs במבנה נתונים שלהן - DACL ו-**SASL**.

את הראשונה כבר הצגנו היטב אך מה פשר האחרת? **System access control list** (SASL) מאפשרת לייצר לוגים לנסיונות גישה מוצלחת (ולא מוצלחת) אל האובייקט ולג'נרט להם security event log. היא מבצעת זאת על ידי שימוש ב-ACEs אשר מייצגים את ה-trustee עצמם וסט דגלים שלפיהם המערכת מפיקה הודעות audit.

בדומה ל-DACL, אם רשימת SASL לא קיימת (תוגדר כ-null) לא יתרחשו שום פעולות תיעוד. אני אציג דברים מגניבים שניתן לעשות עם SACL במאמר **Malware-less Persistence (Done) Right** שיפורסם במסגרת הגליון הבא.

הערה - חשוב לשים את האצבע הוירטואלית שלנו על הנקודה כי למרות ש-DACL מנהל את הגישה לאובייקט הוא לא מנהל את הגישה אל ה-SASL. הפנייה אליה מתנהלת באמצעות הרשאת גישה אחרת שנמצאת בתוך המבנה של ה-SASL עצמו.

למעשה אנחנו יכולים לעשות zoom in נוסף על כל הנושא ולראות את המבנה שמאפיין את האובייקט של ACE עצמו ישירות מהדוקומנטציה של Microsoft:



כל אובייקט ACE מכיל יישות trustee בתוכו שמזהה באופן חד ערכי את המשתמש או הקבוצה ומאפיין **AccessMask** ביחד עם **AceType** שמסמלים אילו פעולות אותו trustee יכול לבצע ברמת האובייקט.

הערך של AccessMask נקבע על פי הסכום של כל פעולה, למשל אם הערך של כתיבת attributes לאובייקט הוא 256, קריאה 16 וכתיבה 8 אזי הערך של AccessMask יהיה 280. הערך של AceType מסמל האם אותה גישה מותרת (0) או לא (1). יש לנושא עוד דקויות אבל לשם ההבנה שלנו כרגע זה בהחלט מספיק.

הערה - בלא מעט מקורות מידע, פורמאלים ופחות פורמאלים, המונח ACL (Access Control List) מתייחס אל DACL ו-SACL כאחד. עקב השוני הניכר בין שניהם אני לא אוהב את שיטת הכתיב הנ"ל ולכן במהלך המאמר אני הולך להקפיד לרשום במפורש כאשר הדיון עוסק ב-DACL או SACL ולהימנע ככל הניתן מכתיבת ACL לחוד אלא למקרים בהם מדובר על הנושא בכלליות.

עוד קצת תיאוריה ועוברים לחלק הפרקטי

בסך הכל ישנם 2 סוגים של ACEs:

אונו. **Generic ACEs**

דוס. **Object-specific ACEs**

כאשר כל סוג מחולק בתוכו אל 3 תתי סוגים - שניים עבור רשומות ב-DACL ואחד עבור SACL. הסוג הראשון, Generic ACEs, מתייחס אל ACEs שנתמכים על ידי secure objects:

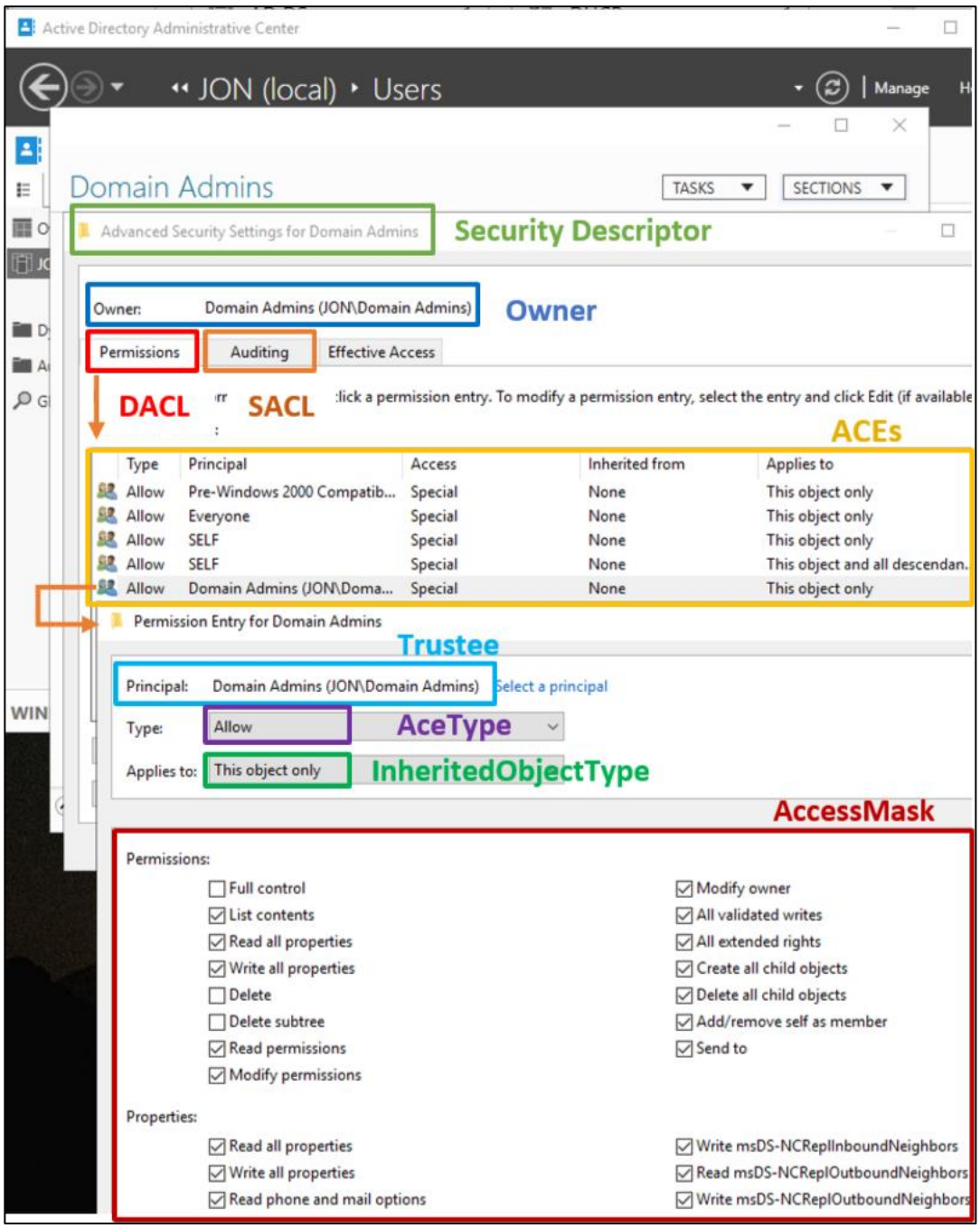
1. Access-denied
2. Access-allowed
3. System-audit

והסוג השני, object specific ACEs, מכיל שלושה ACEs נוספים שנוצרו במיוחד עבור directory service (DS):

4. Access-denied object
5. Access-allowed object
6. System-audit object

כלומר, after all **ישנם 6 סוגים של ACEs**. לכל אחד מאלו צפויים להצטרף סט פעולות כדוגמת List, Read, Write, Delete, Modify וכד'. ת'אמת אני נוטה להסכים עם הטענה ששתי הקבוצות נראות די דומה (תודה Microsoft) אבל הבדל העיקרי הוא שקבוצת ה-object specific מכילה צמד GUIDs (**ObjectType** ו-**InheritedObjectType**) נוספים שבעצם מהווים שדות אקסטרה שאפשר לברור ולקטלג מדיניות ACL לפיהם עבור האובייקטים עצמם ב-AD. בנוסף, חלק מהאובייקטים ב-AD ירשו את ה-ACEs שיש לאובייקט האב שלהם וחלק לא.

אם נפתח את הממשק הגרפי של הגדרת האובייקטים בדומיין דרך ה-ADUC ב-DC נוכל לראות כיצד הכל מתורגם לרמה הפרקטית. לשם הדוגמה נסתכל על אילו הרשאות יש לקבוצת Domain Admins על עצמה:



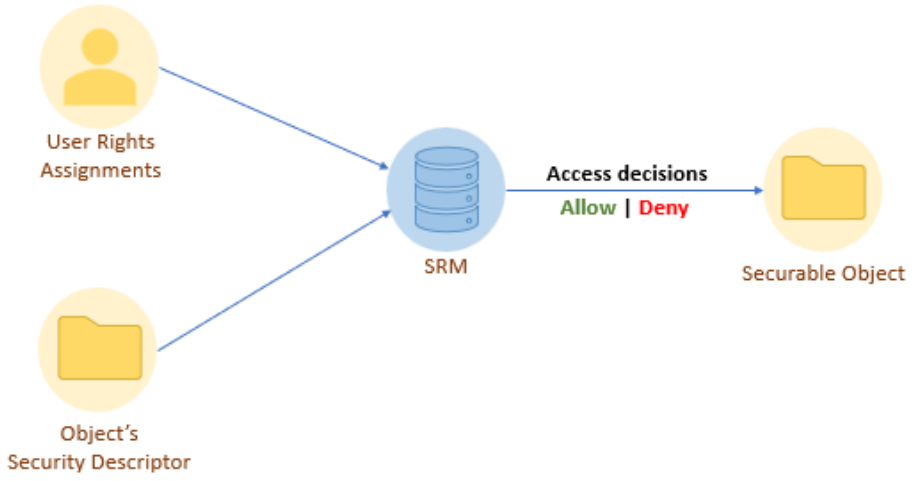
אפשר לראות שלמרות שמדובר בהרשאות של הקבוצה עצמה, לחברים בה אין שליטה מלאה עליה (ובצדק!). אלו ההגדרות הדיפולטיביות בדומיין. דרך הממשק הנ"ל (וממשקים נוספים) כל ארגון יכול לבחור כיצד הוא בוחר לנהל את האובייקטים בדומיין שלו באמצעות ACEs שמוצמדים אליהם.

לשם הדוגמה, אני יכול להחליט שבגלל שבדומיין שאני מנהל יש מספר רב של DAs אני לא מעוניין שהחברים בקבוצת DA יהיו מסוגלים לשנות את ה-Owner של הקבוצה.

הכל בחיים זה סדר

נושא נוסף שחשוב להדגיש הוא הסדר הקנוני (**Canonical order**) של ה-ACEs.

כזכור מהמאמר הקודם, גם במערכת ההפעלה וגם בסביבת AD, החלטות על מתן גישה לאובייקטים נעשות על ידי ה-Security Reference Monitor (**SRM**) ב-kernel-mode החל מ-Windows 2000. הדרך בה נתינת (או נאסרת) הגישה נקבעת על ידי ה-SRM בכך שהיא מזהה שהאובייקט שמנסים לגשת אליו הוא secure object בזכות ההימצאות של שדה ה-security descriptor.

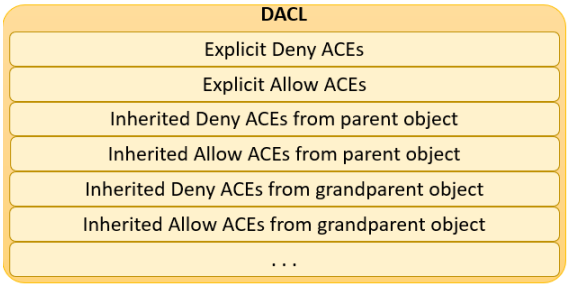


ה-SRM מפרסרת את מערך ה-DACL שב-security descriptor ומשווה את התוכן של כל ACE בלולאה תוכנתית אל מול ה-SIDs שב-Access Token של הבקשה. Simple as that.

מכיוון שה-SRM **מפסיקה לבדוק ACEs נוספים** ברגע שהגישה המבוקשת ניתנת או נדחית **במפורש**, <u>הסדר בו ה-ACEs מופיעים ב-DACL הוא **קריטי**</u>. Microsoft מתארים את הסדר הבא:

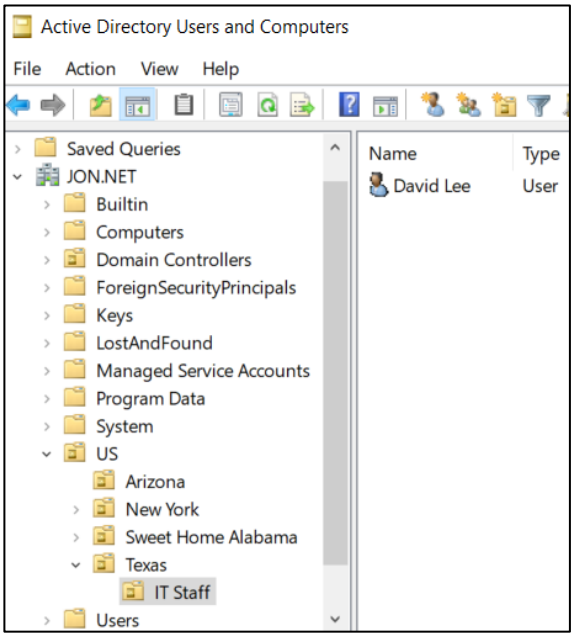
1. כל ה-Explicit ACEs ממוקמים בקבוצה לפני כל ACEs שעברו בירושה לאובייקט.
2. בתוך הקבוצה של ה-Explicit ACEs , סוג Access-denied מופיע לפני סוג Access-allowed.
3. ה-ACEs שעברו בירושה לאובייקט מסודרים ברמות לפי הסדר בו הורשו. כלומר גישה שהגיעה מהורשה של אובייקט האב ממוקמת לפני גישה שהניתנה בהורשה מאובייקט הסב.
4. באופן דומה לסעיף 2, בכל רמה של ACEs בירושה, קודם כל ממקמים את ה-ACEs שדוחים גישה ורק אז את ה-ACEs שנותנים גישה.

אם לסכם את הסדר הקנוני באיור קוהרנטי:



יש לא מעט היגיון בסדר הנ"ל. על ידי השמה של ACEs האוסרים גישה לפני אלו המקנים גישה, ניתן לקנפג blacklist ספציפית לכל מקרה (כדוגמת דחיית גישה של משתמש ספציפי לאובייקט בעוד הגישה מאושרת לקבוצה אליה הוא שייך). כמו כן, לפי כללי ההורשה של ACEs, כל אובייקט קודם כל מחליט על עצמו ורק אז לפי מה שהגדיר אביו. בצורה הזו אם יצרנו קובץ בתוך תיקייה, אנחנו אלו שאחראים על מי יכול לגשת אליו ולא מי שיצר את התיקייה. אבל יש גם מחיר אבטחתי לסדר הזה.

בשביל להבין את הבעיה בסדר הנ"ל נסתכל על אובייקט User לדוגמה מהדומיין:



אובייקט המשתמש David Lee עתיד לרשת ACEs מכל האובייקטים שמעליו, כולל האובייקט של הדומיין (JON.NET), האובייקט אב (IT staff), האובייקט סב (Texas) והאובייקט סבא רבא (US). מכל ה-OUs האלו הוא צפוי לקבל ACEs בירושה ומכיוון שאובייקטים שקרובים יותר בעץ המשפחתי מקבלים תיעדוף אז ACEs שאותו משתמש יקבל מ-IT staff ידרסו את אלו שהוא יקבל מ-Texas.

כלומר קיימת פה לוגיקה בעייתית מהיסוד. אם בתור מנהל הדומיין אני רוצה לאסור על כל המשתמשים מטקסס שלא יוכלו לגשת למשאבי share של החברה באמצעות ACE **Deny**, שום דבר לא מונע ממנהל DC אחר (או מתוקף עם הרשאות גבוהות) ליצור לאותו OU אובייקט אב עם ACE **Allow** ובכך לעקוף אותי.

בהיגיון דומה נציג במאמר המשך כיצד ניתן ליטרלי **להחביא אובייקטים** שלמים מהצד המגן ולהשתמש בהם לשרידות ללא הגבלה וללא שריטה אחת של הדיסק. גם משתמש שרץ בהרשאות SYSTEM לא יוכל לכתוב שאילתת LDAP שתמצא את האובייקט המוחבא. בהחלט תרחיש מרתק.

The DACL to my (Windows) heart

כאמור, שדה security descriptor מגדיר את חוקות הגישה לאובייקט ונראה בצורה הבאה:

```
typedef struct _SECURITY_DESCRIPTOR {
  BYTE                    Revision;
  BYTE                    Sbz1;
  SECURITY_DESCRIPTOR_CONTROL Control;
  PSID                    Owner;
  PSID                    Group;
  PACL                    Sacl;
  PACL                    Dacl;
} SECURITY_DESCRIPTOR, *PISECURITY_DESCRIPTOR;
```

מאפיין חשוב בתוכו הוא שדה ה-Control אשר מייצג תכונות ליבתיות באובייקט. נצלול לתוכו בשביל להבין את מלוא התמונה.

מבנה SECURITY_DESCRIPTOR_CONTROL מכיל 16 ביטים אשר כל אחד מהם מייצג אספקטים של אבטחת האובייקט כדוגמת מקרים פרטיים של ירושה. סה"כ ישנם 5 ערכים הנוגעים ל-DACL ולכן נתעמק עליהם עוד קצת:

- **SE_DACL_PRESENT** (0x0004) - מסמן אם ה-security descriptor מכיל DACL או לא. במידה והביט הנ"ל לא מוגדרת על ידי מי שיוצר את האובייקט (שווה ערך ל-NULL), כולם יוכלו לגשת, לשנות את האובייקט ובכללי לעשות מה שבא להם.
- **SE_DACL_DEFAULTED** (0x0008) - במידה ומי שיצר את האובייקט לא ציין את ערכי ה-DACL, האובייקט יקבל את ה-DACL לפי ה-Access token של מי שיצר אותו. נהוג לקרוא לבחור בשם "DACL דיפולטיבי".
- **SE_DACL_AUTO_INHERIT_REQ** (0x0100) - ה-DACL של האובייקט עם כל ה-ACEs *צריך להיות* מופץ אוטומטית בירושה לכל האובייקטים הבנים שלו.
- **SE_DACL_AUTO_INHERITED** (0x0400) - ה-DACL של האובייקט עם כל ה-ACEs מופץ אוטומטית בירושה לכל האובייקטים הבנים שלו.
- **SE_DACL_PROTECTED** (0x1000) - לא מאפשר את השינוי של שדה ה-DACL בעת ירושה של ACEs.

כל האובייקטים ב-AD חייבים להכיל ב-Security Descriptor שלהם את ה-SID של הבעלים (owner) שלהם. דיפולטיבית, אותו אובייקט מקבל הרשאת WriteDacl ו-RIGHT_READ_CONTROL. נתייחס ונסביר כל מקרה בצורה פרטנית בהמשך.

Windows Integrity Mechanism Design

טוב טוב עוד עיכוב קטן של תיאוריה חשובה ומתחילים עם החלק המעשי. למעשה ישנה עוד שכבה של בדיקות המתרחשת בזמן הגישה לאובייקט והיא ה-**Mandatory Integrity Control** (MIC). מדובר במנגנון נוסף ששולט בגישה אל אובייקטים מאובטחים אשר אחראי על הערכת יושרת הגישה עוד לפני שבודקים את המערך של DACL.

השאלה הראשונה שצריכה לרוץ לכולכם בראש כרגע היא - *WTFF למה אנחנו צריכים שכבה נוספת? האם הרעיון של בדיקת DACL לא מספיק??* ובכן, כן ולא. כמו שאמרנו בתחילת המאמר, רשימת סלקציה היא אחלה מנגנון אבל היא לא מספיקה במקרים שחשוב לנו מאוד טיב הבדיקה.

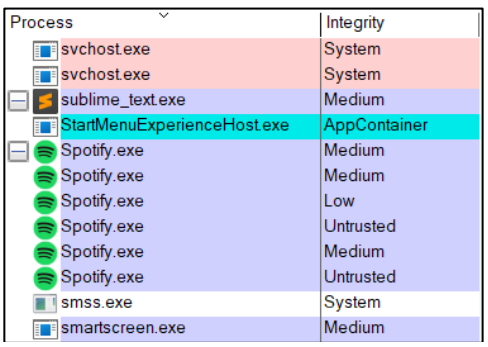
האבסטרקציה של DACL בהחלט מהווה את ליבת תהליך ההרשאות במערכת ההפעלה של Windows ואכן הרוב המכריע של כל בקשות הגישה יינתן או ידחה בהתבסס על בדיקות ה-DACL. MIC היא בסך הכל מכניזם אקסטרה שנוסף ב-Windows Vista ביחד עם **User Access Control** (UAC) בשביל לאפשר שליטה ברזולוציה גבוהה יותר למשאבים רגישים.

כל אובייקט מאובטח מחזיק ב-MANDATORY_INTEGRITY_LABEL בתוך מבנה ה-SACL שלו. השדה הנ"ל יכול לקבל אחת מתוך ארבעת רמות האמון (**Integrity Levels - IL** בלשון הדוקומנטציה) הבאות בשביל לייצג את האובייקט:

1. **Untrusted** : SECURITY_MANDATORY_UNTRUSTED_RID - 0x0000
2. **Low** : SECURITY_MANDATORY_LOW_RID - 0x1000
3. **Medium** : SECURITY_MANDATORY_MEDIUM_RID - 0x2000
4. **High** : SECURITY_MANDATORY_HIGH_RID - 0x3000
5. **System** : SECURITY_MANDATORY_SYSTEM_RID - 0x4000

הרמות יושרה האלו מייצגות את רמת האבטחה שניתנה לאובייקט והן מיוצגות במחרוזת ה-SID של כל אובייקט. לשם הדוגמה נוכל לדעת שהרמה משתמש עם SID של S-1-16-8192 היא Medium בגלל שערך ה-RID הוא 8192 בדצימל השווה אל 0x2000 באקסה.

נוכל להוסיף עמודה ולהסתכל על הרמות השונות באמצעות Process Explorer האהוב:

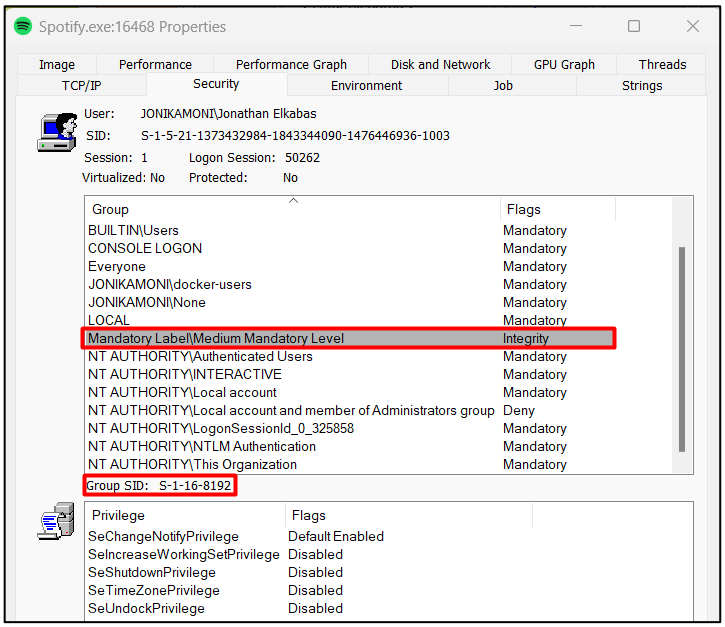


Process	Integrity
svchost.exe	System
svchost.exe	System
sublime_text.exe	Medium
StartMenuExperienceHost.exe	AppContainer
Spotify.exe	Medium
Spotify.exe	Medium
Spotify.exe	Low
Spotify.exe	Untrusted
Spotify.exe	Medium
Spotify.exe	Untrusted
smss.exe	System
smartscreen.exe	Medium

הערה - דיפולטיבית כל האובייקטים (כדוגמת קבצים וכד') מקבלים את התווית של Medium Integrity. תהליכים שרצים בתוך AppContainers יקבלו תווית של Low Integrity. בצורה הזו מתבצע חלק מהבידוד הנחשק של AppContainer שמונע מתהליכים הרצים בתוכו לבצע נזק לחלקים אחרים במ"ה.

על ידי חלוקה לרמות יושרה שונות מערכת ההפעלה יכולה **להגן על הנכסים שלה**. למעשה, כל הסיפור לשימוש בתווים הנ"ל הוא בשביל למנוע מתהליכים לא מהימנים לגשת ולבצע מניפולציות על תהליכים חשובים.

בצורה הזו למשל נוכל לראות בתמונה שתהליך Spotify פתח מספר תהליכים ברמות שונות - לכל רמה יש הרשאות שונות במערכת ההפעלה. סבירות גבוהה שהממשק משתמש דרכו אני גולש לחיפוש מוזיקה נפלאה של שושנה דמארי רץ ברמת יושרה נמוכה על מנת להגן על המערכת ממצב בו פגיעות בתוכנה תאפשר לתוקף רשתי לקרוא קבצים שיש לי במכונה (שכאמור נוצרים דיפולטיבית עם SECURITY_MANDATORY_MEDIUM_RID):



זה פחות או יותר השלב בו הרעיון של MIC אמור להפוך למובן יותר. ללא מנגנון Buffer נוסף בין תהליכים שהמשתמש מריץ אל מערך ה-DACL פגיעות בכל אפליקציית user-mode כדוגמת דפדפן explorer או chrome הייתה עוברת ללא בעיה את ה-DACL מכיוון שאותו תהליך רץ תחת ה-session של המשתמש עם ה-access token שלו **(!)**.

אז האם אפשר לסכם את כל הנושא בכך שתהליכים עם רמת יושרה נמוכים לא יכולים לגשת אל אובייקטים מאובטים עם רמת יושרה גבוהה יותר? ברור שלא, Microsoft אוהבים לסבך את החיים ולאפשר רמות קונפיגורציות שונות לכל מקרי הקצה (שזו בהחלט לא ביקורת רעה).

בזמן תהליך הבדיקה של ה-integrity level ה-SRM משווה את הרמה של התהליך לרמה של המשאב ומחליטה האם ללכת לכיוון של בדיקת DACL או לבטל את הגישה בהסתמך על ה-**Integrity Level Policy** שמוגדרת לאובייקט.

מדובר במבה נתונים המגדיר את חוקי הגישה לאובייקט ונמצא ב-SYSTEM_MANDATORY_LABEL_ACE:

- SYSTEM_MANDATORY_LABEL_**NO_WRITE**_UP - יישות עם רמת יושרה נמוכה יותר לא יכולה לכתוב לאובייקט.
- SYSTEM_MANDATORY_LABEL_**NO_READ**_UP - יישות עם רמת יושרה נמוכה יותר לא יכולה לקרוא מהאובייקט.
- SYSTEM_MANDATORY_LABEL_**NO_EXECUTE**_UP - יישות עם רמת יושרה נמוכה יותר לא יכולה להריץ את האובייקט.

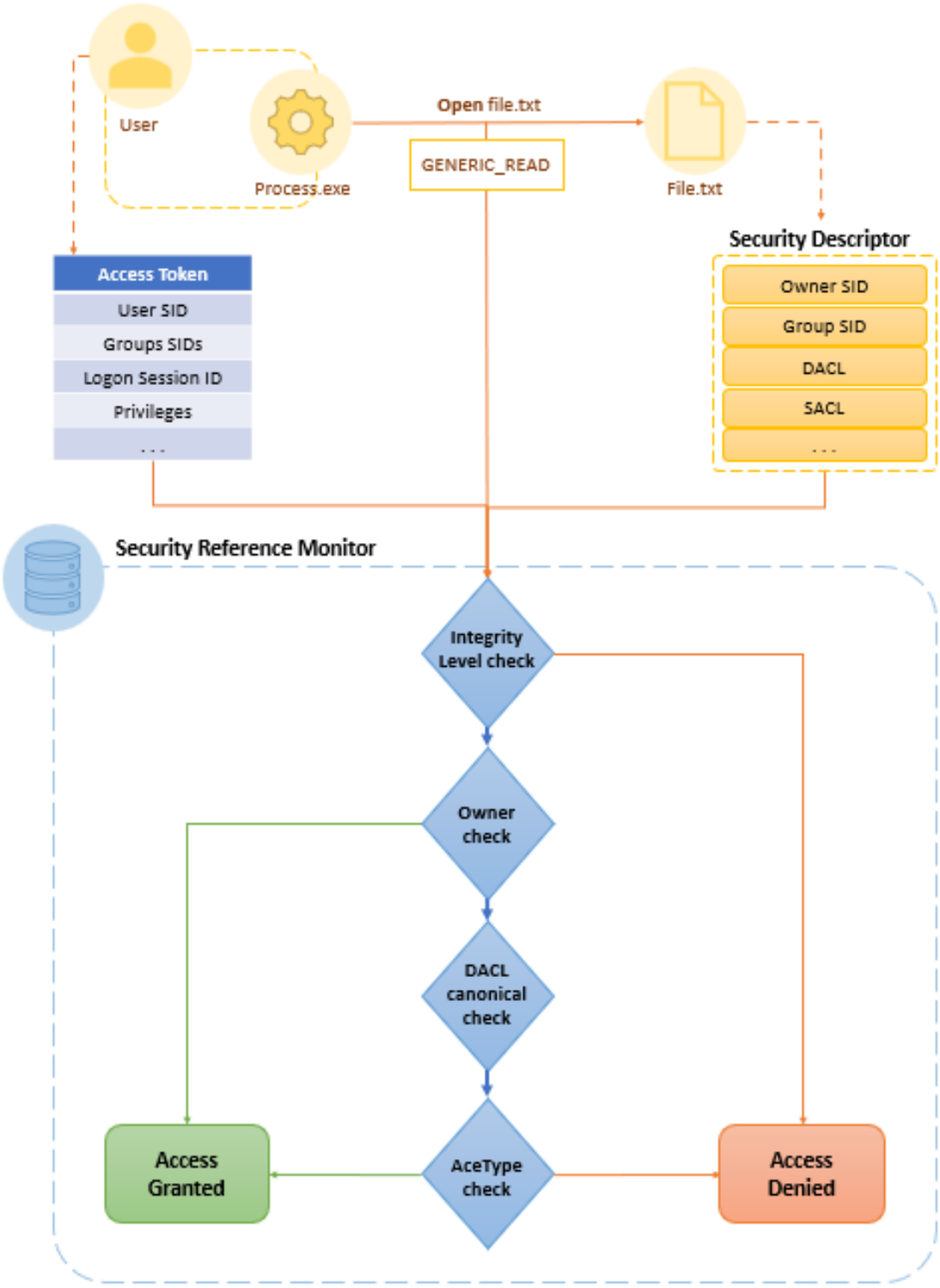
המבנה הנ"ל נמצא גם הוא ב-security descriptor תחת ה-Access Mask של ה-SACL.

ובכן, ניקח נשימה עמוקה ונסכם את כל הנושא של כיצד מערכת ההפעלה קובעת אם הגישה אל אובייקט מקבלת אור ירוק או אדום על ידי שלל הרכיבים השונים:

הדרך בה מערכת ה-**SRM** הקרנלית יודעת אם לאשר או לאסור גישה לאובייקט מאובטח היא על ידי בדיקה האם קיים שוני בין ה-**Integrity levels** של האובייקטים. במידה וכן, היא בודקת אילו פעולות אסורות על ידי מעבר על תוויות ה-System_mandatory_label_ace. במידה ולא, ואכן רמת היושרה של התהליך שניגש לאובייקט גדול מזו של האובייקט עצמו, המערכת בודקת האם המשתמש שמבקש את הגישה הוא ה-Owner של האובייקט. אם כן - מאושרת הגישה ישר.

במקרה ורמות היושרה מתאימות אך המשתמש הוא לא ה-Owner, ה-SRM מתחילה לפרסר ה-**DACL** ולהשוות כל ACE שבתוכו בסדר קנוני אל מול ה-**SIDs** (User SID \ Group SID) של מגיש הבקשה (אשר נמצא בתוך ה-**Access Token** שלו) בצמוד אל ה-**Access right** (כגון GENERIC_READ) עם ה-**Access Mask** שבתוך ה-ACE. אם נמצאה התאמה, שדה **AceType** יקבע אם הגישה מותרת או נאסרת.

בהחלט מדובר במקרה נוסף שאיור אחד יכול להפוך את כל המלל המסובך הזה להרבה יותר מובן:



ניהול אובייקטים באמצעות DWORD אחד

הערכים בשדה AccessMask (ביחד עם ערך 0 בשדה AceType) ב-ACE מייצגים אילו פעולות ניתן לבצע על האובייקט, כלומר קיימים **סוגים שונים** של ניהול אובייקטים. בלשון הדוקומנטציה הסוגים השונים נקראים access rights והם ניתנים לחלוקה על ידי 32 הסיביות של ערך ה-AccessMask עצמו:

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
GR	GW	GE	GA	Reserved			AS	Standard access rights								Object-specific access rights															

GR → Generic_Read
GW → Generic_Write
GE → Generic_Execute
GA → Generic_ALL
AS → Right to access SACL

לפי הפורמט ניתן לראות שישנם 3 סוגים של access rights:

Specific access rights - כשמם כן הם, מדובר בסט מאוד ספציפי של הרשאות עבור פעולות שמתאימות באופן ייחודי לאובייקט מסוים. הרשאות לדוגמה:

- FILE_READ_DATA - הרשאה לקרוא מידע מאובייקט של קובץ.
- FILE_APPEND_DATA - הרשאה לכתוב מידע לאובייקט של קובץ.
- KEY_CREATE_SUB_KEY - הרשאה ליצור subkey חדש ב-registry key.
- KEY_QUERY_VALUE - הרשאה לתשאל את הערך שב-registry key.

Standard access rights - קבוצה של הרשאות נפוצות ברמה אבסטרקטית המשמשות בדרך כלל סוגים שונים של אובייקטים. הרשאות אלו מאפשרות בקרת גישה רחבה יותר. הרשאות לדוגמה:

- READ_CONTROL - הרשאה לקרוא את ה-security descriptor של האובייקט.
- WRITE_DACL - הרשאה לערוך את ה-DACL עצמו של אותו אובייקט.
- WRITE_OWNER - לשנות את הבעלות של האובייקט.
- DELETE - הרשאה למחוק את האובייקט.

Generic access rights - טוב פה אנחנו כבר מדברים על רמה גבוהה יותר של אבסטרקציה והפשטה. הרשאות בקבוצה הזו הן משמעותית יותר גנריות וכפי שנראה בקרוב הן למעשה **שילוב** של מספר specific access rights ו-stand access rights ביחד. הרשאות לדוגמה בקבוצה הזו:

- GENERIC_READ - הרשאה לקרוא מידע שמתוחזק על ידי האובייקט. לשם הדוגמה, כאשר לאובייקט מסוג קובץ יש את ההרשאה הנ"ל, היא מתורגמת אל READ_CONTROL, SYNCHRONIZE ב- standard

access rights ו- FILE_READ_ATTRIBUTES, FILE_READ_EA, FILE_READ_DATA ב- specific access rights. כלומר סה"כ 5 סיביות.

- GENERIC_WRITE - הרשאה לכתוב מידע שמתוחזק על ידי האובייקט.
- GENERIC_EXECUTE - הרשאה להריץ (או להסתכל) את האובייקט.
- GENERIC_ALL - הכל כולל הכל. כלל הסיביות של generic rights.

כלומר, כל אחת מאותם 4 סיביות האחרונות (GR, GW, GE, GA) למעשה שקולות למקבץ של סיביות באובייקט כאשר במידה והגדרנו ACE עם GA הוא הלכה למעשה מאפשר את כולן ביחד ונותן לנו כוח אבסולוטי על האובייקט (*wink wink* משם גם מגיע השם למאמר).

ברגע שאנחנו מגדירים generic rights על אובייקט, הן **מתורגמות** אל הסיביות המתאימות ב-access-mask מבחינת standard & specific rights על ידי ה-I/O manager לפני הקריאה אל מערכת הקבצים.

תאמת זה נשמע די מטופש, הרי מדובר בפעולה של AccessMask על האובייקט של עצמו ולא באיזה מנגנון משורשר שמצריך כזו רמה של הרשאות. *אז למה Microsoft עשו את זה ככה?* וואלה שאלה טובה, אני לא רואה שום סיבה מלבד נוחות לצוותי פיתוח עתידיים כדי שלא יתבלבלו בעצמם בכל הלוגיקה של AccessMask.

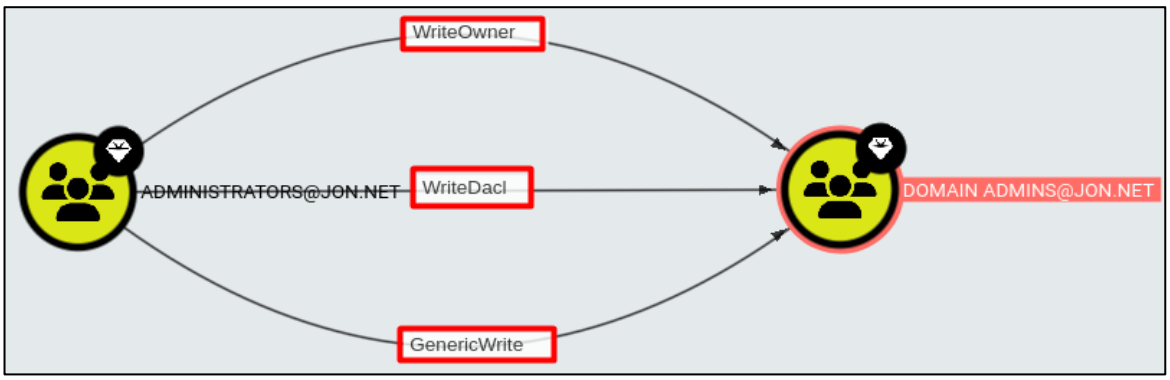
הערה - טוב זה הציק לי אז הייתי חייב לבדוק את הנושא, לפי העמוד access-mask בגיטהב של MicrosoftDocs הם מציינים שהפרדיגמה של איחוד מספר Generic rights ביחד (כדוגמת GENERIC_READ & GENERIC_WRITE) נובעת מהניסיון לחקות את הרכב ההרשאות המוכר של מערכות UNIXעם **"rwx"** באלמנטי הגישה למשאבי קבצים שכולנו מכירים. כמו כן, הם רושמים:

"The generic rights in the access mask simplify application development on Windows since these rights mask the different security rights for various object types."

משמע ההימור הראשוני שלנו התגלה כנכון.

מה ניתן לעשות עם שליטה על אובייקט?

טוב הגענו לתכל'ס; זה פרק עמוס אבל חשוב מאוד. סט הפעולות שדנו בהן בפרק הקודם הן אבני הבניין שישמשו אותנו לביצוע Lateral Movement בסביבת Windows - מעבר מיישות אחת לשנייה. אם נקפוץ להסתכל על גרף הפלט של כלי מוכר כמו BloodHound הרי שהמידע על הקשתות מייצג בדיוק את זה:

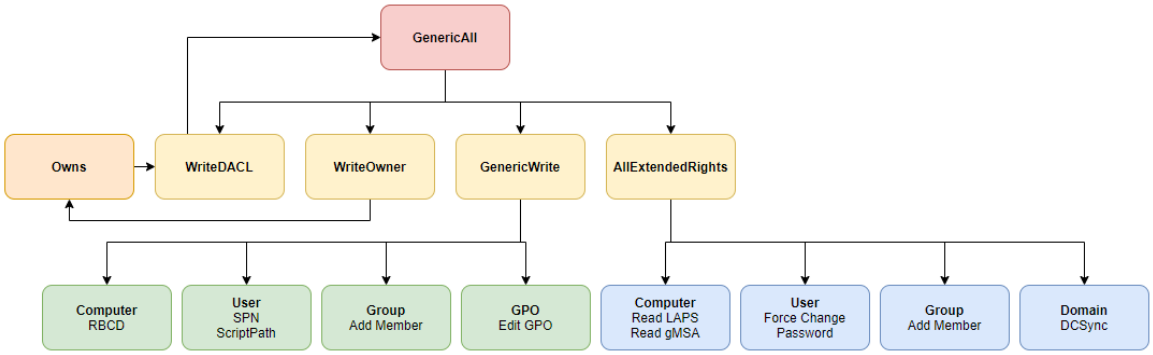


בדוגמה הפשוטה הזו לקבוצת ה-administrators יש ***שליטה*** על קבוצת ה-Domain Admins.

מה הכוונה ב-'שליטה'? ובכן, אלו הן הרשאות GenericWrite, WriteOwner, WriteDacl שבאמצעותן כל מי שחבר בקבוצת Administrators בדומיין יכול לקבוע סט פעולות על האובייקט של קבוצת Domain admins כדוגמת עריכת האובייקט. בצורה הזו, משתמש בקבוצת Administrators יכול להוסיף את עצמו לקבוצת DA ולאחר מכן להתחבר ל-DC ו-'לשלוט' בדומיין - קרי, לקבל גישה לאובייקטים נוספים.

את הנושא של התפשטות רוחבית וקטיעת אותם נתיבים מסוכנים (Attack paths) בדומיין באמצעות פעולות מפצות הצגתי ברמה **הפרקטית** במאמר Blue Hands-on BloodHound, מוזמנים להציץ. מטרת הפרק הקרוב לצאת מהרמה הפרקטית ולהציג את המכניזם מאחורי אותן פעולות.

אז אילו סוגי הרשאות יש לנו על אובייקטים ומה ניתן לעשות איתן? או במילים אחרות, **אילו מתקפות מבוססות ACL קיימות לנו בסביבת AD?** כהרגלנו, תמונה אחת אלף מילים:



[מקור - https://ppn.snovvcrash.rocks/pentest/infrastructure/ad/acl-abuse]

נראה מוכר? עיקר התפשטות רוחבית באמצעות שימוש ב-ACEs יהיו כתוצאה מ-misconfiguration או הרשאות מתירניות ברמת האובייקטים. לכן, תהליך האקספלויטציה של כל אי-הגדרה-מדוייקת עתיד להכיל שינויים בין ACE למשנהו אבל הרעיון הכללי נשאר זהה - יש לנו הרשאות שליטה על אובייקט ואנחנו הולכים להשתמש בהן בשביל **לזוז אליו או באמצעותו**.

הדוקומנטציה הישנה של BloodHound בנושא היא מקור נפלא לפירוט אינומרציית ACEs וכיצד ניתן לנצל אותם. לטובת אורך המאמר אנחנו לא נסקור את כולם אלא רק על כמה בולטים מתוכם. בנוסף לכך אני לא הולך לצרף את הקוד של כל תרחיש מהסיבה שברגע שמבינים את הרעיון, האספקט הטכני יורד בחשיבותו:

ForceChangePassword - יש לנו את ההרשאה **AllExtendedRights** על אובייקט מסוג משתמש ולכן אנחנו יכולים לשנות את הסיסמה שלו מבלי לדעת מה היא. ניתן לשנות סיסמה למשתמש בכמה דרכים, הראשונה (והכנראה הכי מוכרת) היא באמצעות הבינארי net.exe שמגיע דיפולטיבית עם מערכת ההפעלה:

```
net user username 'Password123!' /domain
```

אבל עקב שיקולי OpSec זו בהחלט לא דרך מומלצת מכיוון שהיא דורשת הרצה של הפקודה מהטרמינל ולא מעט סביבות רשתיות מנטרות הרצה של כלל הפקודות לפי בינאריים ותבניות ידועות (על ידי command line logging enabled), מה שללא ספק יעלה את הסיכוי לגילוי.

הדרך השנייה היא על ידי טעינה של ספריית Powershell מוכרת בשם PowerView לזיכרון. באמצעות טיפה PS shenanigans נוכל לייצר אובייקט של PSCredential ואובייקט של Secure string בשביל להזדהות מול ה-DC ואז להשתמש בפונקציה Set-DomainUserPassword (שמיובאת מ-PowerView) על מנת לשנות את הסיסמה:

```
powershell -ep bypass
. ./PowerView.ps1
$SecPassword = ConvertTo-SecureString 'Password123!' -AsPlainText -Force
$Cred = New-Object System.Management.Automation.PSCredential('Domain\\username', $SecPassword)
$UserPassword = ConvertTo-SecureString 'Password123!' -AsPlainText -Force
Set-DomainUserPassword -Identity Username1 -AccountPassword $UserPassword -Credential $Cred
```

כעת, כשיש לנו את הסיסמת plain text של המשתמש אנחנו **יכולים לבצע תזוזה רוחבית** ולעבור להשתמש במשתמש Username1 באמצעות New-PSSession, winrs, psexec, wmiexec או אפילו RDP ישירות. על ידי כך אנחנו נשנה את ה-security context שיש לנו ונוכל לבדוק גישה לאובייקטים נוספים בדומיין. הנושא של תזוזה רוחבית באמצעות סיסמאות וכלים שונים הוא מרתק ואני הולך להקדיש לו חלק במאמר של part 3 בסדרה.

AddMember - שליטה זו על אובייקט נובעת מכך שיש לנו ACE עם הרשאות GeneticWrite או AllExtendedRights על הקבוצה ואנחנו יכולים להוסיף אליה משתמשים נוספים. עקב security group delegation כל מי שחבר בקבוצה מקבל אוטומטית את ההרשאות של אותה קבוצה. לכן, במידה ויש לנו

שליטה כזו על אובייקט של קבוצה אנחנו פשוט יכולים להוסיף את המשתמש שלנו אליה ובכך לשנות את ה-security context שיש לנו.

גם כאן יש לפחות 2 דרכים לבצע את הפעולה. הראשונה, כמו שבוודאי ניחשתם, באמצעות net.exe:

```
net group "Domain Admins" username /add /domain
```

מאותם שיקולי OpSec שהצגנו מקודם ברור לנו שזו לא האופציה המועדפת. הדרך השנייה, באמצעות the mighty PowerView על ידי שימוש בפונקציה Add-DomainGroupMember. ב-ACE הקודם הסברנו את הקווים הכלליים היטב אז כעת נספק ב-PS shenanigans בלבד:

```
$SecPassword = ConvertTo-SecureString 'Password123!' -AsPlainText -Force
$Cred = New-Object System.Management.Automation.PSCredential('Domain\\username', $SecPassword)
Add-DomainGroupMember -Identity 'Domain Admins' -Members 'username' -Credential $Cred
```

כפי שניתן לראות, 2 האופציות האלו הן נקודות הקצה בגרף, כלומר הן מקרה פרטי ברמת האובייקט. בואו נעלה שלב אחד במעלה האיור ונסביר על 3 ACEs בולטים מתוכם:

GenericWrite - כשמו כן הוא, יש לנו הרשאות כתיבה (ושכתוב) של מאפייני attributes על אובייקט אחר. מדובר בסט רחב מאוד של אופציות שכולל בתוכו לא מעט דרכי ניצול ותזוזה בהתאם לסוג האובייקט עליו יש לנו את ההרשאה:

- **User** - במידה ויש לנו השראת GenericWrite על אובייקט משתמש אנחנו יכולים להפוך אותו ל-SPN (כלומר משתמש שמייצג שירות מסוים) באמצעות הפונקציה Set-DomainObject ואז לבצע עליו kerberoasting. מבחינה מעשית, נוכל לבקש את את הטיקט TGS של ה-SPN באמצעות הפונקציה Get-DomainSPNTicket וברגע שיש לנו ה-TGS הלכה למעשה יש לנו את ה-hash של סיסמת המשתמש ואנחנו יכולים לנסות לשבור אותו כנגד מילון ב-offline. גם Set-DomainObject וגם Get-DomainSPNTicket מגיעות עם ספריית PowerView.
- **Group** - אנחנו יכולים לכתוב principals חדשים לקבוצה, כלומר להוסיף אליה משתמשים בדיוק כמו בדוגמה שהראינו ב-AddMember מקודם.
- **Computer** – ת'אמת אנחנו לא יכולים לעשות יותר מדי עם הרשאה של GenericWrite על מחשב בדומיין ברמת המתקפות מלבד ביצוע של resource-based constrained delegation כנגד המכונה עצמה. מדובר בנושא שאני מעוניין לכתוב עליו מאמר בפני עצמו כחלק מהסדרה הנ"ל אז לא נכנס אליו כרגע אלא ארפרנס את המאמר הפנטסטי Kerberos Delegation 101 מאת ספיר פדרובסקי עבור מי שמעוניין להעמיק בנושא.
- **GPO** - עם שליטה על ה-attributes של חוקת GPO אנחנו יכולים לעשות הרבה דברים מעניינים. כמו למשל לבצע שינוי לחוקה קיימת שנוגעת אל משתמשים ו\או מחשבים באמצעות ה-GUI של gpedit דרך משימה מתוזמנת (כמובן שעל מנת לגעת ב-GPO אנחנו צריכים שתהיה לנו גם הרשאת כתיבה לתיקיית הבת המתאימה ב-SYSVOL). ניתן לבצע נושאים נוספים ומורכבים על ידי שליטה בחוקות אך לא נכנס אליהם אלא נרפרנס שוב את גברת פדרובסקי עם מאמר נוסף שפורסם במסגרת המגזין - Group Policy 101.

הערה - מכיוון שההגדרות של חוקות GPO מאוחסנות בתקיית בת עם שם ה-{31B2F340-...} GUID תחת תיקיית SYSVOL ב-DC, ה-security descriptor **משתקף על התיקיה עצמה**. כלומר על מנת לשנות חוקת GPO נדרש שיהיה למשתמש שלנו גם הרשאות write access ל-GPO. לרוב, עקב ירושת הרשאות דיפולטיבית, אם יש לנו שליטה על ה-GPO יהיו לנו גם את ההרשאות המתאימות אבל במידה ומנהלי הרשת ביטלו את עניין הירושה אנחנו צפויים להיתקל ב-false positive.

WriteOwner - ה-owner של כל אובייקט יכול לערוך את ה-headers שלו - משמה הוא יכול לשכתב את ה-security descriptor של האובייקט (ללא תלות בהרשאות שבתוך ה-DACL שלו). אם יש לנו את הרשאת WriteOwner אנחנו יכולים לשנות את הבעלות על אובייקט על ידי שימוש בפונקציה Set-DomainObjectOwner בספריית PowerView. ברגע שאנחנו הבעלים החוקיים של האובייקט שום דבר לא מונע מאיתנו להוסיף את ההרשאה של WriteDACL - כלומר לשנות ה-DACL שלו, מה שמוביל אותנו בצורה מעולה לסעיף הבא.

WriteDACL - אם יש לנו גישה ל-DACL של האובייקט **אנחנו יכולים להוסיף איזה ACE שנרצה** עבור המשתמש שלנו עם (כמעט) כל ההרשאות שנרצה. סוג האובייקט שיש לנו גישה אל ה-DACL שלו משנה את הסקופ של הפעולה אבל העיקרון נשאר דומה - אנחנו מוסיפים ACE באמצעות Add-DomainObjectAcl לגישה לכל האובייקט ואז מנצלים אותה. גם כאן, אציג בקצרה על סוגי אובייקטים שונים:

- **User** - עם WriteDACL על משתמש אנחנו יכולים לתת לעצמו שליטה מלאה עליו ואז לשנות לו סיסמה כמו שראינו בדוגמה למעלה.

```
Add-DomainObjectAcl -TargetIdentity username -Rights All
net user username 'Password123!' /domain
```

- **Group** - על ידי כתיבה אל ה-DACL של קבוצה נוכל לתת לעצמנו את היכולת להוסיף משתמשים נוספים לקבוצה ואז להוסיף את עצמינו אליה:

```
Add-DomainObjectAcl -TargetIdentity "Domain Admins" -Rights WriteMembers
net group "Domain Admins" username /add /domain
```

- **Computer** - בצורה דומה, ניתן לעצמנו שליטה מלאה על המכונה. לאחר מכן נוכל לערוך attributes של המחשב, לקרוא סיסמאות LAPS, לבצע resource-based constrained delegation כנגד מחשב אחר וכד'.
- **Domain** - איזו יכולת אנחנו רוצים בגישה לדומיין? למה לא לקרוא את כל המידע שיש בנוגע למחשבים, קבוצות ומשתמשים שברשות ה-DC על ידי ביצוע רפלקציה עם יכולת DCSync? נוכל לקבל הרשאות לזה בדרך הבאה:

```
Add-DomainObjectAcl -TargetIdentity domain.local -Rights DCSync
```

לאחר מכן נוכל לבצע את ה-DCSync באמצעות מספר אופציות. באמצעות ווריאציה של mimikatz:

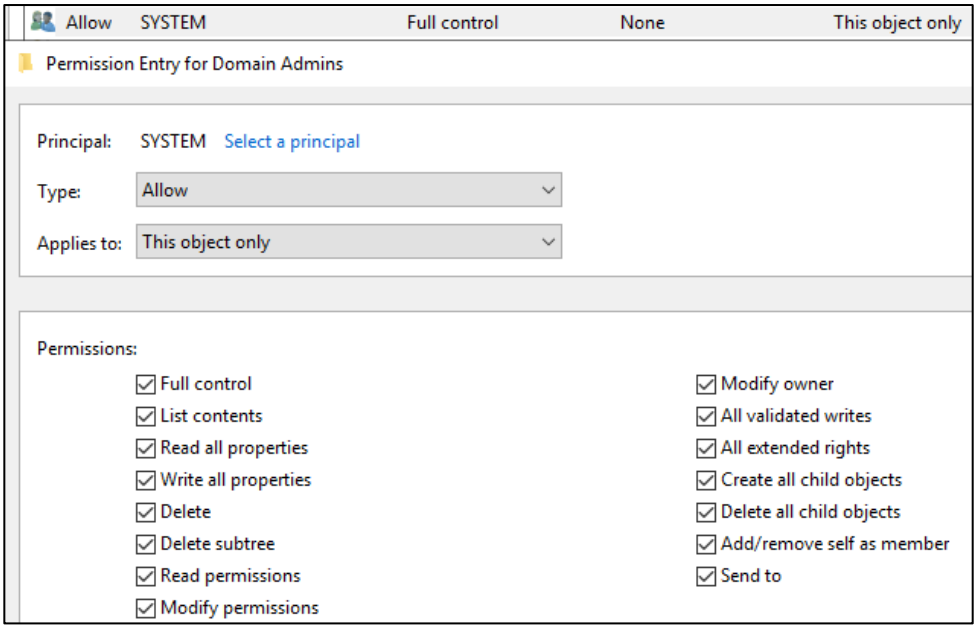
```
BetterSafetyKatz.exe "lsadump::dcsync /all" "exit"
```

- **GPO** - נוסיף ACE עם שליטה על ה-GPO ואז נשתמש בו בשביל לקחת שליטה על אובייקט אחר.

- **OU -** ניקח שליטה על OU ספציפי ואז נוסיף ACE חדש שמוריש לכל ה-child objects ב-OU בשביל לקבל שליטה גם עליהם.

כפי שניתן לראות מהאיור בתחילת הפרק, ארבעת סוגי השליטה שהצגנו עד כה (WriteOwner, WriteDACL , GenericWrite ו-AllExtendedRights) הם תת למעשה סט מצומצם של פעולה ראשית יותר והיא **GenericAll**. אם יש לנו ACE של GenericAll על אובייקט - יש לנו **שליטה מוחלטת** עליו.

מה הכוונה ומה אפשר לעשות עם GenericAll על אובייקט? ובכן התשובה הקצרה היא **הכל מהכל**. קרי, כל מה שעשינו בכל אחד מהמקרים הפרטיים עד כה ויותר. לשם ההמחשה, נכנס לממשק של ADUC ב-DC ונסתכל על ה-ACE של משתמש SYSTEM \NT AUTHORITY על קבוצת DA:



איך מוצאים ACL-ים בכל הג'ונגל של AD

אז הצגנו מה ניתן לעשות במידה ומצאנו DACL שמקונפג לא נכון (או עם הרשאות מתירניות) אבל **כיצד ניתן למצוא אותם?**

השלב הראשון הוא להבין כיצד אנחנו יכולים לתחקר אובייקט בדומיין בנוגע ל-Security descriptor שלו. הכלי המוכר ביותר למשימה הוא לא אחר מאשר BloodHound. מכיוון שהקדשתי לו מאמר שלם בעבר ומכיוון שהכלי עושה המון דברים לפני שהוא ניגש לפרסר אובייקטים (הכנסה ושליפה למסד נתונים מסוג Neo4j ו-Postgres, תרגום להצגה גרפית ב-React וכד') אני לא אסביר את הקוד שלו אלא את הקוד של כלי הרבה יותר נקודתי ורלוונטי להבנה שלנו - פונצקיות נבחרות מ-PowerView.

Fun fact - בכתיבת שניהם לקח חלק Will Schroeder הלא הוא harmj0y המפורסם. זה לא סתם שכל הרפרנסים לשיטות ניצול שונות ב-BloodHound מומלצות לביצוע על ידי PowerView ולא על ידי כלים אחרים. למעשה, לפי ההרצאה Six Degrees of Domain Admin ב-DEF CON 24 שבמהלכה פורסם BloodHound לראשונה, כתוב במפורש שהדרך שבה BH אוסף מידע מבוססת על PowerView.

הערה - על מנת לפשט את ההסבר התיאורטי בחרתי בגרסה ישנה יותר של PowerView. הגרסה העדכנית, אשר מקושרת לרפו של PowerSploit, מבצעת בסך הכל את אותן פעולות אך עם משמעותית יותר 'Powershell shenanigans' עבור מקרי קצה ופלט מסודר. המטרה העיקרית של הפרק היא המחשה ברמת הקוד לתיאוריה שהוצגה עד כה.

אוקיי אוקיי אוקיי, אם קודם היינו מחוברים ל-GUI של Active Directory Administrative Center ב-ADUC ודרכו הסתכלנו על ההרשאות של קבוצת DA על עצמה הרי שכעת נראה את אותו הדבר רק באמצעות הפונקציה **Get-ObjectAcl** (או בשמה השני **Get-DomainObjectAcl**):

```
PS C:\Users\domain_admin\Desktop> Get-ObjectAcl -SamAccountName "domain admins" -domain jon.net


ObjectDN              : CN=Domain Admins,CN=Users,DC=JON,DC=NET
ObjectSID             : S-1-5-21-1403467943-1367565381-54031474-512
ActiveDirectoryRights : GenericRead
InheritanceType       : None
ObjectType            : 00000000-0000-0000-0000-000000000000
InheritedObjectType   : 00000000-0000-0000-0000-000000000000
ObjectFlags           : None                                          <-- ACE #1
AccessControlType     : Allow
IdentityReference     : NT AUTHORITY\Authenticated Users
IsInherited           : False
InheritanceFlags      : None
PropagationFlags      : None

ObjectDN              : CN=Domain Admins,CN=Users,DC=JON,DC=NET
ObjectSID             : S-1-5-21-1403467943-1367565381-54031474-512
ActiveDirectoryRights : GenericAll
InheritanceType       : None
ObjectType            : 00000000-0000-0000-0000-000000000000
InheritedObjectType   : 00000000-0000-0000-0000-000000000000          <-- ACE #2
ObjectFlags           : None
AccessControlType     : Allow
IdentityReference     : NT AUTHORITY\SYSTEM
IsInherited           : False
InheritanceFlags      : None
PropagationFlags      : None
```

הפלט שמתקבל הוא כל ההרשאות שיש למשתמש שלנו (domain_admin שחבר בקבוצת DA) על הקבוצה מקוטלגים לפי סוגי ה-ACEs. כלומר, כל המידע שאנחנו זקוקים לו על מנת לבצע תזוזה רוחבית.

בהחלט מעניין! נעבור על קוד המקור של הפונקציה Get-ObjectAcl בשביל להבין כיצד היא עושה את זה. לאחר קבלת פרמטרים מהמשתמש, הקוד משתמש ב-Get-DomainSearcher אשר מהווה פונקציה פנים

שימושית ב-PowerView לחיפוש אובייקטים ב-AD. ממליץ לעבור על הקוד דרך הקישורים ביחד עם התיאור שבמאמר.

```
begin {
        $Searcher = Get-DomainSearcher -Domain $Domain -DomainController $DomainController -ADSpath $ADSpath
                                       -ADSprefix $ADSprefix -PageSize $PageSize

        # get a GUID -> name mapping
        if($ResolveGUIDs) {
            $GUIDs = Get-GUIDMap -Domain $Domain -DomainController $DomainController -PageSize $PageSize
        }
    }
```

מכיוון שמטרת המאמר להבין קצת יותר לעומק כיצד הכל עובד ולא להישאר ברמה השטחית של הרצת פקודות אני מרשה לעצמי להיכנס לתוך קוד הפונקציה של Get-DomainSearcher גם:

```
$SearchString = "LDAP://"

if($DomainController) {
    $SearchString += $DomainController + "/"
}
if($ADSprefix) {
    $SearchString += $ADSprefix + ","
}

if($ADSpath) {
    if($ADSpath -like "GC://*") {
        # if we're searching the global catalog
        $DistinguishedName = $AdsPath
        $SearchString = ""
    }
    else {
        if($ADSpath -like "LDAP://*") {
            $ADSpath = $ADSpath.Substring(7)
        }
        $DistinguishedName = $ADSpath
    }
}
else {
    $DistinguishedName = "DC=$($Domain.Replace('.', ',DC='))"
}

$SearchString += $DistinguishedName
Write-Verbose "Get-DomainSearcher search string: $SearchString"

$Searcher = New-Object System.DirectoryServices.DirectorySearcher([ADSI]$SearchString)
$Searcher.PageSize = $PageSize
$Searcher
```

אז מה אפשר לראות? בסופו של יום חשוב להבין שכאשר אנחנו מדברים על חיפוש אובייקטים בסביבת Active Dircectory אנחנו נדרשים לבצע תשאול (query) מאוד ממוקד אל מול שירות LDAP ב-DC שמייחצן את הממסד נתונים NTDS ומכיל את כל עץ המידע של הדומיין.

בשביל כך עלינו לבנות שאילתת LDAP ספציפית בנוגע לאובייקט. ניתן לבצע את הנ"ל באמצעות שפת Powershell שמכילה המון מבנים ופונצקיות Built in ממוקדות לסביבת Windows. בחזרה, אנחנו צפויים לקבל מזהה ייחודי לאובייקט - **Distinguished Name** עם מאפייניו.

בשביל להקל על תהליך כתיבת הקוד ב-PS משתמשים במונח שנקרא **Type accelerators** שזה פשוט דרך פנסית פנסית לומר alias לספריות Net.

ADSISearcher היא סוג של Type accelerator לספריה System.DirectoryServices.DirectorySearcher שמשתמשים בה על מנת לחפש אובייקט אחד או יותר על סמך פילטר. ואכן כל המטרה של הפונקציה Get-DomainSearcher היא לבנות את האובייקט ADSI searcher המתאים ולהחזיר אותו להמשך אנאליזה:

```
$Searcher = New-Object System.DirectoryServices.DirectorySearcher([ADSI]$SearchString)
$Searcher.PageSize = $PageSize
$Searcher
```

נחזור לריצה של Get-ObjectAcl - במידה ואנחנו מספקים GUID לכלי, הוא עושה שימוש בפונקציה Get-GUIDMap (פונקציית helper נוספת של הכלי) על מנת לבנות טבלה בהתאם:

```
        # get a GUID -> name mapping
        if($ResolveGUIDs) {
            $GUIDs = Get-GUIDMap -Domain $Domain -DomainController $DomainController -PageSize $PageSize
        }
    }
```

אז ביצענו שאילתא לאובייקט באמצעות Get-DomainSearcher וקיבלנו אותו בחזרה. כעת זה הזמן להתחיל להבין מה יש לנו מול העיניים ולהוציא ממנו את המידע של ה-ACL:

```
process {
    if ($Searcher) {
        if($SamAccountName) {
            $Searcher.filter="(&(samaccountname=$SamAccountName)(name=$Name)(distinguishedname=$DistinguishedName)$Filter)"
        }
        else {
            $Searcher.filter="(&(name=$Name)(distinguishedname=$DistinguishedName)$Filter)"
        }
        try {
            $Searcher.FindAll() | Where-Object {$_} | Foreach-Object {
                $Object = [adsi]($_.path)
                if($Object.distinguishedname) {
                    $Access = $Object.PsBase.ObjectSecurity.access
                    $Access | ForEach-Object {
                        $_ | Add-Member NoteProperty 'ObjectDN' ($Object.distinguishedname[0])
                        if($Object.objectsid[0]){
                            $S = (New-Object System.Security.Principal.SecurityIdentifier($Object.objectsid[0],0)).Value
                        }
                        else {
                            $S = $Null
                        }
                        $_ | Add-Member NoteProperty 'ObjectSID' $S
                        $_
                    }
                }
```

זהו לב הלוגיקה של הקוד. לפי הפרמטרים שהתקבלו מהמשתמש יוצרים פילטר LDAP; לאחר מכן באמצעות $Searcher.FindAll() עוברים אחד אחד על סט האובייקטים של AD שהתקבלו וממירים אותם אל אובייקטי ADSI.

אז, עבור כל אובייקט מריצים את $Object.PsBase.ObjectSecurity.access אשר מחזיר כל ACE בתוך ה-DACL. עבור כל ACE הקוד מוסיף לעצמו מאפיינים של ObjectSID & ObjectDN על מנת לייצג את ה-distinguished name וה-security identifier של האובייקט. לבסוף, הפונקציה יוצרת ושומרת אובייקט powershell-י ומכניסה אליו את כל המאפיינים של ה-ACL של האובייקט.

לאחר שהפונקציה סיימה לפרסר את כל המידע על כלל האובייקטים, במידה והמשתמש הכניס שהוא מחפש הרשאה ספציפית הקוד מפלטר לפיה על פני כל האובייקטים שזה עתה הוא יצר. לא הכי אפקטיבי מבחינת קוד אבל בהחלט עושה את העבודה:

```
} | ForEach-Object {
    if($RightsFilter) {
        $GuidFilter = Switch ($RightsFilter) {
            "ResetPassword" { "00299570-246d-11d0-a768-00aa006e0529" }
            "WriteMembers" { "bf9679c0-0de6-11d0-a285-00aa003049e2" }
            Default { "00000000-0000-0000-0000-000000000000"}
        }
        if($_.ObjectType -eq $GuidFilter) { $_ }
    }
    else {
        $_
    }
```

יש קטע קוד נוסף להתייחסות אל GUID אבל הרעיון דומה.

אבל יונתן יש המון המון אובייקטים בדומיין ולבדוק אחד אחד זה לא פרקטי, האם אין כלי שיכול לבצע אוטומטציה ולבדוק את כולם בשבילנו?

ובכן בהחלט שיש. הסקריפט Find-InterestingDomainAcl (שגם הוא חלק מ-PowerView) עושה בדיוק את זה דרך ה-CLI. גרסה משופרת של הכלי הוא כמובן BloodHound אשר מוסיף אלמנט של טעינה אל GUI מבוסס web ומריץ אלגוריתמיקה גרפית בשביל לחבר את הנקודות בדרכים הקצרות ביותר.

אבל, וזה אבל חשוב, **הרעיון של שני הכלים זהה** - מעבר על כלל האובייקטים בדומיין ובדיקת כלל ה-ACEs ב-DACL לשם הבנה אילו מהם מקבלים הרשאות לאובייקטים אחרים וכיצד ניתן להשתמש בהם להתפשטות רוחבית.

הפונקציה Find-InterestingDomainAcl רצה ב-CLI של PS ולכן עם ידע מינימאלי ב-PS ניתן לבצע חיתוכים מאוד ספציפיים בעזרתה. למשל, אם קיבלנו אחיזה על משתמש דומייני נוכל לראות את כלל האובייקטים שיש לו (ולקבוצות שהוא חבר בהן) גישה על ידי:

```
# check for modify ACL rights/permissions for the UserA
Find-InterestingDomainAcl -ResolveGUIDs | ?{$_.IdentityReferenceName -match "UserA"}

#Since UserA is a member of the RDPUsers group, let us check ACL permissions for it as well
Find-InterestingDomainAcl -ResolveGUIDs | ?{$_.IdentityReferenceName -match "RDPUsers"}
```

אם נסתכל בדוקומנטציה או בקוד המקור של הכלי נוכל לראות שלב ליבו של הקוד היא קריאה לפונקציה שכבר הכרנו בתחילת הפרק הלא היא **Get-DomainObjectAcl** שהצגנו בקודם ומעבר על כלל האובייקטים שהיא מחזירה:

```
PROCESS {
    if ($PSBoundParameters['Domain']) {
        $ACLArguments['Domain'] = $Domain
        $ADNameArguments['Domain'] = $Domain
    }

    Get-DomainObjectAcl @ACLArguments | ForEach-Object {
```

לאחר מכן, עבור כל ACE שחוזר, הפונקציה בודקת אם ה-SID שמקושר למשתמש שלו גבוה מ-1000 (משמה הוא לא משתמש built in שנוצר עם הסביבה) והאם ההרשאות שבתוך ה-ACE מאפשרים לשנות את האובייקט על ידי המשתמש. במידה והתשובה לשני התנאים האלו חיובית - מתרגמים בחזרה את ה-SID של המשתמש אל האובייקט הדומייני ומספקים אותו ביחד עם מידע נוסף לפלט של הכלי.

למעשה, אם נסתכל על מימוש ישן יותר של הסקריפט (אז קראו לו Invoke-ACLScanner) נוכל להכניס לצילום מסך אחד את כלל האלגוריתם פלוס אילו הרשאות יכולות "לשנות את האובייקט":

```
# Get all domain ACLs with the appropriate parameters
  Get-ObjectACL @PSBoundParameters | ForEach-Object {
      # add in the translated SID for the object identity
      $_ | Add-Member Noteproperty 'IdentitySID' ($_.IdentityReference.Translate(
                      [System.Security.Principal.SecurityIdentifier]).Value)
      $_
  } | Where-Object {
      # check for any ACLs with SIDs > -1000
      try {
          [int]($_.IdentitySid.split("-")[-1]) -ge 1000
      }
      catch {}
  } | Where-Object {
      # filter for modifiable rights
      ($_.ActiveDirectoryRights -eq "GenericAll") -or ($_.ActiveDirectoryRights -match "Write")
      -or
      ($_.ActiveDirectoryRights -match "Create") -or ($_.ActiveDirectoryRights -match "Delete")
      -or
      (($_.ActiveDirectoryRights -match "ExtendedRight") -and ($_.AccessControlType -eq "Allow"))
  }
```

כאמור, הגרסה של היום (Find-InterestingDomainAcl) קצת יותר מורכבת אבל רעיונית דומה מאוד.

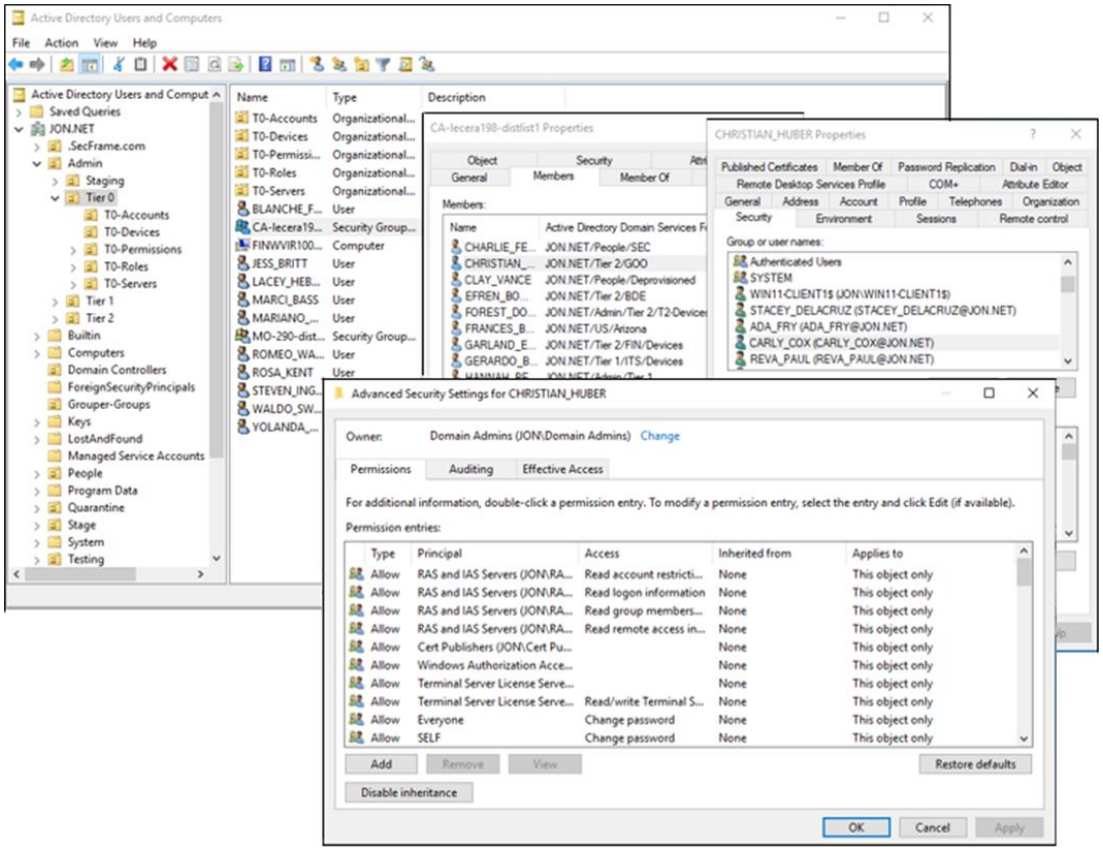
כמה פרקטי כל הסיפור הזה עבור Lateral Movement?

אם את המאמר הקודם סיכמנו בכך שהנושא של Access tokens הוא אחלה אבל הוא לא בדיוק הלחם והחמאה של התפשטות רוחבית הרי שהנושא הנ"ל של ACEs שונה משמעותית. בשביל להדגים את הכוח האמיתי של הנושא ולדמות סביבה איכשהו אמיתית יצירתי דומיין עם 500 משתמשים, 100 קבוצות, 30 מחשבים, רמות שונות של Ous ופיצ'רים מגננתיים כגון LAPS. כל אלו חוברו על ידי עץ הרשאות פגיע על ידי ג'ינרוט של DACLs מאוכלסים ב-ACEs חצי רנדומלית בין האובייקטים שנוצרו.

נשמע כמו אופרציה מורכבת אבל למען הכנות הכל לקח פחות מ-10 דקות על ידי הרצה של כלי מגניב בשם BadBlood אשר נועד לבצע בדיוק את זה - דימוי דומיין מציאותי בשביל לאפשר למהנדסים, אנליסטים וחוקרים לבדוק כראוי מוצרי הגנה ומנגנונים בסביבה מורכבת. מוזמנים לנסות בעצמכם, ממליץ לשמור snapshot של הדומיין עם AD מקונפג לפני הרצת הכלי ופשוט להתנסות עם תצורת AD אחרת בכל פעם מחדש :)

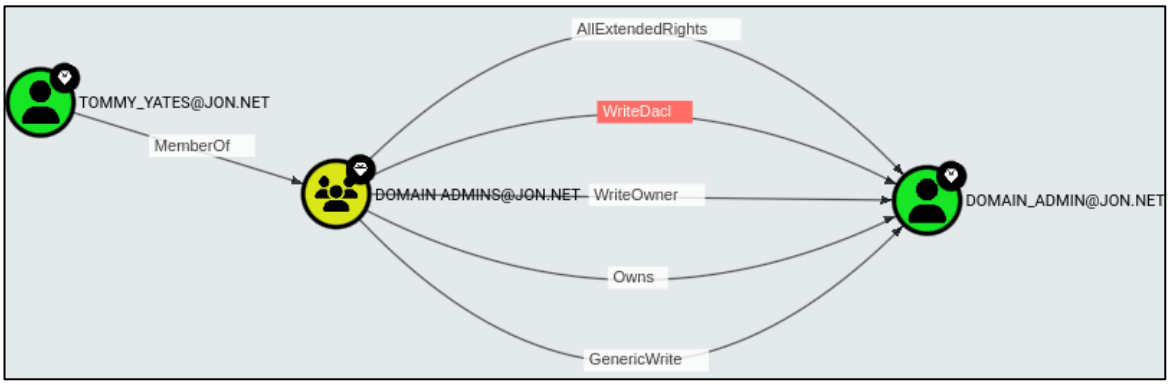
הערה - למרות שהכלי יכול לדמות סביבת דומיינית מציאותית, החיסרון העיקרי שלו היא העובדה שהוא לא יכול לדמות sessions פעילים (שמה הם לא אובייקטים בדומיין ומחייבים התחברות אינטראקטיבית של משתמשים) אבל לצרכי המאמר הנוכחי זה כלל לא מפריע לנו. כמו כן, חשוב לציין שהכלי ממש (ממש!) לא נבנה להרצה בסביבת פרודאקשן כי הוא הלכה למעשה מכניס פגיעויות לסביבה.

לאחר הרצת הכלי, הדומיין החדיש חדש שלנו יראה כך:

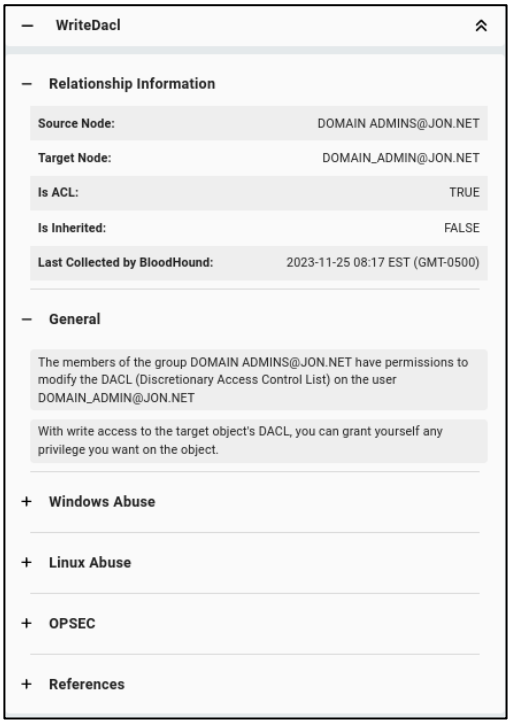


השלב הבא בשביל להציג את נתבי הסלמת הרשאות ותזוזה רוחבית בדומיין על ידי בדיקה של כלל ה-DACL-ים של האובייקטים בדומיין יהיה שימוש ב-BloodHound (או Find-InterestingDomainAcl). לכלים מבוססי CLI תמיד יהיו את היתרונות של המהירות והפשטות (של העברה וההרצה שלהם) אך ללא ספק כלים מבוססים ממשק משתמש ברורים יותר, במיוחד לצרכי מאמרים.

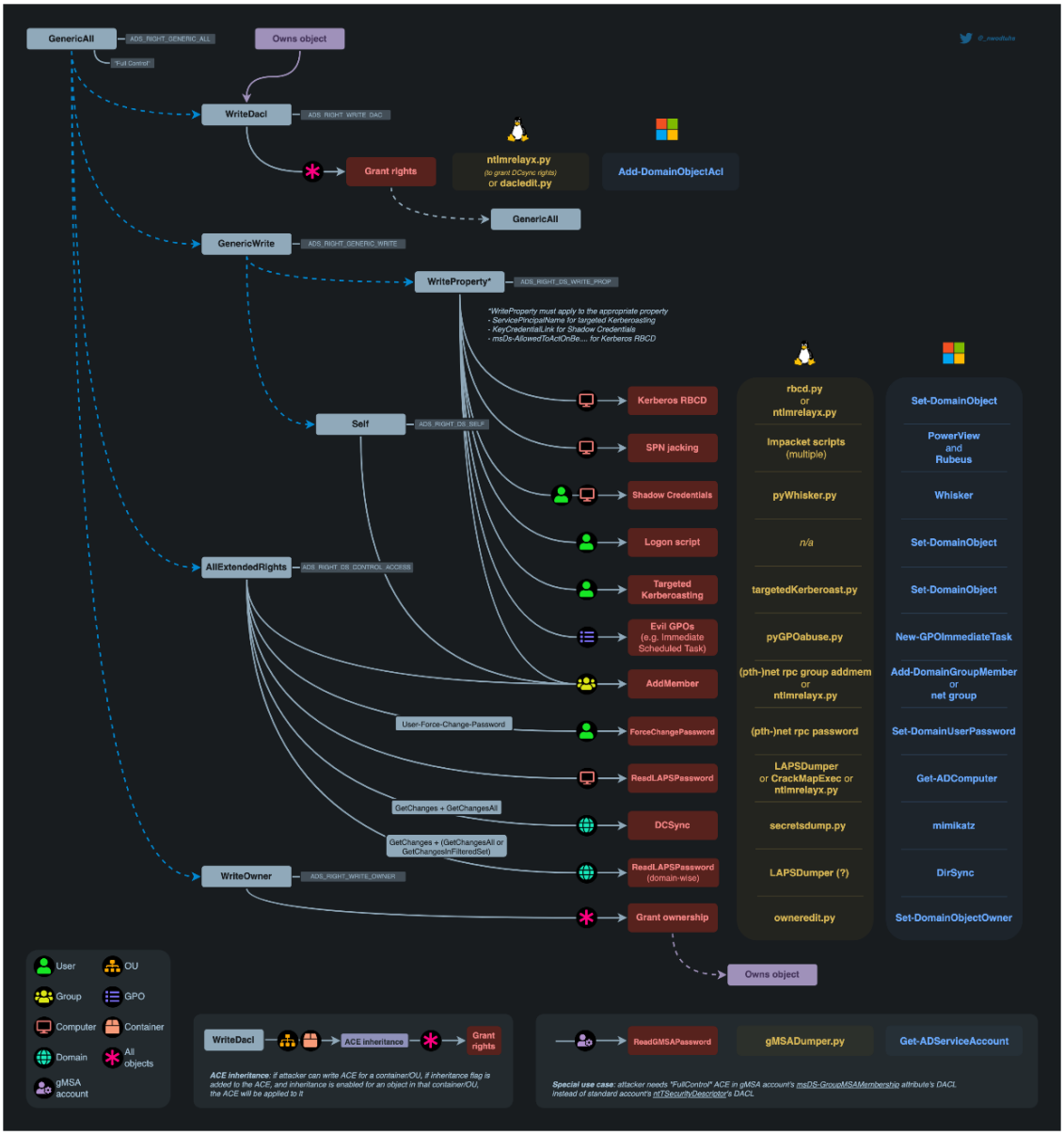
לאחר הרצה של sharphound.exe (הקולקטור של BH) ומעבר כלל הקבצים אל המכונה שלנו, נקליד את המשתמשים שיש לנו אחיזה עליהם ונבחן את נתיבי התקיפה להגעה אל משתמשים חזקים אחרים. לשם הדוגמה נגיד שברשותינו המשתמש Tommy_Yates וברצונינו להגיע למשתמש Domain_admin. נתיב התקיפה יראה בצורה הבאה:



כידוע, המידע על הקשתות מייצג את הדרך אל השתלטות על האוביקט בכיוון החץ. מזהים את המידע שיש בקשתות? **אלו ה-ACEs!** כל נתיב רלוונטי להתפשטות. לשם הדוגמה, אם נרחיב את המידע של קשת WriteDacl שמחברת בין האובייקט של קבוצת Domain Admins אל המשתמש Domain_admin נוכל לקבל מידע ספציפי לגביו:



בחלוניות של Windows Abuse, Linux Abuse נוכל למצוא את הדרכים לביצוע התזוזה הרשתית ברמה הפרקטית. כאמור, לכל הרשאה ומעבר מ-principal אחד לרעהו יש פרוצדורה מובנת וידועה. איור חביב שחוזר לא פעם בנושא ומסכם את כל ההיבט הפרקטי מבחינת הרשאות בצמוד אל כלים לביצוע האקספלויטציה:



[מקור - https://www.thehacker.recipes/a-d/movement/dacl]

כפי שניתן לראות מהאיור, ההיבט המעשי של איזה כלי להריץ באיזו סביבה בהתאם ל-output שהתקבל מביצוע האנומרציה על שדה DACL באובייקט לביצוע המתקפה (באדום) הוא אומנם מסועף אבל לא מורכב. למעשה עם סיכום מקוטלג ומאורגן, הדרך אל ניצול אפקטיבי ומהיר די פשוטה ושטחית.

Either you die as an open-source project or live long enough to become a commercial one

כאמור, פרויקט BloodHound מוכר מאוד לקבלת מראה גרפי של הדומיין. למעשה, כל המידע על תזוזה רשתית על ידי אינומרציה של שדה DACL נכנס לכלי במאי 2017 בגרסה 1.3 לפי פרסום מאמר An ACE Up the Sleeve מאת אותם הכותבים של BH עצמו. מדובר במאמר פנטסטי, 10\10 אם תשאלו אותי ולכן אני מפציר בכל מי שמתעניין בנושא לקרוא אותו.

עם חלוף השנים, הפרויקט של BH גדל ונוספו לו מספר רב של יכולות. לא אכנס לכל רבדיו של הסיפור אבל בסופו של יום חברת SpecterOps לקחה בעלות על הפרויקט ומעתה ואילך ישנן 3 גרסאות לכלי:

- BloodHound - גרסת Legacy אשר צפויה להיגרט בעתיד הקרוב
- BloodHound Community Edition - גרסת קהילה חינמית מבוססת קוד פתוח
- BloodHound Enterprise - גרסה מבוססת קוד סגור עם פיצ'רים נוספים לגרסת CE

ההבדל העיקרי בין גרסת הקהילה לגרסה המסחרית של הכלי הוא פעולת ההמרה של נתיבי תקיפה אל "**choke points**" מרכזיים אשר עתידים לקטוע את רצף התפשטות התוקפים. עם כל הכבוד ל-SpecterOps (ויש המון כבוד!!!) מדובר ביכולות שקיימות כבר שנים בשוק. בין אם במוצרים בוגרים כדוגמת PingCastle, Forest Druid ובין אם בפרויקטי קוד פתוח כדוגמת PlumHound, ADACLScanner ו-GoodHound שאת הפתרון שלהם הצגתי במאמר Blue Hands-on Bloodhound בעמודים 17-21.

אין לי יותר מדי ביקורת על המהלך הזה, בסופו של יום זו אחת הדרכים העיקריות להרוויח שכר מכלי open-source שפיתחת וללא ספק למודל העסקי של One-stop Shop יש את היתרונות שלו.

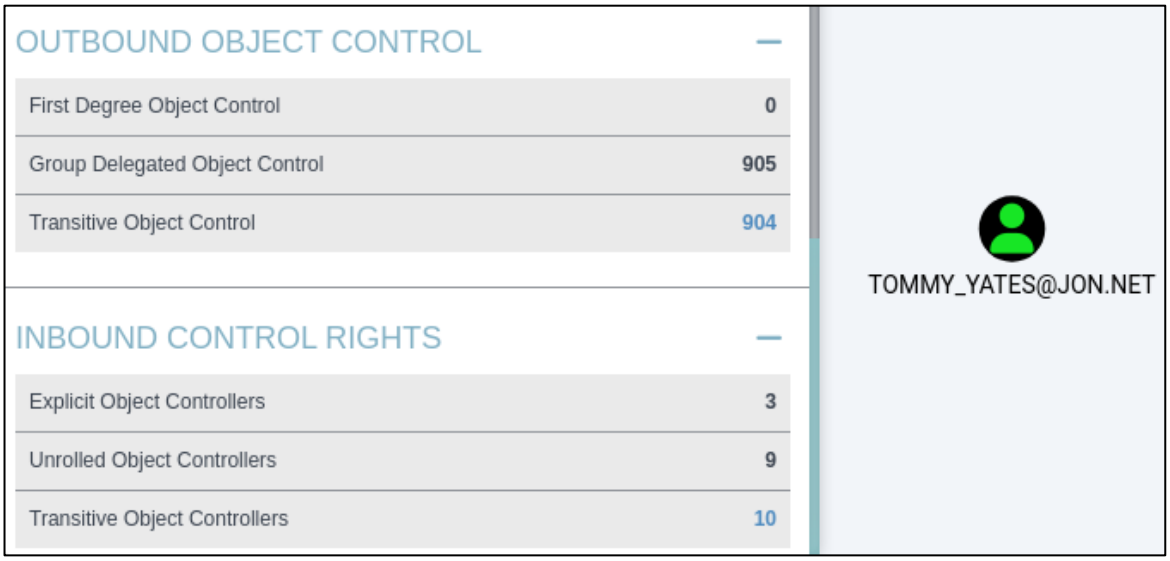
נחזור לתכלית המאמר - מציאת נתיבי התפשטות לתזוזה רוחבית מבוסס DACL. ישנן 3 אופציות שיעניינו אותנו במיוחד:

Explicit Object Controllers - הצלבה של מספר יישויות עם ACEs שניתן לנצל בתוך ה-DACL של אובייקט ספציפי. כלומר, על ידי כל אחת מהיישויות הללו ניתן לבצע תזוזה רוחבית בשביל להשתלט על האובייקט שמעניין אותנו.

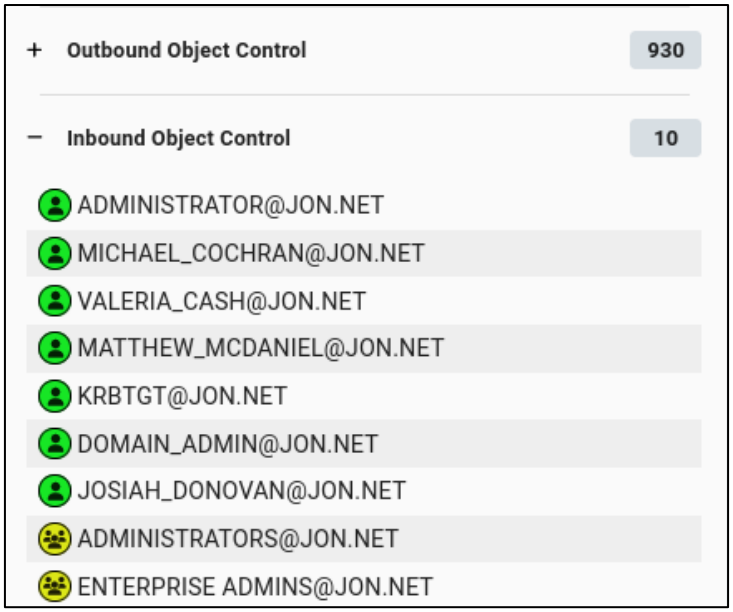
Unrolled Object Controllers - מספר היישויות עם ACEs שניתן לנצל ביחס אל security group delegation. כלומר אם קבוצה שולטת על קבוצה אחרת יראו את כל מי שחבר בה (בצורה כזו אם יש לנו שליטה על אחד מהם ניתן להשתלט בקלות גם על האובייקט שמעניין אותנו).

Transitive Object Controllers - שילוב של שתי האופציות הקודמות + נתיבים מבוסס DACL ישירים.

בממשק גרפי (**של גרסת ה-Legacy**) נוכל למצוא את שלושת האופציות הנ"ל תחת Node Info ב-Inbound Control Rights:



תאמינו או לא אבל בגרסה החדשה של BloodHound Community Edition הורידו את הפיצ'רים האלה וכעת במקום הפירוט על ההצלבה של השליטה באובייקט יש רק רשימה של שמות ה-Principals עם התצורה הגרפית של Explicit Object Controllers:



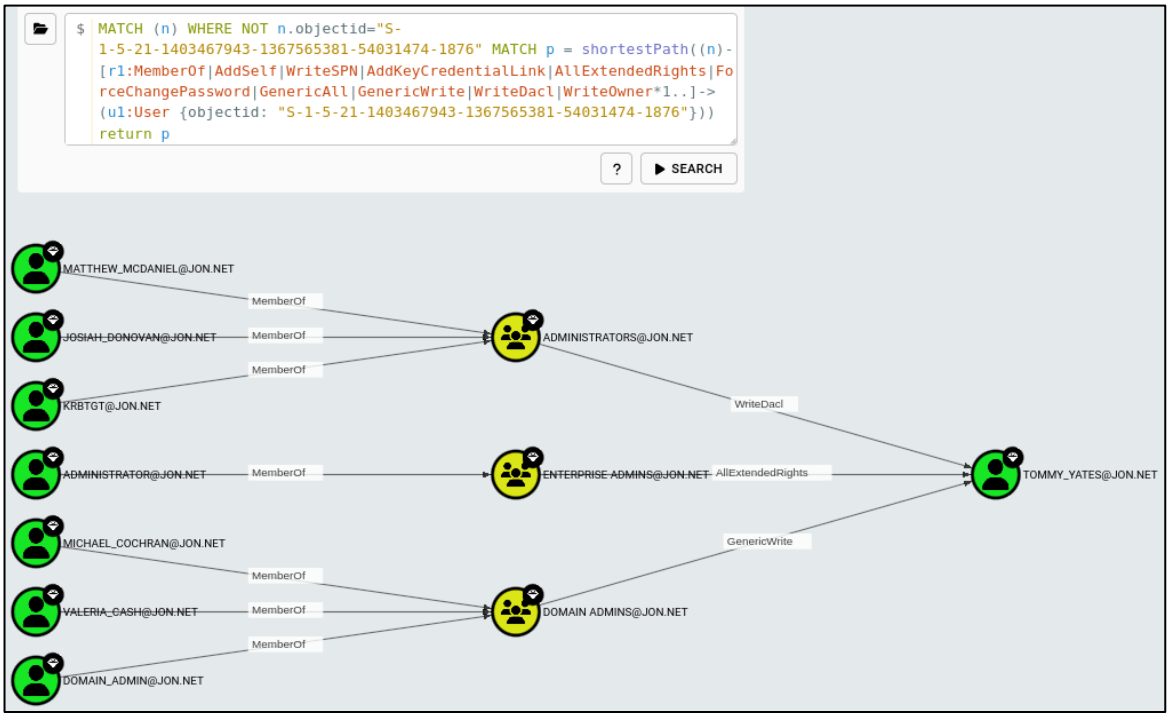
בהחלט מקומם שחברה בוחרת להוריד במודע פיצ'רים במוצר שלה. בכל מקרה, בין אם עשו זאת מצרכי נוחות UI/UX או בשביל להגדיל את הפער בין הגרסת Community אל גרסת Entreprised של הכלי, שווה לזכור שממשק המשתמש של BH מהווה בסך הכל שכבה גרפית להצגת מידע החוזר משאילתות הנשלחות אל ממסד הנתונים. כל לחיצה על אופציה כזו או אחרת בעצם מתורגמת ישירות עם לחיצתה אל שאילתת **CQL** על ידי ה-backend.

יתרה מכך, מכיוון שמדובר בקוד שעד לא מזמן היה זמין לכולם, אנחנו יכולים **להוסיף את הפיצ'רים האלו בחזרה** לגרסת CE הנוכחית.

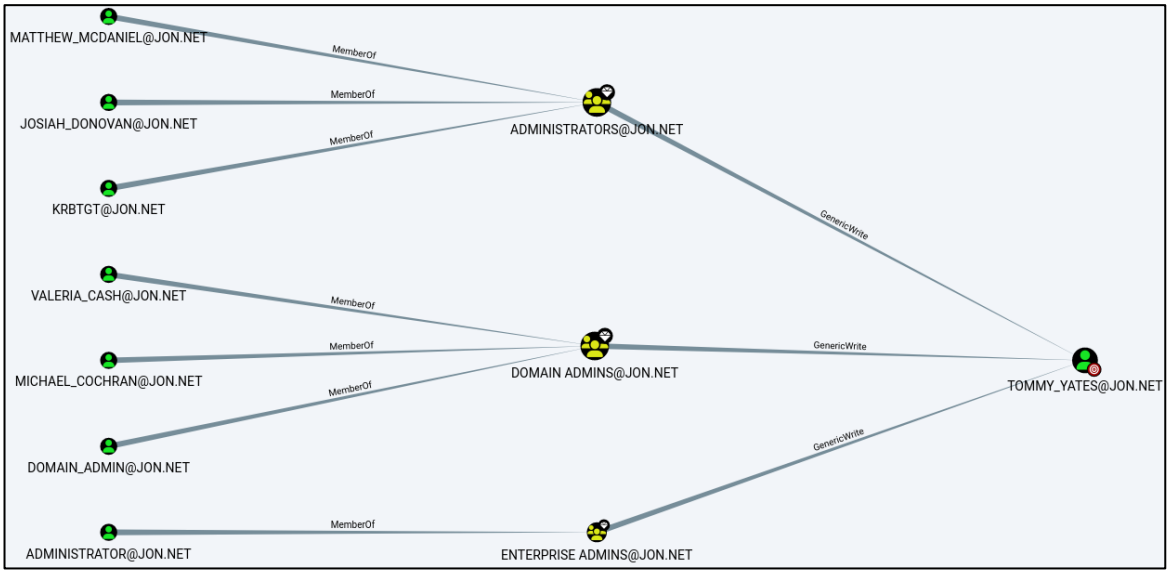
בשביל כך, ראשית נמצא את קוד המקור שמבצע את האנליזות ויוצר את שאילתת CQL בגרסה הקודמת של BH. חיפוש קצרצר יביא אותנו אל קובץ UserNodeData.jsx. לשם הדוגמה אני אבחר לממש את השאילתא "Transitive Object Controllers". בקוד מקור הישן היא נכתבת בצורה הבאה:

```
<NodePlayCypherLink
   property='Transitive Object Controllers'
   target={objectId}
   baseQuery={
      'MATCH (n) WHERE NOT n.objectid=$objectid MATCH p = shortestPath((n)-
[r1:MemberOf|AddSelf|WriteSPN|AddKeyCredentialLink|AllExtendedRights|ForceChangePassword|Gene
ricAll|GenericWrite|WriteDacl|WriteOwner*1..]->(u1:User {objectid: $objectid}))'
   }
   end={label}
   distinct
/>
```

נעתיק את השאילתא ועם שינויים מינימאליים נריץ אותה מול משתמש קיים (Tommy_Yates) לבדיקת היתכנות:



אבל האם היא נכונה? נכנס לגרסה הישנה של BH בה הפיצ'ר היה קיים ונבדוק אם אכן מה שאנחנו מקבלים זהה:



בינגו.

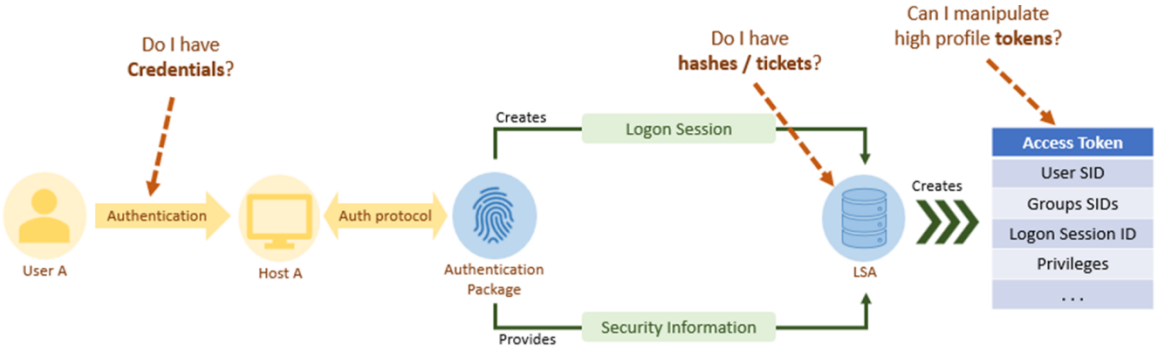
בצורה דומה ניתן להוסיף שאילתות משלנו לכלי ולפלטר לפיהן. יש המון (המוווון) שאילתות מגניבות שאפשר לחקור את הדומיין איתן אבל נשאיר את גילויהן לקורא הסקרן.

סיכום

עברנו הרבה במאמר, החל מההבנה מדוע המפתחים של ליבת מערכת ההפעלה יצרו לעצמם שכבה אבסטרקטית מאופיינת אובייקטים מסוגים שונים לניהול וחלה בהבנה של מדוע פילטור של כלל הדומיין על פי שדה ה-security descriptor הוא כיוון נכון לצד האדום.

למרות השכיחות הרבה של נתבי תקיפה מבוססים ACL-ים, כנראה שלא תופתעו לשמוע שזה בהחלט לא סוף הסיפור. **אז אילו עוד נושאים צפויים לנו במאמרי המשך?**

האיור של תהליך יצירת Access token מהמאמר הקודם יזכיר לנו את הנקודות בהם הצד האדום יכול להשתלב (ולנצל) את תהליך האימות בשביל לשנות את קונטקסט הגישה שלו:



בין אם זה שימוש ב-Kerberos Tickets והינדוס תצורתם ו\או תהליך קבלתם, משחק עם hash-ים וסיסמאות, העברה ישירה של Credentials לשם יצירת שירותים ותהליכים במכונות אחרות או הרצת קוד מרחוק באמצעות חיבוריות SQL; בסופו של יום חשוב להבין שכולם מתבססים על **שינוי הקונטקסט האבטחתי** שיש לנו לשם יצירת Token **וקבלת גישה** למשאבים נוספים. זוהי תזוזה רוחבית.

בקיצור - עולם ומלואו. יש למה לצפות! (:

על הכותב

יהונתן אלקבס, חוקר אבטחת מידע, חובב סוקולנטים, טיולים ואוהב את המדינה שלנו.