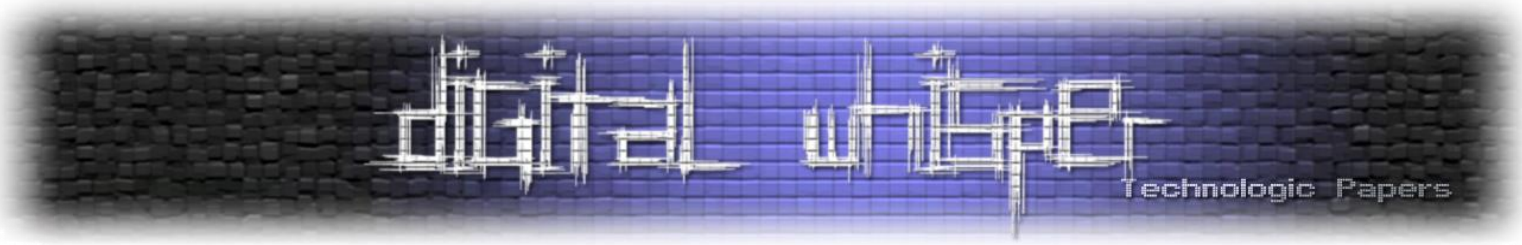


Leave the Door Open - Abusing Smartcard Authentication

מאת ערן נחשון

הקדמה

רוב הארגונים כיום משתמשים ב-Active directory לניהול הסביבה שלהם, ומדובר בתחום אשר נחקר רבות בהיבט האבטחתי. בנקודה בה אנו נמצאים היום, סביבת AD יכולה לשמש מגרש משחקים מדהים לתוקפים. למרות כל ה-Fuck-Ups אשר נהיו עם הזמן מובנים מאליהם גם לצד הכחול וגם לצד האדום, מסתמן ש-AD לא יעבור מהעולם שלנו כל כך במהרה. כשמצרפים לכל הכיף הזה קונספטים של הצפנות וחתימות דיגיטליות, זה אפילו יותר כיף, ושימושי יותר ממה שנדמה ממבט ראשון ☺

היום נבצע תקיפה אשר זכתה (על ידי) לכינוי Anderson PAC שמטרתה היא להוציא NTHash למשתמש מבלי לגעת ב-LSASS. בדרך לשם ניגע במגוון טכנולוגיות בסביבת Active Directory, נבצע Deep Diving לפרוטוקולי התאמתות ולעולם ה-Certificates ב-Domain - תחום עולה ששווה לשים לב אליו.

המאמר מיועד לאנשים שהתעסקו בסביבות AD, מכירים את הפרוטוקולים ואת המתקפות הבסיסיות, וגם קונספטים בסיסיים בהצפנות. במאמר זה לא נסביר את הפרוטוקולים Kerberos ו-NTLM לעומק, אלא ניכנס קצת יותר לעובי הקורה.

נתחיל מהשאלות המתבקשות - למה Smartcard? מדוע זה קיים?

מדובר באחד ממקרים רבים בהם ניסו למצוא פתרון טוב יותר לבעיה מסוימת: אימות אבטחתי וקל של משתמשים בסביבה דומיינית. בסביבת AD שגרתית שאנחנו מכירים, לכל משתמש תהיה סיסמא, שאותו היא משמשת להתחברות בכל עמדה בדומיין. לסיסמא (ולבני אנוש שמשתמשים בה) יש כמה בעיות קלאסיות:

- קל לשכוח אותה.
- Hash שניתן לשבירה (NetNTLMv1 לדוגמא).
- דורש תחזוקה ותחלופה על ידי המשתמשים.

- קל לנחש סיסמאות קלות בתור תוקף.

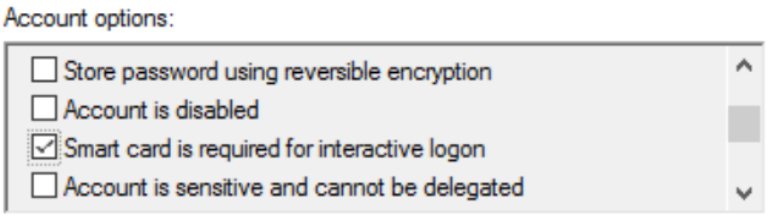
כמובן שניתן לשפר את הבעיות האלה עם פוליסת סיסמאות מיטבית וקביעת תוקף קצר עד לשינוי הסיסמא הבא, אך בכל זאת מדובר בפתרון די מיושן ומסורבל לתפעול בארגון גדול. לשם כך קיימות מגוון אלטרנטיבות, המרכזיות שביניהן היא התאמתות בעזרת Smartcards ו/או התאמתות ביומטרית. מבחינת המשתמש, הוא סך הכל צריך להקיש pincode קצרצר או לשים טביעת אצבע (שני אלו הם דברים שקל לזכור) ומה שבפועל קורה מאחורי הקלעים הוא התאמתות בעזרת Certificate ו-Private key, אשר הונפק על ידי שרת CA עליו ה-DC סומך, והוא מהווה את ה-Proof of identity עבור המשתמש. התאמתות כזו נקראית PKINIT Kerberos, או בשמה המלא Public **K**ey for **Init**ial Authentication in Kerberos.

שרת CA - Certificate Authority, שרת אשר משמש כבר-סמכא להנפקת Digital Certificates למגוון שימושים. הישויות ברשת סומכות על השרת, וה-Certificates שהוא מנפק מספקים הוכחת אותנטיות ויצירת מודל אמון ברשת. השימושים הנפוצים לכך הם בעיקר הנפקת HTTPS Certificates, או בסביבת AD - אימות משתמשים.

נפרט בהמשך יותר לעומק לגבי PKINIT Kerberos, ולגבי ההבדלים בין שיטת ההתאמתות המקורית ולבין ה-Extension של PKINIT.

כיצד קובעים אילו משתמשים יתאמתו בצורה הזו?

אכיפה של smartcard היא לא רוחבית על Domain, זו הגדרה פר משתמש. מן הסתם זה מוחל ב-scope של דומיין לרוב (או OU בחלק מהמקרים). כך זה נראה ב-GUI בממשק ה-dsa.msc:



או בשאילתת LDAP:

```
# Check if user is SmartCard enforced. if the query returns
output,
# it is enforced. (Can append! before userAccountControl).
$results
=([adsisearcher"(&(objectCategory=user)(userAccountControl:1.2.84
0.113556.1.4.803:=262144))").FindAll()
```

תזכורת על Kerberos

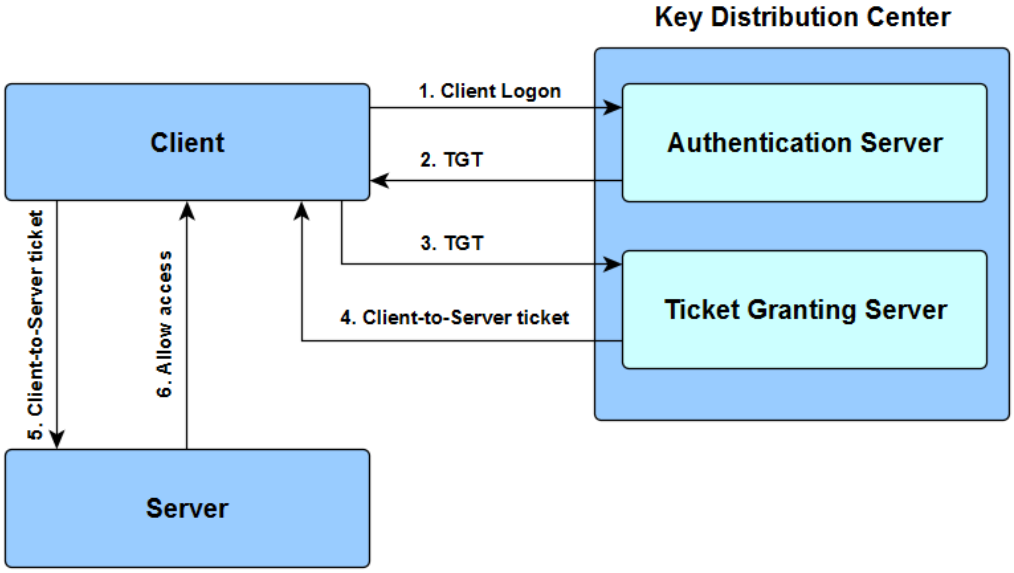
בכל זאת נעשה brief קצרצר על Kerberos לטובת רענון הזכרון. למי שפחות מכיר את הפרוטוקול ואת תהליך ההתאמתות לדומיין בעזרתו, אמליץ לקרוא מהקישור הבא. ומי שמרגיש ממש בקיא - יכול לדלג.

מי השחקנים שלנו? **הלקוח, שרת היעד, וה-KDC.**

Key Distribution Center הוא השירות שפועל על ה-DC ומטפל בבקשות ה-Kerberos שמגיעות. ה-KDC מכיל שני תתי שירותים שאחראיים לטפל בחלקים שונים בתהליך ההזדהות והגישה למשאב.

- **AS** - Authentication Server, כשמו כן הוא, אחראי על החלק הראשון של התהליך. האימות על מנת להבין אם מדובר במשתמש, או במתחזה. זה יושג לרוב על ידי shared secret (סיסמא).
- **TGS** - Ticket Granting Service, תת השירות שאחראי על חלוקת tickets לשירותים ספציפיים. כלומר, אחרי שהמשתמש מאומת, הוא מבקש ticket שמעניק לו גישה ל-service הרצוי.

התהליך עצמו מחולק לשלושה חלקים של Request-Response, ונפרט אותו:



- **KRB_AS_REQ** - זו הבקשה שהלקוח שולח לשרת במטרה שהוא יאמת אותו. הוכחת הזהות תימצא בשדה pre-authentication, בדמות Timestamp מוצפן בסיסמת המשתמש. מטרתו היא לספק הוכחת זהות, וגם למנוע התקפות replay, שכן ערך ה-Timestamp צריך להיות ב-5 דקות האחרונות.
- **KRB_AS_REP** - לאחר הבקשה, ה-DC ילך ויחפש את הרשומה של המשתמש ב-ntds.dit, וינסה לפענח את ה-authenticator (כלומר ה-Timestamp) עם ה-hash של הסיסמא השייכת למשתמש. אם זה מצליח, השרת עובד על להכין את התשובה. הוא מג'נרט session key לשיחה הנוכחית, ואת ה-PAC (מבנה שמכיל פרטים נוספים על המשתמש, יורחב בהמשך). **ה-DC מחזיר שני דברים - ה-TGT** שבו

יאוחסן ה-session key וה-PAC (מוצפן ב-hash של הסיסמא של krbtgt) והחלק השני הוא **ה-TGS session key**, מוצפן בסיסמא של המשתמש.

- **KRB_TGS_REQ** - נקבל מהשלב הקודם את את ה-TGS session key והוא יישמר יחד עם ה-TGT, והם ישמשו אותנו לבקשת שירותים. כשאנחנו מצוידים ב-TGT, נפנה ל-TGS ונבקש שירות מסוים בdomain. השירות הזה מסומן עם SPN (Service Principal Name). נבקש את ה-SPN בצירוף ה-TGT, ונצרף authenticator שמוצפן ב-TGS session key, כדי להוכיח שיש לנו את ה-session key.
- **KRB_TGS_REP** - ה-DC מפענח את ה-TGT בעזרת krbtgt, ומוציא משם את ה-session key ואת ה-PAC. בעזרת ה-session key הוא מאמת את ה-authenticator. אם הכל כשורה, סימן שבאמת מי שמדבר עם ה-DC מחזיק את ה-TGT וה-session key. בשלב הזה ה-DC מחפש את ה-SPN על מנת ליצור לו **service ticket** מוצפן בסיסמא של השרת שאליו נרצה לגשת. **ביחד איתו ה-DC יעביר את ה-server session key** מוצפן ב-TGS session key. כאן נגמר השיח של הלקוח וה-DC.

בהתחברות רגילה, המקרה מעט שונה, כי ה"שירות" שנבקש הוא העמדה שלנו (SPN host לעמדה שלנו).

ה-service session key נכנס ל-service ticket ביחד עם ה-PAC של המשתמש שניגש, וכל זה כאמור מוצפן בסיסמא של השרת אליו ניגשים (במקרה של התחברות, כלומר host, זה יוצפן ב-computer account password שלו).

- **KRB_AP_REQ** - לאחר שהלקוח מפענח את ה-server session key ושומר אותו ביחד עם ה-service ticket, הוא יכול לגשת ל-Server. השלב הזה מאוד דומה ל-TGS_REQ. השרת יקבל את ה-service ticket, וגם authenticator מוצפן ב-server session key. הוא יפענח את ה-service ticket שאמור להיות קריא אך ורק לו (ול-DC), ויימצא שם server session key. איתו הוא יאמת את ה-authenticator. אם הצליח, אז הוא יודע שהגישה לגיטימית ושהלקוח עבר ב-DC. הוא ייקח את ה-PAC מה-Ticket ועל פיו יחליט אם תינתן גישה על פי ההרשאות.
- **KRB_AP_REP** - שלב שהוא אופציונאלי, נועד על מנת לתת ללקוח לאמת את השרת, ולהבין שהוא ניגש לשרת ולא סתם למתחזה. זה סך הכל authenticator שנשלח חזרה ללקוח, והוא יאמת אותו עם ה-server session key. על פי האימות הזה הלקוח יבין אם השרת הצליח לפתוח את ה-service ticket, ועל פי זה - אם הוא אינו מתחזה.

איך PKINIT Kerberos עובד?

לאחר שנזכרנו בצורה האלגנטית שבה Kerberos עובד, נסתכל על ההבדלים בין Kebreros הרגיל, לבין ה-extension שבו אנחנו עוסקים בו היום: PKINIT Kerberos.

נתחיל בדגש חשוב: **ההבדל קיים אך ורק בשלב ההתחברות הראשונית.** לאחר ההתחברות הראשונית, NTLM יקרה בדיוק באותה צורה ו-Kerberos יקרה עם ה-TGT, לא משנה איך קיבלו את ה-TGT במקור.

חשוב לזכור ש-Kerberos הוא פרוטוקול שלא תומך בהצפנה א-סימטרית בצורת העבודה הבסיסית שלו. כלומר, הוא עובד בעזרת הצפנה סימטרית על בסיס ה-hash של הסיסמא והחלפת מפתחות שישמשו כ-session key. לכן, יש כמה התאמות שצריכות להתבצע על מנת שהוא יידע לעבוד עם Public/Private Keys. ההתאמה הזו באה לידי ביטוי ב-extension בשם PKINIT, אשר מאפשר למשתמש לשלוח סוג אחר של ערך בשדה Pre-Authentication ב-AS-REQ, והרי זהו הערך שמשמש כ-Proof of Identity למשתמש.

הצורה בה Kerberos עובד נשענת על העובדה שקיים shared secret בין המשתמש שמתחבר לבין ה-DC, וככה בעקיפין לשני הצדדים יש וריפיקציה על הזהות של משנהו. כלומר עצם ידיעת הסוד והצלחת ההתחברות מהווה אישור לכך שהלקוח הוא באמת הלקוח, וה-server הוא באמת ה-KDC. האימות ההדדי הזה קורה גם בשלב ה-KRB_AP (שלב 5-6) אם נשלחת בנוסף גם פקטת ה-KRB_AP_REP (כזכור, שלב אופציונלי). בגלל שבמקרה שלנו מדובר על התחברות ללא סיסמא, נוצרת כאן בעיה, ואין סוד משותף. אז איך נוכיח ל-DC את זהותנו?

ב-Kerberos רגיל - יישלח בשדה Pre-Authentication ב-AS-REQ את חותמת הזמן הנוכחי מוצפנת ב-hash של סיסמת המשתמש.

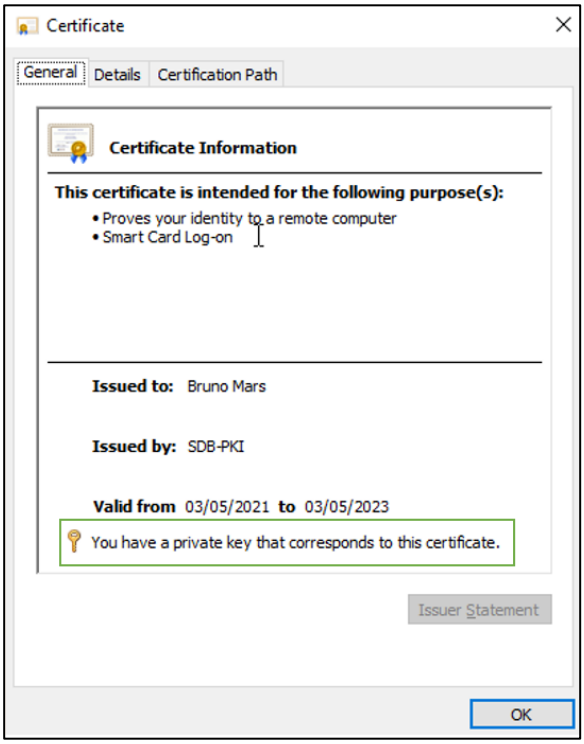
בהתחברות PKINIT - בשדה Pre-Authentication ב-AS-REQ נשלח ערך מסוג PA-PK-AS-REQ אשר מכיל את ה-Proof of Identity הקריפטוגרפית. איך זה עובד? למשתמש יש שני דברים לטובת ההתאמתות:

- Certificate שבו מוכל שם המשתמש והמפתח הציבורי שלו. ה-Certificate הוא ההוכחה שקיבלת מה-CA לזהות שלך.
- מפתח פרטי (התואם למפתח הציבורי שבתוך ה-Certificate), שאמור להיות סודי לחלוטין! המפתח הפרטי משמש להוכיח שאתה הוא המחזיק החוקי של ה-Certificate הנ"ל, ואתה אכן קיבלת אותו מה-CA המנפיק.

אז איך נוכיח ל-DC את זהותנו? נשלח ב-Pre-Auth את ה-Certificate, ואיזשהו Timestamp מוצפן במפתח הפרטי (מגן כך על Replay attacks). ה-DC יקבל את שני הדברים האלו, ויוודא את הדברים הבאים:

- יוודא את ה-Certificate בעזרת רשימת של CA Public keys שהוא מחזיק. הרשימה הזו היא המפתחות הציבוריים של כל שרתי ה-CA אשר ה-DC סומך עליהם, לכן הוא ינסה לאמת את ה-Certificate בעזרת אחד מאלה.
- לאחר מכן הוא לוקח את המפתח הציבורי שכתוב בתוך ה-Certificate, ואיתו הוא מוודא את החתימה של המשתמש (ה-Timestamp המוצפן). כך בעצם ה-DC מוודא שלמשתמש יש את המפתח הפרטי.
- אם הבדיקות עברו בהצלחה, וזמן הבקשה תואם לזמן הנוכחי, ה-DC יג'נרט TGT למשתמש (על פי שם המשתמש הכתוב ב-Certificate) ויחזיר לנו.

דוגמא ל-Certificate שמשומש להתחברות:



[**זכרו!** ה-Certificate לבדו לא יכול להיות משומש להתחברות, דרוש המפתח הפרטי בנוסף]

אוקיי, אז הבנו שב-Kerberos PKINIT המשתמש מוכיח את זהותו באמצעות Certificate ומפתח פרטי שמקושר אליו. אבל כיצד שני אלו נשמרים בצורה בטוחה, לא ניתנת לגניבה, ושיהיה ניתן להתחבר באמצעותם לכל מחשב? הרי סיסמא זוכרים בראש, אבל Certificate פחות. פה נכנס ה-Smartcard.

ה-smartcard הוא רכיב חומרתי קטן, שמכיל בתוכו Certificate בפורמט x.509 ומפתחות פרטיים בצורה שלא ניתן לשכתוב וקריאה. בנוסף, הוא מסוגל לבצע פעילויות קריפטוגרפיות בסיסיות כך ש**המפתח הפרטי אף פעם לא מגיע לעמדה**, אלא מערכת ההפעלה מתממשקת באמצעות ה-API של ה-Smartcard על מנת לבקש שיבצע פעולות באמצעות המפתח הפרטי, והוא מבצע ומחזיר את התוצאה בלבד.

אבל ה-Smartcard לא פראייר, הוא ייתן את השירות שלו רק אחרי שנספק לו PIN או זיהוי ביומטרי כלשהו. לאחר שהכנסנו pincode\מידע ביומטרי ואלידי, תישלח בקשת AS_REQ מסוג PA-PK-AS-REP אשר ב--Pre auth שלה נמצא ה-Certificate, וחתימה עם ה-Private key שתואם ל-Certificate (שוב, לכל Certificate, כתוב בתוכו המפתח הציבורי. לעומת זאת, המפתח הפרטי שמור על ה-Smartcard). מכאן והלאה מתרחש תהליך ה-Kerberos בצורה סטנדרטית.

רגע, אז אין יותר NTHashes?

לאור ההבנה שצברנו על PKINIT Kerberos, ננסה להבין כיצד להשיג אחיזה על משתמש. מנקודת מבט שטחית של תוקף, כאשר משתמשים **בהתאמתות רגילה**, אם השגת את ה-hash או את הסיסמא עצמה - ניצחת. כאן ב-smartcard זה קצת יותר מורכב, אבל יש גם דברים שמשחקים לטובתנו.

על מנת לענות על השאלה לעיל, נצטרך להיזכר בפרוטוקול המקביל ל-Kerberos, NTLM הישן וה"טוב".

ב-NTLM רגיל מתבססים על ה-Hash של הסיסמא לטובת חישוב Response ל-Challange שקיבלנו, וזוהי הוכחת הזהות שלנו. אך כעת אין לנו סיסמא, רק Certificate ומפתח פרטי. **האם קיים איזשהו PKINIT NTLM לצורך העניין? התשובה לכך היא לא.**

אז ניתן לחשוב שלא ניתן להתאמת עם NTLM בכלל עם משתמש אכוף. זה כמובן לא נכון. לטובת תאימות לאחור ותמיכה אל מול מערכות שלא תומכות ב-Kerberos, **גם משתמשי smartcard אמורים לדעת לדבר ב-NTLM**, וזה באמת קורה.

הצורה שזה עובד היא שה-KDC משייך בכל זאת NTHash לכל משתמש אכוף שאיתו הוא יוכל להתאמת ב-NTLM. אפילו נגדיל ונאמר, ה-NThash הספציפי הזה ניתן לשימוש אך ורק ל-NTLM ולא ל-Kerberos! ההתחברות הKerberos-ית עובדת עם PKINIT ועם Certificates בלבד.

נבין מהו ה-Hash הזה. ה-NTHash המסורתי הוא פשוט MD4 של סיסמת ה-Plaintext שלך. אבל כרגע אין לנו באמת סיסמא רגילה, אבל עדיין יש NTHash - אז איך הוא נקבע? מסתבר שהוא **נקבע רנדומלית** בכל השמת אכיפה של משתמש (ה-Checkbox בתמונה המצורפת בתחילת המאמר). כלומר, אכפת משתמש - ה-KDC מגריל לו NTHash על מנת שיוכל לתמוך גם ב-NTLM. הורדת אכיפה והחזרת - התגלגל.

נראה כיצד לאותו משתמש יש NTHash-ים שונים, כאשר הדבר היחיד שעשינו בין לקיחת שתי התמונות האלה הוא הורדת והחזרת האכיפה:

```
Object RDN           : Avril Lavigne

** SAM ACCOUNT **

SAM Username         : girlfriend
User Principal Name  : girlfriend@SDB.secret
Account Type         : 30000000 ( USER_OBJECT )
User Account Control : 00050200 ( NORMAL_ACCOUNT DONT_EXPIRE_PASSWD SMARTCARD_REQUIRED )
Account expiration   :
Password last change : 12/24/2020 3:39:31 PM
Object Security ID   : S-1-5-21-3887654849-150919805-3624264558-5437
Object Relative ID   : 5437

Credentials:
  Hash NTLM: ca93806ba1809a3689ab2344b10c11f1
```

אחרי:

```
Object RDN           : Avril Lavigne

** SAM ACCOUNT **

SAM Username         : girlfriend
User Principal Name  : girlfriend@SDB.secret
Account Type         : 30000000 ( USER_OBJECT )
User Account Control : 00050200 ( NORMAL_ACCOUNT DONT_EXPIRE_PASSWD SMARTCARD_REQUIRED )
Account expiration   :
Password last change : 12/24/2020 3:39:31 PM
Object Security ID   : S-1-5-21-3887654849-150919805-3624264558-5437
Object Relative ID   : 5437

Credentials:
  Hash NTLM: aab6ad73b6465db9b92209861b20c14c
```

דוגמא קטנה לאילו פרטים ה-DC שומר על משתמש כשהוא לא אכוף, לעומת כשהוא כן אכוף:

לפני אכיפת Smartcard, ניתן לראות שיש למשתמש NTHash, LMHash, AES Keys לטובת התחברות Kerberos-ית רגילה ו-NTLM-ית.

בצורה דיפולטית, ה-Hash של משתמש אכוף לא משתנה כל עוד אקטיבית לא מגלגלים אותו\מסירים ומחזירים אכיפה, לכן הוא אפילו יותר Persistent מסיסמא רגילה אשר פגה תוקף כל כמה חודשים. המשמעות - **NTHash של משתמש אכוף הוא נכס משמעותי מאוד עבור תוקף**, אפילו יותר מהרגיל.

איך בכל זאת גורמים לHash להתגלגל? ב-Domain Functional Level 2016 ניתן להפעיל את ההגדרה ל-Rolling אוטומטי של ה-NTHash למשתמשים אשר מתאמתים עם PKINIT, בעזרת תאריך תפוגה שניתן להגדיר (דיפולטית חודשיים). אחרת, קיימים פתרונות אלטרנטיביים בדמות סקריפטים אשר מגלגלים את ה-NTHash (קישורים בסוף המאמר).

לפני:

```
Object RDN           : Avril Lavigne

** SAM ACCOUNT **

SAM Username         : girlfriend
User Principal Name  : girlfriend@SDB.secret
Account Type         : 30000000 ( USER_OBJECT )
User Account Control : 00010200 ( NORMAL_ACCOUNT DONT_EXPIRE_PASSWD )
Account expiration   :
Password last change : 12/24/2020 3:39:31 PM
Object Security ID   : S-1-5-21-3887654849-150919805-3624264558-5437
Object Relative ID   : 5437

Credentials:
  Hash NTLM: b15ae9c3693978ed0f9aa70d98ee85d9
    ntlm- 0: b15ae9c3693978ed0f9aa70d98ee85d9
    lm  - 0: 3e3f42d82705542fab4202242b49224f

Supplemental Credentials:
* Primary:NTLM-Strong-NTOWF *
    Random Value : 6b3d0a93fd577a669d06d2b7593fef6e

* Primary:Kerberos-Newer-Keys *
    Default Salt : SDB.SECRETgirlfriend
    Default Iterations : 4096
    Credentials
      aes256_hmac       (4096) : 9f997a71b60e43b3756239f8be4bacfc089402abf3f628dc80bd6b024aba4e3a
      aes128_hmac       (4096) : b8c61567da82048b461b9cf94a93605f
      des_cbc_md5       (4096) : 2c8034b5dfc87f45
```

אחרי האכיפה, יש אך ורק NTHash:

```
Object RDN           : Avril Lavigne

** SAM ACCOUNT **

SAM Username         : girlfriend
User Principal Name  : girlfriend@SDB.secret
Account Type         : 30000000 ( USER_OBJECT )
User Account Control : 00050200 ( NORMAL_ACCOUNT DONT_EXPIRE_PASSWD SMARTCARD_REQUIRED )
Account expiration   :
Password last change : 12/24/2020 3:39:31 PM
Object Security ID   : S-1-5-21-3887654849-150919805-3624264558-5437
Object Relative ID   : 5437

Credentials:
  Hash NTLM: ca93806ba1809a3689ab2344b10c11f1
```

אחרי שהבנו את משמעות ה-NTHash לתוקף במקרה של התאמתות smartcard, נדבר קצת על איך נשיג אותו ☺

אמ;לק - הוצאת NTHash ללא נגיעה ב-LSASS למשתמש אכוף Smartcard

אז הכל טוב ויפה ומעניין, אבל באנו לעשות בלאגן. נציג כעת טכניקה אשר מאפשרת להוציא NTHash של קורבן, מבלי לגעת בתהליך ה-LSASS שלו. נניח ותוקף מגיע לעמדה של משתמש אכוף smartcard, אשר מוחלות שם הגנות כמו Credentials guard ו/או LSA Protection אשר לא מאפשרות לנו לגעת ב-LSASS. השיטה הזו מספקת מעקף שקט ואלגנטי בצורה תקשורתית בלבד אשר מנצלת את הצורה בה PKINIT Kerberos עובד ואחרי שנבצע אותה, נחזיק ב-NTHash.

ה-NTHash, כפי שהסברנו מוקדם יותר במאמר, שווה ערך לאחיזה מוחלטת על המשתמש עד הגלגול הבא (פחות או יותר ☺). זה מקרה ספציפי בו ניתן להשתמש בטכניקה שנתאר, ואתן דוגמאות לעוד Use-cases רלוונטיים מאוד. **ה-pincode של הכרטיס הכרחי לטכניקה שנדגים.**

איך הכל התחיל?

במהלך מחקרנו על SmartCards Authentication, הבנו כיצד מתבצע PKINIT Kerberos. הזדהות בעזרת מפתחות פרטיים/ציבוריים, ולמעשה ה-Certificate שנמצא על ה-smartcards מהווה בתור ה-Proof of Identity, ועל פי ההצלחה של התהליך הזה, משתמש מקבל TGT ומתחבר בהצלחה לעמדה. את כל זה אתם יודעים אם הגעתם עד לכאן. עוד דבר שאתם יודעים, הוא שגם משתמש אכוף יכול להתאמת ב-NTLM, על מנת לתמוך גם במערכות Legacy ולא לשבור תאימות. השילוב של שני אלו מותיר לנו פער מסוים מכיוון שלאחר התחברות עם ה-Smartcard, **עדיין יכולנו להוציא מהעמדה באמצעות mimikatz את ה-NTHash** (בעמדות ללא הגנות כמובן), אשר משמש להתאמתות NTLM-ית. **נדגיש פעם נוספת: ל-NTHash אין שום חלק בהתחברות PKINIT Kerberos!**

אבל איך הם הגיעו לשם?

מבולבלים? גם אנחנו. מיד נפתור את התעלומה. אז אחרי שהבנו כיצד ה-Hash-ים נקבעים, חסרה לנו הנקודה המרכזית - כיצד הם מגיעים לעמדה של המשתמש שמתחבר? אם בהתחברות רגילה ה-Provider של NTLM (שהוא כמובן מקומי) יודע לקבל את הסיסמא ולשמור את ה-Hash שלה ב-LSASS על מנת לתמוך ב-SSO, אז בהתחברות Smartcard, **לעמדה שאליה אתה מתחבר אין שום דרך לדעת את ה-NTHash שלך**. לכן נשארו רק שני הסברים הגיוניים: או שה-DC מעביר לנו את המידע הזה בתהליך התחברות ה-PKINIT Kerberos (הוא היחיד שמכיר את הסודות האלו), או שהמידע מגיע על כנפי מלאכי השרת בדרך קסם לעמדה.

מעבר ה-NTHash

ניסינו את מזלנו ב-StackExchange, וקיבלנו תשובה מאחד החבר'ה היותר ברי-סמכא בתחום, שהסביר שהאופציה הראשונה היא הנכונה. מיד נסביר כיצד זה קורה, הקישור לשאלה ולתשובה המלאה יצורף בסוף המאמר.

בתמצות - בתהליך ה-PKINIT Kerberos, בתוך ה-PAC של ה-Service ticket (בהרחבה בהמשך), עובר ה-NTHash. ועל איזה Service ticket מדובר? ה-Host ticket כמובן!

רלוונטי אך ורק ל-PKINIT Kerberos: ב-Kerberos רגיל אין צורך להעביר את ה-NTHash והוא אינו עובר.

Host Ticket?

נזכיר מה הוא ה-Host ticket, ולמה זה הדבר שמאפשר את קיומה של המתקפה הזו. ה-Host ticket הוא ticket דיפולטי, אין לו SPN רשום, ולמעשה הוא ה-service ticket שאנחנו מבקשים לעמדה מסוימת בעת התחברות אינטראקטיבית. וזאת למעשה, מעולה, כי **הוא מוצפן עם ה-Computer Account Password** של אותה העמדה. ומה שמאפשר את השיטה הזו, הוא שהמחשב שאליו יבוקש ה-host - יכול להיות בשליטתנו. לכן נצטרך Computer Account Password של מחשב כלשהו בדומיין, ואז נבקש את ה-host ticket עליו.

הוצאת Computer Account Password זו טכניקה די פשוטה, שגם לא נוגעת ב-LSASS מאחר וכל הפרטים שמורים ב-Registry. ונניח שלא הצלחנו להשיג אחיזה על עמדה, אפשר להוסיף עם משתמש חלש Computer account עם Addcomputer.py של impacket:

```
hax0r@SDB-Kali-2020-3:~$ python3 /usr/share/doc/python3-
imacket/examples/addcomputer.py sdb.secret/caspit:Bb123456 -computer-
name ERAN$
Impacket v0.9.23 - C opyright 2021 SecureAuth Corporation
[*] Successfully add machine account ERAN$ with password
LV869HHqYivQA2JXrNY1ixIKWGefKqo8.
```

אנחנו יוצאים מנקודת הנחה שה-Host Service Ticket הוא על עמדה בשליטתנו, כלומר ביכולתנו לפענח. אז נבין מה בדיוק מחכה לנו אחרי הפענוח.

בונוס שלא ארחיב עליו יותר מדי: ניתן לבקש במקום Host ticket, U2U Service Ticket from itself to itself, והוא מוצפן בסיסמא של המשתמש שאיתו ביקשנו את ה-Ticket. בצורה הזו נוכל לפענח את ה-Ticket ולהשיג את מבוקשנו ללא צורך בלשלוט ב-Computer account. הקרדיט לטריק הוא לחוקר האבטחה אלעד שמיר שהציג אותו במחקר שלו בנושא.

מה זה PAC?

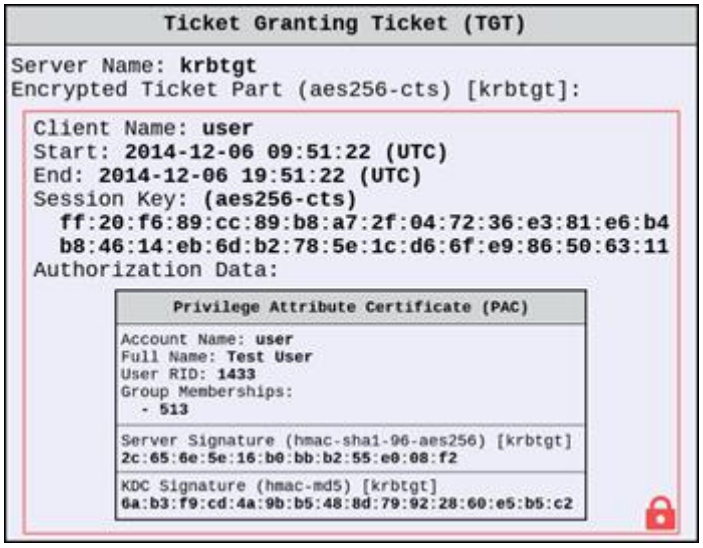
PAC, או בשמו המלא Privilege Attribute Certificate הוא מרכיב בתוך Kerberos tickets שאחראי בין היתר על ה-authorization של המשתמש. כלומר, מעבר לאימות, ה-KDC אמור לתת פרטים נוספים על המשתמש על מנת לייחד אותו ולתת לו הרשאות ספציפיות שיתאימו לו. ה-PAC מכיל מידע על המשתמש, בעיקר SID-ים, חברות בקבוצות ופריבילגיות למיניהן, כלומר כל מה שדרוש על מנת ליצור למשתמש Access Token לאחר ההזדהות.

- **Authentication** - אימות, הוכחת זהות.
- **Authorization** - פירוט הזהות של היישות, לצרכי הרשאות גישה.

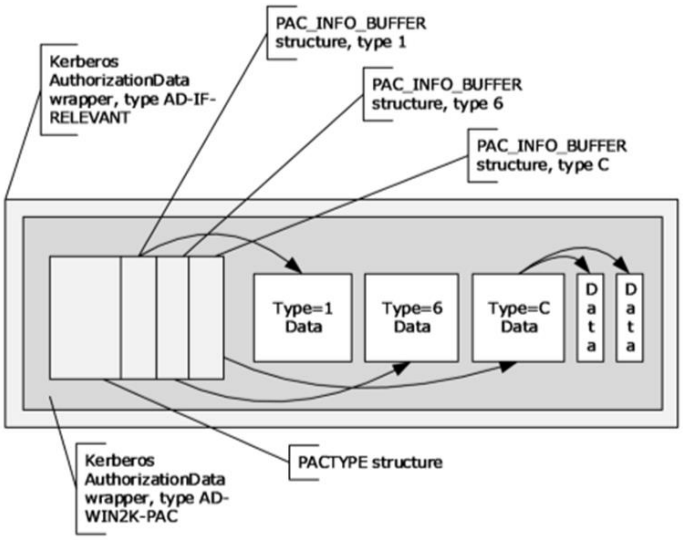
אז כל המידע הזה נארז למבנה ה-PAC, והוא ייכנס ל-TGT. למה ל-TGT? כדי שה-DC יוכל לשים את אותו ה-PAC בתוך service tickets עתידיים, וכאשר הם ייפתחו אצל שרת קבצים שאליו ניגשנו לצורך העניין, יהיה לו את ה-PAC כדי ליצור Access Token.

בנוסף לפרטים האלה, ב-PAC גם כתוב Additional Credential Information (זכרו אותו), ועוד כמה דברים לא רלוונטיים.

בתמונה ניתן לראות אחלה המחשה, TGT, שבו החלק המוצפן מכיל את הפרטים על המשתמש, כלומר בין היתר את ה-PAC, וגם חתימות על ה-PAC:



בפועל ה-PAC נראה הרבה יותר כמו בתמונה. הוא מכיל Struct מסוג של AD-IF-RELEVANT, שמכיל Struct בשם AD-WIN2K-PAC, שהם מתפקדים בתור ה-wrappers, ומה שנמצא בתוכם הם המבנים המעניינים:



המבנה PACTYPE הוא ה-Header של ה-PAC, ולאחריו מגיעים מיד מבני ה-PAC_INFO_BUFFER אשר מסודרים במערך, כשכל אחד מהם משמש כמצביע ל-buffer וגם מגדיר מהו ה-data type של אותו ה-buffer על מנת לאפשר ל-PAC להכיל סוגים שונים של מידע.

- **PACTYPE** - משמש בתור Header לקראת ה-PAC_INFO_BUFFER, ואין בו יותר מדי. ה-cBuffers מצביע על מספר ה-PAC_INFO_BUFFER, וה-buffers הוא כבר מערך של PAC_INFO_BUFFER-ים, שמיד נפרט עליהם.
- **PAC_INFO_BUFFER-** משמש כ-Header של Info Buffer. מכיל את גודל ה-buffer ואת ה-offset שלו, והכי חשוב, מכיל את ה-buffer type, שמתאר מהו סוג המידע שיש ב-Buffer.

בתמונה ניתן לראות את את הסוגים המעניינים, 0x1 פירושו מידע שדרוש לזיהוי משתמש (SID-ים וכו'), 0x2 פירושו additional credential information ו-0x6-0x7 פירושם Checksum וכן הלאה.

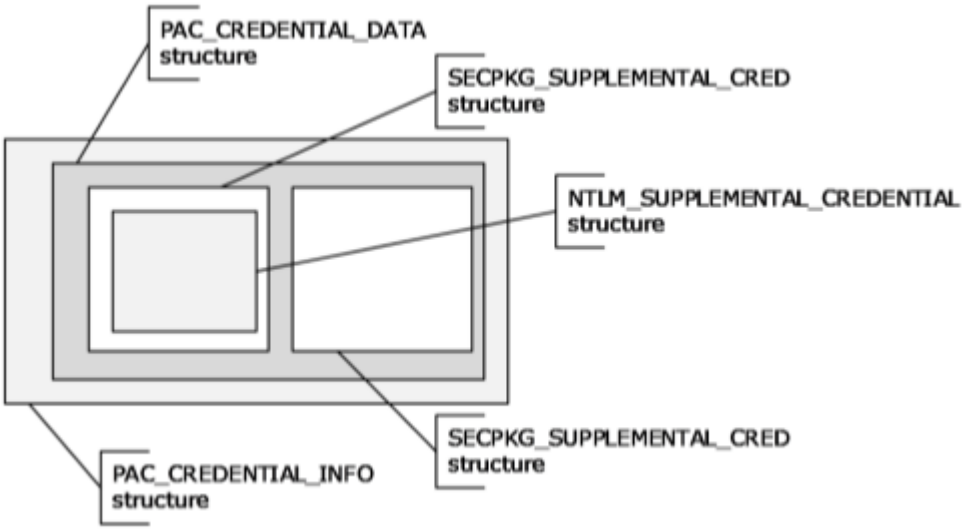
Value	Meaning
0x00000001	Logon information (section 2.5). P Additional logon information buffe
0x00000002	Credentials information (section 2 buffer of this type, based on const credentials information buffers MU
0x00000006	Server checksum (section 2.8). PA Additional logon server checksum
0x00000007	KDC (privilege server) checksum (this type. Additional KDC checksu

מקווה שכל הנורות שלך נדלקו בשלב זה, אבל גם לא, נחזור אחורה במאמר, וניזכר למה בכלל אנחנו חופרים בכל המבנים הסבוכים האלה של ה-PAC. זאת בגלל שאנחנו רוצים למצוא את **ה-PAC Buffer שיכיל**

את ה-NTHash הנכסף אשר מובטח לנו. או בפועל, נצטרך לרוץ על כל ה-PAC_INFO_BUFFER-ים עד שנמצא את האחד שמצביע על Buffer עם Type 0x2 אשר מכיל **PAC Credentials**.

PAC Credentials

המבנה הזה ב-PAC מיועד להעברת Credentials שמתאימים לפרוטוקולים אלטרנטיביים (סתם זורק, NTLM) מאלו ששומשו בהתחברות ההתחלתית (Kerberos PKINIT). נכמר ליבנו כשגילינו שבגלל שהוא מכיל פרטים מאוד רגישים, הוא גם מוצפן בפני עצמו בתוך ה-Ticket המוצפן! אנחנו נתקל את זה בהמשך.



- **PAC_CREDENTIAL_INFO** - המבנה החיצוני, המשמש כאן בתור ה-Header (אינו מוצפן), אשר מכיל את סוג ההצפנה ואת שאר ה-Data המוצפן. סוגי ההצפנה הנפוצים הם AES256_CTS_HMAC_SHA1_96 ו-RC4. הערכים שלהם הם בהתאמה 0x12 ו-0x17. ה-Data המוצפן הוא כבר המבנה PAC_CREDENTIAL_DATA.
- **PAC_CREDENTIAL_DATA** (מוצפן) - מכיל מערך של כל ה-SECPKG_SUPPLEMENTAL_CRED ואת מספר האיברים (סוגי ה-Creds) במערך.
- **SECPKG_SUPPLEMENTAL_CRED** - מכיל את ה-Package name שאליו משויך ה-Credential, ומצביע (וגודל) ל-Credentials עצמם.
- **NTLM_SUPPLEMENTAL_CREDENTIAL** - כשמו כן הוא - מכיל Credentials נוספים עבור התאמתות NTLM-ית. בול פגיעה . מפה ניקח את ה-NtPassword וסיימנו! זה ה-NTHash הנחשק.

```
typedef struct _NTLM_SUPPLEMENTAL_CREDENTIAL {
  ULONG Version;
  ULONG Flags;
  BYTE LmPassword[16];
  BYTE NtPassword[16];
} NTLM_SUPPLEMENTAL_CREDENTIAL,
 *PNTLM_SUPPLEMENTAL_CREDENTIAL;
```

כיצד מפענחים את ה-PAC_CREDENTIAL_DATA?

חוב שיש ממקודם: בנוסף לכך שה-PAC מוצפן (בכללי בתוך ה-Ticket), ה-PAC_CREDENTIAL_DATA מוצפן בפני עצמו בתוך ה-PAC. נשאלת השאלה, במה ה-Ticket Granting Service (שנמצא על ה-DC כמובן) יכול להצפין את זה כך שרק הלקוח יוכל לפענח? (הרי לא קיים shared secret בינינו לבין ה-DC). אז מסתבר שיש מפתח Diffie Helman שמוחלף במהלך ה-AS REQ-REP, והוא נקרא ה-**AS Replay Key**, והוא משמש להצפנת ה-PAC_CREDENTIAL_DATA בתוך ה-Service ticket.

מכיוון שהעברת ה-NTHash רלוונטית רק להתחברות אינטראקטיבית, אז בקשת ה-Host service ticket תתרחש מיד אחרי סיום ה-AS. כלומר, הלקוח ישמור בצד את ה-DH Key, ויידע לפענח את ה-PAC_CREDENTIAL_DATA מיד בקבלת ה-TGS-REP, ישמור בזכרון את כל ה-Creds שחשובים ויזרוק את ה-DH Key לפח. **כלומר, לתפוס Host Service Ticket אינו מספיק בפני עצמו!**

מכאן משתמע שלביצוע המתקפה נצטרך ליזום בעצמנו את בקשת ה-TGT גם כן, על מנת להחליף ב-Diffie Hellman בעצמנו AS Replay Key, לבקש עם ה-TGT שקיבלנו את ה-Host Service Ticket ל-Host שיש לנו אחיזה עליו, ומשם נוכל לפענח גם את ה-Ticket וגם את ה-PAC_CREDENTIAL_DATA שבתוך ה-PAC שלו.

בשלב זה, מה שנותר לעשות הוא להוסיף פקודת הדפסה פשוטה לפונקציה askTGT של Rubeus שתדפיס את ה-AS Replay Key, אותו נשמור בצד:

```
key =
pkAsReq.Agreement.GenerateKey(pkAsRep.DHRepInfo.KDCDHKeyInfo.SubjectPubl
icKey.DepadLeft(),new byte[0], pkAsRep.DHRepInfo.ServerDHNonce,
GetKeySize(etype));

 //OFIR ADDED THIS
 Console.WriteLine("\r\n[*] AS-Reply Key: {0}\r\n",
Convert.ToBase64String(key));
```

דוגמא מלאה

כן, זה הרבה מבנים והרבה מונחים, אז הכי טוב להבין עם דוגמא פרקטית. קודם כל, אנחנו חייבים לבקש TGT עם askTGT, בעזרת הגרסה הערוכה של Rubeus ששומרת בצד את ה-AS-Replay key. הוא יחזור מאוחר יותר. עם ה-TGT נבקש TGS, שהוא לא אחר מ-Host service ticket ל-host לבחירתנו, אליו יש לנו את ה-Computer Account Password, ויאללה מתחילים.

לקסם הבא נשתמש ב-decryptKerbTicket.py היישר מ-Github על מנת לפענח את ה-Host Service Ticket שקיבלנו, כאשר אנחנו מעבירים את ה-Ticket בפורמט kirbi ואת הסיסמא ל-Computer account אליו התבקש ה-Ticket:

```
python.exe decryptKerbTicket.py -t "C:\Users\Eran\Desktop\Anderson PAC Example\ho
st_1803pc.kirbi" -k cadeeb097d19c97c7f4c930b5546ece54b0cba72608297d0f05be7394a558
b27
        Client: caspit@SDB.SECRET
        Server: HOST/SDB-win1803-pc.sdb.secret@SDB.SECRET
[+] DECRYPTING TICKET
```

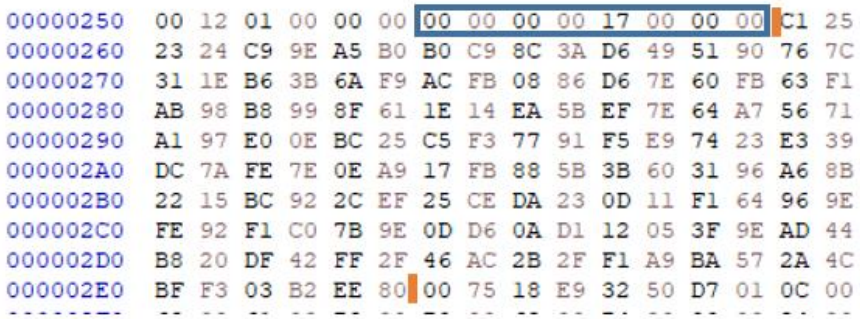
נפענח (מנסיוני decryptKerbTicket.py דרש Python2.7 ו-impacket 9.18 כדי לעבוד חלק):

נשלוף את ה-Ticket המפוענח מהפלט ונמשיך לעבוד איתו. ניתן לראות ב-hxd את חלקי מבני ה-PAC ואת ערכיהם:

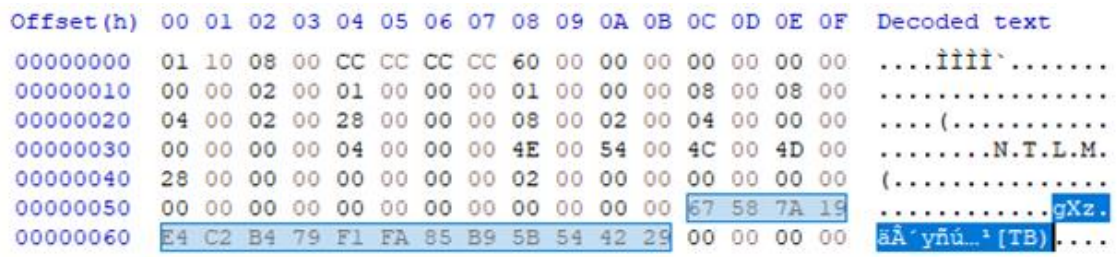
- **שחור** - ה-PACTYPE (ניתן לראות בו את שני השדות הראשונים, מספר ה-PAC_INFO_BUFFER שהוא 6, וה-Version שהוא 0)
- לאחר מכן יש מערך של כל ששת ה-PAC_INFO_BUFFER, סימנתי רק את שני הראשונים.
- **אדום** - ה-PAC_INFO_BUFFER של ה-KERB_VALIDATION_INFO (סוג 1, גודל 0x01D8, אופסט 0x68). מכיל SID-ים וכו'.
- **כחול** - ה-PAC_INFO_BUFFER של ה-PAC_CREDENTIAL_INFO (סוג 2, גודל 0x90, אופסט 0x240), מכיל את מה שלשמו התכנסנו.

```
00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E
30 82 03 72 30 82 03 6E A0 04 02 02 00 80 A1
82 03 64 04 82 03 60 06 00 00 00 00 00 00 00
01 00 00 00 D8 01 00 00 68 00 00 00 00 00 00
00 02 00 00 00 90 00 00 00 40 02 00 00 00 00
00 00 0A 00 00 00 16 00 00 00 D0 02 00 00 00
00 00 00 0C 00 00 00 50 00 00 00 E8 02 00 00
00 00 00 00 06 00 00 00 10 00 00 00 38 03 00
00 00 00 00 00 07 00 00 00 14 00 00 00 48 03
00 00 00 00 00 00 01 10 08 00 CC CC CC CC C8
01 00 00 00 00 00 00 00 00 02 00 64 1E C9 E8
31 50 D7 01 FF FF FF FF FF FF FF 7F FF FF FF
FF FF FF FF 7F 69 CE 4A 5F 7F 2A D7 01 69 8E
B4 89 48 2B D7 01 FF FF FF FF FF FF FF 7F 0C
00 0C 00 04 00 02 00 14 00 14 00 08 00 02 00
00 00 00 00 0C 00 02 00 00 00 00 00 10 00 02
00 00 00 00 00 14 00 02 00 00 00 00 00 18 00
```

נלך ל-offset של 0x240 וניקח שם buffer של 0x90:



ניתן לראות את ה-Version - 0x00000000, את ה-EncryptionType - 0x00000017, כלומר סוג הצפנה RC4. לאחר מכן, כל מה שמתוחם בין הכתומים הוא ה-blob המוצפן, הלא הוא PAC_CREDENTIAL_DATA. נפענח עם סקריפט פייתון פשוט מבוסס Impacket, כאשר מפתח בפענוח הוא AS-Replay key ששמרנו בצד מתחילת התהליך:



מפה יש קצת encoding שלא נתעכב עליו, אבל ניתן לראות ב-hex את ה-NTLM - package name, ומה שמסומן, הוא **ה-NTHash הנכסף.**

כל הקבצים הבינאריים לאורך ההדגמה, וסקריפט הממשש את המתקפה, יימצא ב-Repo הבא ב-Github: https://github.com/e99ran/Anderson_PAC.py

Use-Cases נוספים

נוכל להשתמש בטכניקה שפירטנו עליה כעת בכל תרחיש שבו נצטרך לתקוף התאמתות PKINIT Kerberos, וזה כמובן קורה בעוד מקרים מלבד משתמש שמחבר smartcard.

לא ממזמן חוקר אבטחה ישראלי בשם אלעד שמיר הוציא כלי שמאפשר השגת אחיזה על משתמש בהינתן הרשאות לכתוב לו ל-AD Attributes. בקצרה - זו שיטה שממנפת את המנגנון Windows hello, שמאפשר הזדהות ביומטרית בסביבה בה אין CA. המנגנון עובד בצורה כזו שנשמר למשתמש מפתח ציבורי, התואם

למפתח פרטי שנשמר אצל המשתמש, בצורה שמאוד מזכירה הזדהות של SSH באמצעות Key-Pair, ומייתרת את הצורך ב-Certificate. ההזדהות, אם תהיתם כמובן מתרחשת בעזרת PKINIT Kerberos, ובעזרת המתקפה שתוארה במאמר זה נוכל להוציא בכל רגע נתון את ה-NTHash של המשתמש, גם אם הוא משנה סיסמא!

מקרה נוסף הוא אם הצלחנו לתקוף את שרת ה-ADCS (המימוש הMicrosoftי ל-CA בסביבה דומיינית), ולקבל Certificate למשתמש שהוא בדומיין. נוכל להשתמש ב-Ceritificate הזה על מנת לקבל את ה-NTHash שלו בכל רגע נתון. דוגמא לתקיפה על NTLM Relay -ADCS מה-DC לממשק ה-HTTP של ה-ADCS לטובת קבלת Certificate לבחירתנו.

לקריאה נוספת ומרחיבה:

הכלי שמממש את הטכניקה הראשונה של אלעד שמיר:

https://github.com/eladshamir/Whisker

whitepaper בנושא ADCS ומתקפות עליו אשר יצא ממש לפני חודשים ספורים:

https://www.specterops.io/assets/resources/Certified_Pre-Owned.pdf

כלי שמממש את הנפקת ה-Certificate בעזרת NTLM Relay:

https://github.com/bats3c/ADCSPwn

עובדת בונוס - לאחר סיום המחקר, גיליתי שבנג'מין דלפי שילב את זה ב-Kekeo לפני 4 שנים, ולא טרח לכתוב את זה באף מקום (באמת חשבתם אחרת?) מלבד בציוץ הבא:

https://twitter.com/gentilkiwi/status/826932815518371841?lang=he

סיכום

דיברנו במאמר על טכניקה מתקדמת להוצאת NTHash ללא נגיעה בזכרון, על הUse-caseים שלה ועל הסיבות שהיא אפשרית.

השיטה הזו נותרה מתחת לרדאר בשנים האחרונות, וכשביצעתי את המחקר לא היה עליה כמעט תיעוד. מאז תחום ה-Certificates ב-Active-Directory תפס תאוצה (ונהיה רלוונטי יותר ויותר), והתפרסמו כמה מחקרים וכלים בנושא. המימושים החדשים שיצאו ובהם תוכלו להשתמש: PKINIT Tools של Dirkjan או בגרסה האחרונה של Rubeus, גרסא 2.0.

בכללי, תחום ההתאמתות בעזרת ADCS והתקיפות עליו תופס תאוצה מרשימה, וייתכן שמדובר בעתיד הקרוב של תקיפות ה-AD. מקווה שהצלחתי לתת כאן איזשהי נקודת פתיחה טובה לתחום, ואם לא, אז לפחות שהיה מעניין.

קצת עליי ותודות

ערן נחשון, בן 22, Pentester בסביבות AD וחוקר אבטחה. רוצה להודות לחברי לצוות, אופירטל, שעזר הרבה במהלך המחקר, והוא גם חבר טוב ותותח. לכל שאלה או הערה בהקשר למאמר או בכלל, שלחו לי DM ב-twitter:

https://twitter.com/NahshonEran

או במייל:

eerann9@gmail.com

מקורות

https://security.stackexchange.com/questions/246304/how-ntlm-sso-is-preformed-on-smartcard-Kerberos-logon

https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-pac/166d8064-c863-41e1-9c23-edaaa5f36962

https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-pkca/d0cf1763-3541-4008-a75f-a577fa5e8c5b

https://datatracker.ietf.org/doc/html/rfc4556.html

https://gist.github.com/xan7r/ca99181e3d45ee2042425f4f9181e614

לגבי NTHash Rolling for PKINIT Users:

https://docs.microsoft.com/en-us/windows-server/security/credentials-protection-and-management/whats-new-in-credential-protection

https://www.stigviewer.com/stig/active_directory_domain/2016-12-20/finding/V-72821