

# Digital Whisper

גליון 66, נובמבר 2015

מערכת המגזין:

מייסדים:

אפיק קסטיאל, ניר אדר

מוביל הפרויקט:

אפיק קסטיאל

עורכים:

אפיק קסטיאל

כתבים:

ניר עופר (hyprnir), עדן אלון, עידו קנר וים מסיקה.

יש לראות בכל האמור במגזין Digital Whisper מידע כללי בלבד. כל פעולה שנעשית על פי המידע והפרטים האמורים במגזין Digital Whisper הינה על אחריות הקורא בלבד. בשום מקרה בעלי Digital Whisper /או הכותבים השונים אינם אחראים בשום צורה ואופן לתוצאות השימוש במידע המובא במגזין. עשיית שימוש במידע המובא במגזין הינה על אחריותו של הקורא בלבד.

פניות, תגובות, כתבות וכל הערה אחרת - נא לשלוח אל [editor@digitalwhisper.co.il](mailto:editor@digitalwhisper.co.il)

---

## דבר העורכים

---

ברוכים הבאים לגליון ה-66 של Digital Whisper!  
אז מה שלומכם? אנחנו מקווים שלמרות המצב - הכל אצלכם בסדר גמור, שיש לכם חשמל ושהרכב שלכם לא מוצף מים.

אף פעם לא הצלחתי להבין את הדחף של התעשייה להצמיד לכל מיני מוצרים כתובות IP, כאילו, בטח שאני מבין את הפוטנציאל שבדבר, הרי כל אחד יכול לדמיין את הסיטואציה הבאה:

אתה יושב על הספה, ופתאום מכה בך הרעב לחביתת גזר. כדרך-אגב אתה פותח בסמארטפון את אפליקציית הניהול של הבית החכם, גולש לפורטל המרכזי, משם לעמוד המטבח וממנו למקרר, שומע את עצמך אומר "מצרכים לחביתת גזר" ולאחר כשתי שניות מקבל הודעה שגיאה שאומרת "לא נמצא גזר, האם ברצונך להזמין גזר או לחפש מתכוני חביתה אחרים, שאנשים שאוהבים חביתת גזר סימנו עליהם Like".

בלי לחשוב פעמיים, אתה לוחץ על "הזמן גזר - הפרופיל הרגיל", וכעבור כ-2 דקות אתה מקבל SMS ממרכז הפצה איזורי שמודה לך על השימוש בשירותיו ושמח לבשר לך ש-Quadcopter עושה דרכו אליך ברגעים אלו ממנו ממש, ובעוד 7 דקות המשלוח יגיע אל נקודת ה-GPS שסופקה על-ידי האפליקציה. ובנוסף שולח לך URL שמאפשר לך לראות בזמן אמת את הנ"צ של הגזר שלך מתעופף ברחבי העיר.

לאחר 7 דקות, אתה מקבל SMS שמבקש ממך לבדוק את התקן הנחיתה שהתקנת לפני חודש ונגיש מהחלון במטבח, אתה ניגש לשם ואכן רואים חבילת גזרים מונחת ו-Quadcopter עושה דרכו חזרה למחסני החברה.

אתה מוציא גזר אחד מהשקית ושם אותו בפורס גזר (שבמידי דואג לשלוח לך מייל על כך שהאחריות על אחת הסכינים שלו פגה בעוד כשבועיים), חיישן ה-RFID שמותקן במקרר שלך מזהה את תג ה-RFID שטבוע בחבילה ומעדכן את ה-DB שהתווספה חבילת גזרים למאגר. במקביל אתה מקבל דו"ח יומי שמעדכן אותך כי החלב יגמר ככל הנראה אחרי הקפה של מחר (לפי היסטוריית השימוש שלך בחלב), הברוקולי מלפני חודשיים עדיין במקרר ושני חברים שלך בדיוק עדכנו את גרסאת ה-FreezeOS של המקרר שלהם לגרסה האחרונה וממליצים לך גם.

הגרסה האחרונה כוללת תיקון של 4 חולשות, ש-2 מהן הוגדרו כ"קריטיות". אחת מהן אף נמצאה In-The-Wild באחת מגרסאות ה-ZigBotnet וממה שראית הרצת הקוד התאפשרה לאחר הכנסת



מחרוזת מספיק ארוכה בערך ה-RFID של המוצר שנכנס למקרר, והשניה איפשרה הרצת קוד בשל מימוש לקוי של מנגנון משיכת העדכונים מהרשת החברתית הדיפולטיבית שהגיע עם המקרר (וואלה, מזל שהחלפת...).

אעצור כאן, אך אני מאמין שהבנתם את הכיוון.

למרות כל הפוטנציאל שבדבר, לי קשה מאוד לשמוע סיפור כזה ולא לחשוב על כל הסכנות שבדרך, עולות לי מחשבות כמו: "רגע, מה הפרוטוקול שה-Quadcopter משתמש בו על מנת להזדהות אל מול התקן הנחיתה?", "איך אני יכול להיות בטוח שאף אחד לא החליף לי את המצרכים בדרך?", "האם מישהו פרץ למחסני המזון ושינה את שקיות המצרכים כך שמזדהה ה-RFID שלהם יגרום להרצת קוד על המקרר שלי?", "האם ה-NSA יודעים מה אני אוכל לארוחת בוקר?", או "למה לעזאזל האפליקציה של שמנהלת את הבית החכם צריכה כל כך הרבה הרשאות?", ובכלל - "איך אני יכול לסמוך על כל ה-Setup הזה כשאני יודע שכל הציפים שמרכיבים אותו פותחו בסין?!"...

הפירצה הקטנה ביותר במערכת מאפשרת לכל אחד להריץ קוד על המקרר שלי, משם לקפוץ ל-Streamer (כי ברור שהם מחוברים - אני מעוניין לגשת לעמוד המצרכים ישירות מהטלוויזיה) וממנה להגיע ישירות למחשב האישי שלי.

ברור לי שיהיה ניתן לממש את הכל בצורה מאובטחת, אבל כפי שההיסטוריה מלמדת אותנו - אם ניתן לממש משהו בצורה נכונה ומאובטחת או בצורה גרועה ורשלנית - התעשייה בחרה באופציה השניה. בין שמדובר בפרוטוקולי תקשורת מרכזיים ביותר בתעשייה שאנ'לא-יודע-מי-לעזאזל-אישר-מבחינת-אבטחת-מידע ([?CANbus](#) [?SMTP](#) [?FTP](#) [?DNS](#) [?WEP](#) [?ARP](#)) ובין שמדובר בפרוטוקולים שעוד איכשהו אפשר לדבר עליהם מבחינת אבטחה אבל החברות בתעשייה בחרו לממש אותם בצורה זוועתית, ברוב המקרים, התוצר שהלקוח קיבל - היה לוקה בכל הקשור באבטחה.

כנראה שאני אהיה בין האחרונים שלמקרר או לטוסטר שלהם תיהיה כתובת IP, אבל ברור לי שרוב לקוחות התעשייה יאמצו את הטכנולוגיות האלה (ואז גם יחכו שעות בתור למענה הקולי של הוט, כשהם ימצאו את עצמם נעולים מחוץ למקרר שלהם בגלל ששרתי ה-DNS של הספקית קרסו לאחר מתקפה של אנונימוס...).

עולם כזה יהיה מדהים מזווית ראייה של האקר, אך מחריד מזווית הראיה של המשתמש הביתי. וכמובן, לפני שניגש לסיבה בגינה הביטים האלה נמצאים על המחשב שלכם, ברצוננו להודות לכל מי שהשקיע ונתן מזמנו האישי ובזכותו הגיליון פורסם, תודה רבה ל **ניר עופר (hyprnir)**, תודה רבה ל**עידן אלון**, תודה רבה ל**עידו קנר** ותודה רבה ל**ים מסיקה!**

**קריאה מהנה!**  
ניר אדר ואפיק קסטיאל.



---

## תוכן עניינים

---

2	דבר העורכים
4	תוכן עניינים
5	אז מה קרה החודש?
14	בטוח שאתה זוכר נכון?
38	אם תמונה שווה אלף מילים, כמה מילים שווה חולשה באינסטגרם?
46	הכרת SCTP - חלק ראשון
59	דברי סיכום

## אז מה קרה החודש?

מאת ים מסיקה

### על אנדרואידים וכחד במה

בחודשים האחרונים התפרסמו שוב ושוב ידיעות על פרצת אבטחה באנדרואיד בשם [Stagefright](#). אותה פרצה היא למעשה קבוצה של חורי אבטחה שנמצאו בספרייה בשם `libstagefright`, שמטרתה ניגון מדיה במכשירי אנדרואיד. מאז התגלו הפרצות הראשונות בחודש אפריל השנה, נמצאות בספרייה שוב ושוב פרצות שדורשות תיקון.

[חורי האבטחה](#) שהתגלו עד כה, רובם מסוג [Integer Overflow/Underflow](#) ומאפשרים Remote Code Execution ו-Privilege Escalation. הם נמצאו על-ידי חוקר בשם יהושע דרייק מחברת Zimperium, והוצגו לאחר מכן בכנס BlackHat ובכנס DEF CON ה-23 (המצגת מעט ארוכה, אבל מומלצת מאוד). אותן פרצות יצרו רעש גדול מאוד בעולם האבטחה, שכן הספרייה קיימת מאז אנדרואיד 2.2 ועד הגרסה הנוכחית (5.1.1), מה שיוצר את הנתון הבלתי-נתפס של כמיליארד מכשירים פריצים.

הרצת הקוד מתרחשת כאשר `libstagefright` מנסה לעבד סרטון שנערך באופן זדוני. לרוע המזל, אפליקציות שונות באנדרואיד מנסות לעבד סרטונים עוד לפני שהמשתמש מנסה לפתוח אותם. במקרה של קבלת סרטון ב-MMS, לדוגמה, הסרטון יעובד על-ידי `libstagefright` אפילו עוד לפני שתוצג התראה למשתמש. מאחר והחל מגרסה 4.1 של Android מערכת ההפעלה מימשה מנגנון [ASLR](#)<sup>1</sup>, ככל הנראה יהיה קצת יותר קשה לנצל את החולשה בפועל, וכרגע לא ידוע על מימושים שהצליחו לעשות זאת.

אחד הדברים שהורגשו באופן משמעותי מאז שהתגלתה הפרצה, היה חוסר האפקטיביות של תהליך עדכון מערכת ההפעלה. העובדה שהפצת עדכוני הקושחה נמצאת באחריות ספקיות הסלולר פגמו ביכולת של גוגל לספק עדכון מהיר לפרצה, וגרמו לעיכובים בהפצת הטלאי.

כפתרון לבעיה הזו, הקימו Zimperium את "התאגדות Zimperium לטלפונים סלולאריים" (באנגלית זה נשמע כליכך הרבה יותר טוב: Zimperium Handset Alliance) שמאפשרת החלפת מידע וקבלת עדכונים בזמן אמת על בעיות אבטחה הקשורות באנדרואיד.

אז מה החדשות, אתם שואלים? ממש בתחילת החודש ידידינו דרייק [גילה](#) זוג חולשות נוספות:

- [CVE-2015-6602](#) הוקצה עבור חולשה בספרייה בשם `libutils`. הפגיעות ניתנת לניצול מגרסה 1.0 של אנדרואיד (!) רק עבור תוכנות חיצוניות שמשמשות בפונקציה הפגיעה.

<sup>1</sup> למעשה, מנגנון ASLR ממומש ב-Android עוד מגרסה 4.0. למרבה הצער, ה-ASLR שם ניתן לעקיפה יחסית בקלות, כך לפי דרייק.



- [CVE-2015-3876](#) הוקצה עבור חולשה באותה ספרייה, libstagefright שמאפשרת Remote Code Execution בגרסאות אנדרואיד מ-5.0 ומעלה.

החולשות האלו קיבלו את השם "Stagefright 2.0", וטיפה יותר קשה לנצל אותן עכשיו, כי מרבית האפליקציות הפסיקו לעבד את הסרטונים באופן אוטומטי. דרייק מציע בנתיים שיטות חלופיות, כמו MITM (ובעזרתה להפעיל את הסרטון הזדוני דרך הדפדפן) או הנדסה חברתית (שבעזרתה תגרמו למישהו ללחוץ על לינק לסרטון הזדוני).

אגב, ברכות למנהל החברה Zimperium – יצחק Zuk אברהם, ידיד צוות המגזין על מציאת החולשה ☺ כן ירבו.

### על סכרים, קריפטוגרפיה ו-NSA

לא פעם עולה השאלה, לפחות אצל הפרנואידים שבנינו – מי יכול להאזין לנו? באילו מקרים ותחת אילו מצבים?

אף אחד לא יכול להגיד שהדלפותיו של סנודן עזרו לנו, הפרנואידים, להשקיט את החששות שלנו. אחרי פרסום תוכנת הפענוח האמריקאית [BULLRUN](#) ואחותה הבריטית Edgell חלקנו עברנו להשתמש ביוני-דואר. איזו ספקולציה לא עלתה לגבי השיטה בה NSA מפענחת את התקשורת המוצפנת שלנו? כל דבר החל מ-Backdoors בתוכנות מוכרות (טוב, בזה אולי [צדקנו](#), למרות שזה [לא היה לא צפוי](#)) ועד גישה למאגרים של חברות ענק (טוב, יש מצב שגם בזה [צדקנו](#)). אני לא אגזים אם אגיד שעלו יותר תיאוריות אפשריות [מהספקולציות לגבי Lost](#).

אז על מה הרעש החודש, אתם שואלים? ברכות, ידידי, הכירו את [Logjam](#), פרצה חדשה שהתגלתה על ידי חוקרים בתחום הקריפטוגרפיה ופורסמה לציבור במאי 2015. היא הגיחה לכותרות החודש כשחבורת החוקרים החביבים פרסמו [מאמר](#) מעניין בשם "איך דיפיהלמן נכשל בפועל", וחשפה, אולי, את הדרך של ה-NSA לצוותת לכמות תעבורה לא סבירה שעוברת ברשת.

כדי להבין מה קורה שם, זה הזמן להכיר רעיון חדש שנקרא "[קריפטוגרפיה מורשית לייצוא](#)" – אוסף של חוקים שמטרתם להגביל את החוזק המקסימלי של ההצפנה שעוברים מסרים לפני שהם יוצאים מארץ מסוימת. בארצות-הברית, לדוגמה, היו חוקים ברורים ומאוד קפדניים מאז מלחמת-העולם השנייה, שדענו בתהליך ארוך ומייגע מאז 1992 ועד שנת 2000, כאשר גם לאחר הדעיכה נשארו חוקים לא רלוונטיים. כחלק מהבעיות שהשאירו לנו החוקים האלו, הרבה אלגוריתמי הצפנה מאפשרים להשתמש גם ב"צורה

---

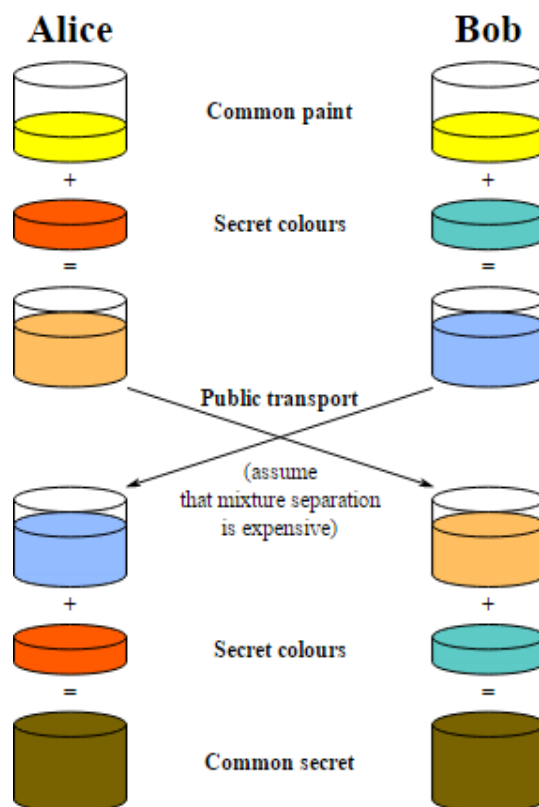
אז מה קרה החודש?

[www.DigitalWhisper.co.il](http://www.DigitalWhisper.co.il)

חלשה" שלהם, כזו שחזקה מספיק כדי למנוע ממשתמשים פרטיים לפענח את המסר וחלשה מספיק כדי שמעצמות יוכלו לפענח אותו.

עוד רעיון שנצטרך להבין נקרא [דיפיה-הלמן](#). דיפיה-הלמן הוא פתרון מודרני ל**בעיית הפצת המפתחות**, ולמעשה הצורה שבה אנחנו יכולים להעביר מפתח הצפנה סודי שיהיה ידוע רק לשנינו, בלי שאנשים אחרים יוכלו להבין אותו – גם אם הם מאזינים לנו. הפתרון הזה עומד בבסיס בסיסה של רוב התקשורת המאובטחת שאנחנו עושים כיום באינטרנט – HTTPS, SSH, SMTPS ועוד. אנחנו משתמשים בו כיום בעיקר כדי להעביר בעזרת הצפנה אסימטרית, באופן שבו אנשים שמאזינים לא יוכלו להבין, מפתח סימטרי שבעזרתו נצפין את המשך התקשורת שלנו עם הצד השני.<sup>2</sup>

[תמונה](#) שגנבתי מוויקיפדיה ([קרדיט!](#)) ומסבירה מדהים את הרעיון הכללי מאחורי איך זה עובד:



כדי להתחיל את החגיגה של ערבובי הצבעים האלו, יש לנו את אליס (הלקוחה) ואת בוב (השרת). אליס אומרת לבוב באילו פרטוקולי הפצת מפתחות היא יודעת לדבר. היא עושה את זה כדי שהם יוכלו להסכים ביניהם על העיגול צהוב שבתמונה – המספרים המשותפים לשניהם, שבעזרתם הם יתחילו להשתמש בפרטוקול דיפיה-הלמן. אז לצורך הדוגמה שלנו, אליס תשלח לבוב שהיא רוצה להשתמש ב-"DCE", שמשמעו: Diffie-Hellman Encryption, הצפנת דיפיה-הלמן. כשבוב ישמע את כל ה-Cipher suites שאליס

<sup>2</sup> המעבר ממפתח אסימטרי למפתח סימטרי נעשה כיוון שפענוח בעזרת מפתח אסימטרי הוא איטי משמעותית, והעברת מידע המוצפן בעזרת מפתח אסימטרי דורש העברה של 10% יותר נפח ב-Cleartext.

אז מה קרה החודש?

[www.DigitalWhisper.co.il](http://www.DigitalWhisper.co.il)



יודעת לדבר, הוא יבחר את החזק ביותר מבניהם שגם הוא יודע לדבר, ואז לפיו הוא יבחר מספר גדול במיוחד. ככל שהמספרים האלו יהיו גדולים יותר, כך ההצפנה תהיה חזקה יותר.

סתם כדי לשבר את האוזן: החוקרים אמרו ששימוש במספרים באורך 512 bit ושל 768 bit אינו מומלץ ויכול להישבר אפילו על ידי צוות אקדמי במעבדה (הדגמות בהמשך), שניתן לשבור מספרים באורך 1024 bit אם אתה ממשלה עם הרבה כסף, ושמספרים בגודל 2048 bit נחשבים לבטוחים כרגע.

אבל מה אם הייתי אומר לכם שיש פרוטוקול הפצת מפתחות שנקרא *DHE\_EXPORT*? אתם יכולים לנחש כבר בעצמכם – אותו דיפי-הלמן (DHE) שדיברנו עליו, רק "מורשה לייצוא" שכזה (לא זוכרים מה זה? גלגלו עמוד אחורה). גרסה שמבקשת מבוב לא להשתגע עם הפרמטרים שהוא בוחר; והוא מציית ובוחר פרמטרים באורך 512 bit.

עכשיו בואו נחשוב שיש תוקפת בעמדת Man in the Middle, אחת שיכולה להתחזות לאליס ולגרום לבוב להאמין בזה. לשם הנוחות, נקרא לה איב. אז אליס באה לשלוח לבוב את זה שהיא רוצה לדבר ב-DHE עם מפתח באורך 1024 bit, אבל איב מעבירה לבוב שהיא יודעת לדבר רק ב-DHE\_EXPORT. בוב מעביר לאליס פרמטרים ממש חלשים, ואז אליס חושבת שזה הכי טוב שהוא יודע. אז אליס זורמת, ואיב זוכה בהצפנה שיחסית קל מאוד לפענח. 8.4% ממיליון האתרים הפופולריים ביותר באינטרנט היו אהבלים כמו בוב והסכימו לדבר מעל DHE\_EXPORT.

הדבר היחיד שאליס יכולה לעשות במצב כזה, הוא להחליט שלדבר עם פרמטרים באורך כל-כך קצר לא בא בחשבון. בכל מקרה אחר, אין לה דרך להימנע מהמתקפה – הרי אין לה באמת דרך לדעת שאיב נמצאת על הקו ומבלפת את בוב!

אם הגעתם עד לכאן אתם די קשוחים, והגיע הזמן לספר לכם שזה עדיין לא באמת כזה פשוט. למעשה, אם בכל חיבור בוב ישלח לאליס פרמטרים שונים לגמרי באורך 512 bit, אז עדיין ייקח לאיב יחסית הרבה זמן לפענח את ההתקשרות ביניהם. 😊

המתקפה מתבססת בעקרון על זה שאיב הולכת לצד, משקיעה המון זמן בהליך מתמטי מורכב, מפרקת את הפרמטרים הספציפיים שבו שלח לאליס (במקרה שאנחנו מדברים עליו – פרמטרים באורך 512 bit, אבל נכון גם עבור שאר המקרים), וכשהיא תצליח יהיה לה משמעותית יותר קל להבין את השיחה ביניהם.

איזה מזל, אם כך, שבוב לא בוחר כל פעם פרמטרים אחרים! אתם מבינים, לפני שגילו את כל הסיפור הזה עם איב, לא מצאו סיבה שלא לשלוח את אותם פרמטרים כל הזמן. זה לא היה נראה כאילו זה יכול לפגוע יותר מדי. ההנחה הזו גרמה להמון שרתים להשתמש כל הזמן באותם פרמטרים (ברוב התוכנות



הם פשוט היו Hardcoded), מה שהוביל לכך ש-7% מכל תקשורת<sup>3</sup> שהוחלט שתוצפן ב-bit 512 תחת דיפיה-הלמן, תמיד התבצעה עם אותם מספרים<sup>4</sup>.

החוקרים מדברים על כך ש-18% מהאתרים הפופולאריים בעולם משתמשים באותם פרמטרים עבור הצפנת ה-bit 1024 שלהם ל-HTTPS, דבר שחמור בפני עצמו (גם אם לא משתמשים ב-Man in the middle). זאת אומרת שאם ה-NSA מחליטה לשבור את אותם פרמטרים, מה שהחוקרים מניחים שיש באפשרות גוף ממשלתי לעשות, הם יכולים להאזין ל-18% מתעבורת האינטרנט מעל HTTPS. שבירה של הפרמטרים הבאים בתור יאפשרו להם להאזין ל-66% מתעבורת ה-VPNים ול-26% מהתקשורת מול שרתי SSH.

אם כן, אולי לא תופתעו מדי לגלות שלפי מסמכים שפרסם סנודן, ה-NSA [השקיעו קצת דמי כיס](#) (11 מיליארד דולר לשנה!) ב"יכולות קריפטואנליטיות פורצות-דרך". זה בוודאי, אמנם, לא אומר ש-NSA הפכו להיות איב, וזה לא אומר שזה מה שהם עשו. למרות זאת, החוקרים טוענים ש"השבירה של פרוטוקול דיפיה-הלמן כפי שתיארנו אותה מתאימה לפרטים הטכניים הידועים לנו על הפענוח שלהם בקנה מידה גדול יותר מאשר כל הסבר מתחרה, כולל שליחת חיבורים מוצפנים מסוימים למחשבי-על, שמחזירים מפתח מסוים".

אפרופו מעקבים, NSA, סוכנויות בין וכולו, הנה אנקדוטה משעשעת שחובה לסגור איתה את האייטם



הזה: ביום שבו הגשתי את הכתבה במגזין האחרון, אדוארד סנודן פתח [חשבון טוויטר](#) (מזל טוב אחי). יש לו כבר 1.56 מיליון עוקבים לשעת כתיבת הכתבה(!), אבל הוא עוקב רק אחרי משתמש אחד. סנודן שלנו, מסתבר, טרול לא קטן. אחרת, איך הייתם מסבירים את התמונה?

## מקורות:

- המחקר: "Imperfect Forward Secrecy: How Diffie-Hellman Fails in Practice":

<https://weakdh.org/imperfect-forward-secrecy-ccs15.pdf>

<sup>3</sup> נמדד לפי מיליון האתרים המובילים ב-Alexa. החוקרים טוענים שמדובר ביותר ממיליון שרתים שמגישים דפים ב-HTTPS, שמגיעים ל-3% מכלל שרתי ה-HTTPS בעולם.

<sup>4</sup> המספרים האלו נמצאים בשימוש ב-Apache מגרסה 2.1.5 ועד גרסה 2.4.7. החוקרים פרסמו את השבירה עבור הפרמטרים האלו, אותה הם ביצעו במעבדה שלהם. הם גם שברו את הפרמטרים ש-OpenSSL משתמשת בהם, הן עבור DHE והן עבור DHE\_EXPORT. עבור כל מספר לקח להם שבוע (סך הכל – שבועיים).

אז מה קרה החודש?

[www.DigitalWhisper.co.il](http://www.DigitalWhisper.co.il)



- אתר מוצלח שהחוקרים הקימו, ומדבר על הפרצה ברמה מאוד פרקטית ומציע גם דרכי התגוננות:  
<https://weakdh.org/>
- כתבה על הפרצה:  
<https://freedom-to-tinker.com/blog/haldermanheninger/how-is-nsa-breaking-so-much-crypto/>
- מאמר טכני של Cloudflare:  
<https://blog.cloudflare.com/logjam-the-latest-tls-vulnerability-explained/>

## ליב רע SSL?

באג ה-Heartbleed שפורסם באפריל דאשתקד גרם לבהלה ולרעש רב, וכבר באותו חודש נפוצו ההודעות על מימוש חדש לפרוטוקולי ה-TLS וה-SSL בשם [LibreSSL](#). המימוש החדש מגיע ממש מתוך צוות OpenSSL, עקב הבנה שהתגבשה שם לגבי העובדה שתחזוקה ובקרה של הקוד הפכו להיות מטלות קשות מאוד.

מפה לשם נוצר Fork מספריית הקריפטוגרפיה של OpenSSL, כאשר המטרה היא לעשות refactoring לקוד של OpenSSL כך שיהיה בטוח יותר. הפרויקט הוכרז ב-22 באפריל, ובתוך כשבוע נמחקו 90,000 שורות קוד שירשה מ-LibreSSL מ-OpenSSL.

בזמן שמאז 2014 נמצאו ב-OpenSSL 43 חולשות אשר מתוכן 5 מסווגות כקריטיות, ב-LibreSSL נמצאו רק 22, מתוכן אף לא חולשה קריטית אחת. רוב החולשות שפורסמו היו חולשות השייכות ל-OpenSSL שנמצאו גם בקוד המקור של LibreSSL, אך בחודש האחרון קרה משהו חריג, ופורסמו שתי חולשות שהיו קיימות ב-LibreSSL – ולא ב-OpenSSL.

[החולשות](#) בתוכנה, Buffer Overflow ו-Memory Leak, נתגלו על ידי Qualys Security ופורסמו ב-15 באוקטובר. הן שותפו עם החברה [שהודיעה](#) על פרסום הטלאי תוך יום.

## הבוטנט של השכן

חברת האבטחה Imperva מספרת על אחד הלקוחות שלה שהותקף ב-20,000 בקשות HTTP לשנייה. כשהם חיפשו את מקורן של כתובות ה-IP שגרמו לכל הבלגאן, הם גילו שאחד הבוטנטים נמצא ממש בסביבה. אחרי מחקר שערך זמן קצר הצוות הבין שהמכשיר הזה הוא למעשה מצלמה במעגל סגור. מסתבר שכל הבוטנטים היו מצלמות שניתן להתחבר אליהן עם שם המשתמש והסיסמה שמוגדרים אצלן

---

אז מה קרה החודש?

[www.DigitalWhisper.co.il](http://www.DigitalWhisper.co.il)

כברירת מחדל. מן הסתם שהם התחברו מיד למצלמה הקרובה, וראו שם חנות שנמצאת במרחק 5 דקות נסיעה מהם. לא ייסעו? בטח שייסעו!

Imperva מספרים שהשוק הזה גדול במיוחד: ישנן 245 מיליון מצלמות מעקב בעולם ש"הותקנו באופן מקצועי". מעניין כמה בלתי מדווחות יש, ומעניין עד כמה זה משפיע על רמת האבטחה שלהן.

בכל מקרה, אחרי שהעבירו לבעל החנות שיעור באבטחת מידע, הם גילו שהתקיפה שהתבצעה על האתר של הלקוח שלהם הייתה מכ-900 מצלמות שפנו לאתר בבקשות GET סטנדרטיות. על כל המכונות היה [BusyBox](#), והן ככל הנראה הותקפו בעזרת גרסה מסוימת של ELF\_BASHLITE. לרוב המצלמות שזוהו התחברו מספר יחסית גדול של משתמשים, כך שנראה שהן נפרצו על ידי יותר מאדם אחד.

כששמים את הידיעה הזו ליד [ידיעה](#) על קומקום שעוזר לאנשים לפרוץ לנו ל-WiFi בבית, כל הקונספט של "האינטרנט של הדברים" גורם לי להרהר פעם נוספת לגבי היתרונות שיש בלהדליק את הקומקום שלי מהמיטה.

## טוק טוק – סיפורה של בדיחה עצובה

בתאריך ה-21 באוקטובר 2015 הודיעה הענקית הבריטית TalkTalk על פריצה למאגריה הממוחשבים. החברה מספקת שירותי טלוויזיה, טלוקומוניקציה, אינטרנט ו-Mobile, ומוצבת במקום השני מבחינת נתח שוק של השירותים הללו בכל הממלכה המאוחדת.

ביום הפרצה החברה נשמעה מאוד מבולבלת והיה ניכר מאוד שאנשי המקצוע שלה לא מתקשרים באופן מושלם עם אנשי הדוברות, בלשון המעטה. דוברות החברה הודיעה שנתונים של לקוחות ככל הנראה נגנבו על ידי מתקפת DDoS. זה בסדר, לא השתגעו – זה פשוט בלתי אפשרי וההודעה הזו מוזרה (זאת אומרת, ייתכן שזה קרה, לא ייתכן טכנית שכך נגנבו נתוני המשתמשים), אבל זו בדיוק ההצהרה שיצאה מ-TalkTalk. מעבר לכך, הם הצהירו ש"המערכות היו הכי מוגנות שהן יכלו להיות", [למרות ש](#)נתוני המשתמשים לא היו מוצפנים ואנחנו לא היינו חייבים להצפין אותם לפי חוק". יותר מכך, במאגר הוחזקו גם משתמשים שעזבו את החברה, ואפילו משתמשים שביקשו להימחק ממאגר הנתונים שלהם. מוזר.

נראה שהבלבול והעיכוב בתגובה מסודרת ומדויקת עלו לחברה ביוקר. הפריצה גרמה לה נזק עצום, שכן [מניותיה צנחו ב-10.7%](#) יום אחרי שהעניין התפרסם. עדכונים בקשר לפריצה הגיעו לאחר-מכן, ולפי ההודעה חולשת SQL Injection שהייתה באתר החברה היא האחראית לכל הבלגאן. במהלך האירוע ניסו התוקפים לסחוט את TalkTalk, ואמרו שאם לא יקבלו 80,000 פאונד ב-Bitcoin הם יפרסמו את נתוני הלקוחות. בהמשך הוקם [דף הסברה](#) מיוחד עבור האירוע, ויצא [סרטון](#) שבו המנכ"לית מנסה לפייס את לקוחות החברה בצורה רהוטה, תוך מה שנראה כמו טיפול מקצועי באירוע (אם כי בעיקוב רציני).

---

אז מה קרה החודש?

[www.DigitalWhisper.co.il](http://www.DigitalWhisper.co.il)

מחקר שערכה החברה גילה שבמסגרת הפרצה התגלו 1.2 מיליון כתובות דואר-אלקטרוני, שמות וטלפונים של לקוחות, 21,000 מספרי חשבון בנק, 28,000 מספרים של כרטיסי אשראי ללא 6 ספרות אמצעיות ו-15,000 תאריכי לידה. החברה דואגת להדריך את מנוייה כיצד להימנע מניסיונות הנדסה חברתית שעלולים לבצע עליהם עם הפרטים שנגנבו, ואף מספקת להם שירות למעקב אחרי הפעולות בכרטיס האשראי למשך 12 החודשים הקרובים בחינם.

הסיפור מסתבך כשמסתבר שהתוקפים אכן מימשו את הצהרתם ופרסמו את הנתונים למכירה באתר בשם AlphaBay, אתר ב-Deep Web שמטרתו מכירת סמים בלתי-חוקיים ודברים גנובים. [מאשכול שיחה](#) ב-Reddit ניתן ללמוד שהמוכר, רמה 6 ב-AlphaBay, מכר כבר סחורה לפחות ב-\$75,000 ושיש לו לפחות 90% פידבק חיובי.

חקירת האירוע נעשית על ידי משטרת מטרופוליטן, ועד כה נעצר ילד בן 15. לא נראה שהאירוע קרוב לסיומו, אבל דבר אחד בטוח – מדובר בפרצה בסדר גודל ענק שיש לחברות מסחריות הרבה מה ללמוד ממנה.

## Not-so-True-Crypt

אני לא אשכח את אותו יום מפתיע והזוי במאי 2014, שחבר שלח לי לינק לאתר של TrueCrypt. באתר הופיעה הודעה עמומה בנוסח מוזר שיגרום לכל אחד להרים גבה ולשאול את עצמו מה לעזאזל קורה שם:

**אזהרה:** השימוש ב-TrueCrypt אינו בטוח וייתכן שהתוכנה כוללת בעיות אבטחה שלא תוקנו. הדף הזה קיים רק כדי לעזור לך לייצא את הנתונים שלך שהוצפנו על-ידי TrueCrypt.

כשבדקנו, לא נמצאו שינויים ב-DNS או ב-WHOIS של האתר, והגרסה שהועלתה ב-27 במאי נחתמה בעזרת אותו מפתח ששימש את צוות TrueCrypt לחתום על הגרסה שהועלתה בינואר 2014. נראה שההודעה הייתה אותנטית.

הסיפור המסתורי הגיע במהרה לחדשות בדמות אינסוף כתבות ב-RSS Feed שלנו. זה היה השלב שבו הפסקנו להשתמש ב-TrueCrypt ועברנו ל-[VeraCrypt](#), אבל אי אפשר שלא להתעניין במה בדיוק קרה שם: יוצרים אנונימיים של תוכנה שמפותחת בערך 10 שנים סוגרים לילה אחד את האתר שלהם ומזהירים אנשים מלהשתמש בה. קריפי.

בעבר כבר [נמצא](#) פרצות שסווגו ברמת סיכון בינונית או נמוכה ב-TrueCrypt. זה לא מנע מחוקרי האבטחה ב-Project Zero של Google להסתער החודש על TrueCrypt, ולמצוא שתי חולשות, אחת מהן

היא [חולשה ב-Driver](#) שמאפשרת Priviledge Escalation, והשנייה חולשה משמעותית פחות מעניינת שמאפשרת לתוקף לנתק כוננים (dismount) ולקבל עליהם מידע.

חשוב לציין שאף אחת מהחולשות לא משפיעה על הבטיחות של המידע בכוננים המוצפנים בשום צורה. ניתן לפנות [לכאן](#) לקריאה נוספת של פרטים טכניים על החולשה, כולל PoC.

## שה-1 תמים

4 שנים אחרי שפורסמה SHA-2, היורשת של SHA-1, [התפרסמה](#) החולשה המשמעותית הראשונה ב-SHA-1. זה היה בשנת 2005, וכבר אז לכולם היה ברור שמשהו חייב לקרות וחייבים להתחיל מעבר לקראת SHA-2. לכולם גם היה ברור שהדבר הזה לא יקרה בן לילה.

היום, 10 שנים אחרי מציאת החולשה המשמעותית באלגוריתם ואחרי שכולם הבטיחו לכולם שכולם יעברו מיד, ישנם עדיין מקומות שמשמשים ב-SHA-1 ([אתרים](#) של צבא ארצות-הברית, נניח). מכיוון שאי אפשר לתת לבעיה להישאר כמו שהיא, כל הדפדפנים הכריזו שהם יפסיקו לתמוך ב-SSL Certificates שנחתמו בעזרת SHA-1 עד תחילת 2017.

בינתיים, ברוס שנייר השתמש בחוק מור כדי [לחזות](#) כמה יעלה לשבור SHA-1 בשנים הקרובות, וחזה שעד 2018 יצירת collision ל-SHA-1 תעלה \$173,000 – לא משהו יקר מדי, בוודאי לא עבור מעצמה או ארגון פשע שמחזיק בכסף רב.

לא עבר הרבה זמן עד שכמה חבריה חביבים מהולנד, צרפת וסינגפור עשו [מחקר נוסף](#) על SHA-1 שמצליח לשפר התקפות קיימות, והצליחו לקרב קצת את ההערכה של ברוס. כשאני אומר "קצת", אני מתכוון שלפי איך שזה נראה מההערכות שלהם – בעוד חודשים ספורים אפשר יהיה לקדם את המחקר שלהם לכיוון יצירת collisions באופן פרקטי, ולהשקיע "רק" \$75,000-\$120,000 כדי להשיג collision.

מפה לשם, לאור האירועים האחרונים, מוזילה [שוקלים](#) להקדים את התאריך של הפסקת התמיכה ב-SHA-1 1 לתחילת יולי 2016. החוקרים מעודדים את שאר הדפדפנים לא להיכנע לבקשות שבאות אליהם מכיוון התעשייה, ולהקדים ככל האפשר את הפסקת התמיכה ב-SHA-1, שידוע כאלגוריתם שבור מזה זמן רב.



---

## בטוח שאתה זוכר נכון?

מאת ניר עופר / hyprnir

---

### הקדמה

המאמר הבא יעסוק בכתיבת כלי בשפת C שפועל כמו Cheat Engine המוכר. מהותו העיקרית של הכלי היא מציאת ערך בזיכרון של תהליך אחר ושינויו לערך רצוי. בעזרת פעולה פשוטה כזו אפשר לעשות הרבה דברים, אך Cheat Engine התפרסם בעיקר בהיותו שימושי במניפולציה על משחקים. המשתמש יכול למשל לחפש את המספר שמייצג את כמות התחמושת שלו בזיכרון של תהליך המשחק ולשנות ערך זה.

שימוש ביכולת זו כולל הרבה ניסוי וטעיה מפני שלא ניתן לדעת איך בדיוק התהליך משתמש בערך הרלוונטי, כך שיכול להיות שאם למשתמש יש 80 פצצות במשחק והוא יחפש את הערך 80 הוא ימצא הופעות רבות שלו בזיכרון התהליך. בעיה נוספת שלא ניתן לפתור באמצעות Cheat Engine היא שמירה של ערכים בצורה שונה ולא גלויה למשתמש. למשל, שמירה של 80 פצצות כ-320. ישנה שיטה עיקרית לצמצום הממצאים, אך גם היא לא מושלמת ואציג אותה בהמשך. אני ממליץ לקרוא את חלקי המאמר יחד עם [קוד המקור הרלוונטי](#).

### איך הכל התחיל?

הגעתי לנושא בזכות crackme שכתב דימה פשול. ([כתבה שפירסם בגיליון ה-65](#)) ה-crackme מגיע עם הקדמה בצורת קובץ Header כחלק מהקבצים בקישור.

ה-crackme מכיל רשימת סיסמאות עם סיסמאות למערכות שונות. כל סיסמא מיוצגת על ידי struct מסוג linked\_list\_member ומאפיינים אותה אורך ו-ID. הסיסמא שאנחנו מחפשים היא הסיסמא למערכת הטילים, עם ה-ID 0x30.

---

בטוח שאתה זוכר נכון?

[www.DigitalWhisper.co.il](http://www.DigitalWhisper.co.il)

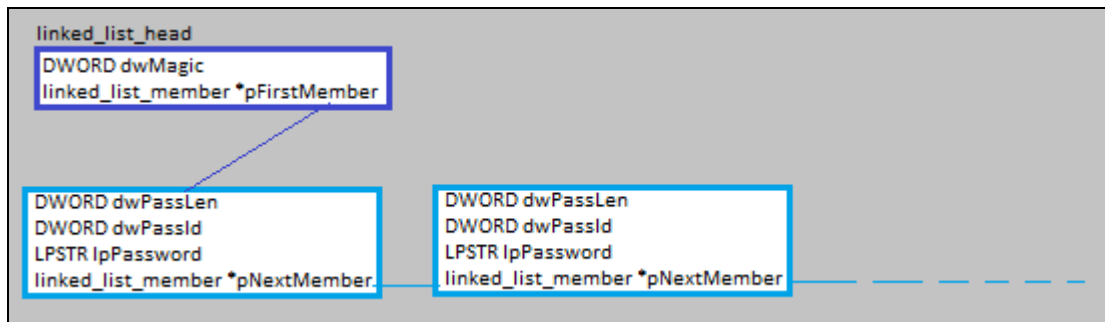
הרשימה המקושרת בנויה באופן הבא:

```
#define LIST_HEAD_MAGIC 0x12345678
#define ROCKET_SYSTEM_PASSWORD 0x00000030
#define HR_SYSTEM_PASSWORD 0x00000020
#define ERP_SYSTEM_PASSWORD 0x00000010
#define FACEBOOK_PASSWORD 0x00000005

typedef struct linked_list_member
{
    DWORD dwPassLen;
    DWORD dwPassId;
    LPSTR lpPassword;
    struct linked_list_member *pNextMember;
}*plinked_list_member;

typedef struct linked_list_head
{
    DWORD dwMagic = LIST_HEAD_MAGIC;
    struct linked_list_member *pFirstMember;
}*plinked_list_head;
```

הרשימה נראית כך:



כאשר מריצים את crackmen-, כל מה שרואים הוא החלון הבא:

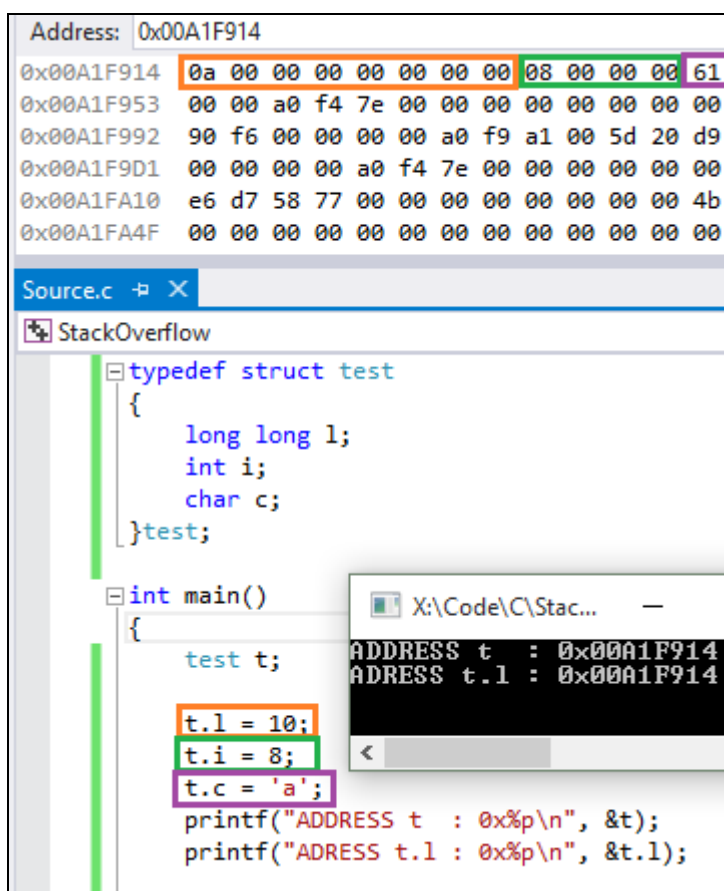


מהמידע שאנחנו מקבלים לגבי ה-crackme מסתמן שהפתרון לא מסובך, אך בעיני הוא מגניב ואלגנטי. מה שעלינו לעשות הוא לכתוב כלי שמגדיר את ה-structים linked\_list\_head ו-linked\_list\_member. הכלי מחפש בזיכרון התהליך של ה-crackme את הערך LIST\_HEAD\_MAGIC המסמן את linked\_list\_head. משם אפשר להשתמש ב-structים, לרוץ על הרשימה המקושרת ולמצוא את הסיסמא.

כדי לעשות את זה, מומלץ להבין ברמה מסוימת איך structים נראים בזיכרון ולשחק איתם קצת. העיקרון שמשנה לנו הוא הבנה של מהי כתובת הבסיס של struct. (ריפוד בין משתני ה-struct למשל לא משנה לנו במקרה הזה).

structים הם תאגיד של משתנים שונים שמאוגדים תחת משתנה/מצביע אחד. הם פרוסים ברצף בזיכרון כך שכתובת הבסיס של ה-struct היא כתובת המשתנה הראשון. מיקומם בזיכרון נקבע לפי סדר ההצהרה עליהם. לכן, אם נמצא את הכתובת של dwMagic, נוכל להצביע לכתובת זו עם מצביע של struct מסוג linked\_list\_head ולעבוד איתו בצורה הגיונית.

המחשה של פריסת struct בזיכרון:



The screenshot displays a debugger window with a memory dump and source code. The memory dump at address 0x00A1F914 shows the following hex values: 0a 00 00 00 00 00 00 00 08 00 00 00 61. The source code shows a struct test with fields long long l, int i, and char c. The main function initializes t.l = 10, t.i = 8, and t.c = 'a', and prints the addresses of t and t.l.





בחרתי לפתור את ה-crackme בעזרת dll injection כי רציתי שהקוד שלי יוכל פשוט לפנות למשתני ה-struct כחלק מהתהליך בלי קריאות זיכרון נוספות עם ReadProcessMemory שהיו דרושות בריצה מחוץ לתהליך. הקריאות הנוספות היו דרושות מכיוון שב-Windows כל תהליך מתנהל במרחב כתובות פרטי ומערכת ההפעלה דואגת להמרה בין כתובת פרטית לכתובת פיזית. ארכיטקטורה זו לא מאפשרת לתהליך פשוט לפנות לזיכרון של תהליך אחר ללא עזרה ממערכת ההפעלה. העזרה הזו באה לידי ביטוי בפונקציות API שונות. בהמשך אציג גם פיתרון ללא dll injection. ההבדלים בין שתי הדרכים זניחים.

## פיתרון 1: עם DLL injection

דבר ראשון, יש צורך ביכולת להזריק DLL. אפשר להוריד כלי מוכן, או לכתוב באופן עצמאי. (ניתן לקרוא את המאמר [Code Injection בגיליון 13](#) שנכתב על-ידי אוראל ארד).

אני בחרתי לכתוב injector, אך לא אסביר אותו. הוא מצורף למאמר. ועכשיו לכתובת ה-DLL. נתחיל מהגדרת הקבועים הרלוונטיים:

```
#define LIST_HEAD_MAGIC 0x12345678  
#define ROCKET_SYSTEM_PASSWORD 0x00000030
```

עכשיו נגדיר את ה-structים. אני שיניתי קצת את השמות כדי שיתאימו לי. חשוב מאוד לשמור על סדר המשתנים כמו ב-struct המקורי. אחרת יוכל לקרות מצב בו נתייחס למשתנה אחד כמשתנה אחר, או שריפוד בין המשתנים יוביל אותנו לקריאה של ערכים לא נכונים:

```
typedef struct LINKED_LIST_MEMBER  
{  
    DWORD dwPassLen;  
    DWORD dwPassId;  
    char *lpPassword;  
    struct LINKED_LIST_MEMBER *Next;  
}LINKED_LIST_MEMBER;  
  
typedef struct LINKED_LIST_HEAD  
{  
    DWORD dwMagic;  
    struct LINKED_LIST_MEMBER *pFirstMember;  
}LINKED_LIST_HEAD;
```

תחת DLL\_PROCESS\_ATTACH בפונקציית DllMain נקרא לפונקציה InjectionMain. בתחילת הפונקציה נגדיר את המשתנים הבאים:

```
MEMORY_BASIC_INFORMATION mbi;  
SYSTEM_INFO si;  
LPVOID minAddress, maxAddress;
```

מרחב הכתובות הפרטי של תהליך הוא רצף לינארי של כתובות. כדי לסרוק את הזיכרון אנחנו צריכים לדעת מהי הכתובת המינימאלית ומהי הכתובת המקסימאלית. maxAddress תישאר קבועה, וב-minAddress

בטוח שאתה זוכר נכון?

[www.DigitalWhisper.co.il](http://www.DigitalWhisper.co.il)

נשתמש כדי לרוץ על הזיכרון. את הכתובות הדרושות אפשר למצוא באמצעות הפונקציה `GetSystemInfo` שכותבת את הפלט שלה ל-`struct` מסוג `SYSTEM_INFO`. לכן אנחנו מגדירים את `si`. בין משתני ה-`struct` נמצאים המשתנים הבאים:

```
LPVOID lpMinimumApplicationAddress;
```

```
LPVOID lpMaximumApplicationAddress;
```

משתנים אלו הם הכתובת המינימאלית והכתובת המקסימאלית שמוקצות למרחב הזיכרון הפרטי של תהליך במערכת. אלה הם הגבולות שבהם נשתמש. `MEMORY_BASIC_INFORMATION` הוא `struct` שמוחזר מהפונקציה `VirtualQuery` ומכיל מאפיינים על בלוק זיכרון. נשתמש בו כדי לאפיין את חלקי הזיכרון השונים שאנחנו סורקים בתהליך. לאחר שהגדרנו את המשתנים נקרא לפונקציה הדרושה ונתחיל לרוץ על הזיכרון:

```
GetSystemInfo(&si);
minAddress = si.lpMinimumApplicationAddress;
maxAddress = si.lpMaximumApplicationAddress;
```

כדי לרוץ על הזיכרון נשתמש בלולאת `while` ובכל ריצה שלה נגדיל את `minAddress`. הלולאה תראה כך:

```
while (minAddress < maxAddress)
{
    VirtualQuery(minAddress, &mbi, sizeof(mbi));
    if (mbi.State == MEM_COMMIT) SearchValue(&mbi);
    minAddress = (LPBYTE)mbi.BaseAddress + mbi.RegionSize;
}
```

כדי להבין למה הלולאה עובדת, צריך להבין איך עובדת הפונקציה `VirtualQuery`. נניח שתחילת הזיכרון נראית כך:



רצף של זיכרון עם אותם מאפיינים מיוצג על ידי אותו צבע. האותיות `a`, `b`, ו-`c` מייצגות כתובות הבסיס של רצפי זיכרון עם אותם מאפיינים. בפעם הראשונה שאנחנו קוראים ל-`VirtualQuery`, הערך של `a` הוא `minAddress`. הפונקציה מתחילה לרוץ על זיכרון התהליך החל מהכתובת שהיא מקבלת וממשיכה לרוץ על הזיכרון עד שהיא נתקלת בכתובת זיכרון עם מאפיינים שונים מהזיכרון שעליו רצה, כלומר, היא הגיעה לסוף של רצף זיכרון עם אותם מאפיינים. במקרה הזה הכתובת היא `b`. הפונקציה מחזירה `struct` מסוג `MEMORY_BASIC_INFORMATION` עם מאפייני רצף הזיכרון. ב-`struct` זה קיימים המשתנים:

```
PVOID BaseAddress;
```

```
SIZE_T RegionSize;
```

בטוח שאתה זוכר נכון?

[www.DigitalWhisper.co.il](http://www.DigitalWhisper.co.il)



במקרה הזה BaseAddress הוא a, RegionSize הוא b-a. מידע זה מאפשר לנו לרוץ על כל כתובות התהליך, לפי רצפי זיכרון בעלי מאפיינים זהים. לאחר שאנחנו קוראים לפונקציה אנחנו בודקים:

```
if (mbi.State == MEM_COMMIT && mbi.Protect == PAGE_READWRITE)
```

כלומר, האם לזיכרון הוירטואלי הזה הוקצה זיכרון פיזי. תהליך עובד במרחב כתובות וירטואלי משלו. לזיכרון שבו הוא משתמש בפועל מוקצה זיכרון פיזי במערכת. יתכן שהוא לא משתמש בכל הזיכרון הוירטואלי, ולכן מעניין אותנו רק הזיכרון שלו הוקצה זיכרון פיזי. בנוסף, נבדוק שההרשאות על הזיכרון מאפשרות קריאה וכתובה. הכתיבה לא קריטית אך אלה ההרשאות על הזיכרון במקרה הזה (אפשר לשחק עם התנאי ולהשמיט את החלק השני. במקרה שלנו אלה המאפיינים של הזיכרון שאנחנו מחפשים). אם הזיכרון אכן עומד בתנאים. נקרא לפונקציה SearchValue.

לאחר מכן נגדיל את minAddress כדי להמשיך את ריצת הלולאה. כדי להגיע לבלוק הבא בזיכרון עלינו לחבר את כתובת הבסיס של הבלוק הנוכחי עם גודל הבלוק. כלומר, נגיד וכתובת הבסיס הנוכחית היא a, וגודל הבלוק הוא b-a, כדי להגיע לכתובת הבסיס הבאה, b, עלינו לבצע את החיבור  $a + (b-a)$ . השימוש ב-LPBYTE נעשה כדי שהחישוב יהיה מדויק. הגדלה של מצביע תלויה בגודל הערך שעליו הוא מצביע. הוספה של 1 למצביע מסוג char\* תוסיף לכתובת שלו 1, אך הוספה של 1 למצביע מסוג int\* תוסיף לכתובת שלו 4.

הפונקציה SearchValue מקבלת מצביע מסוג MEMORY\_BASIC\_INFORMATION המצביע על בלוק זיכרון ובו היא מחפשת ערך לבחירתנו. במקרה הזה הערך הוא LIST\_HEAD\_MAGIC. מכיוון שהDLL רץ כחלק מהתהליך, הפונקציה פשוט תרוץ על הכתובות ותבדוק את ערכיהן. בפונקציה נגדיר שני משתנים. הכתובת המינימאלית של בלוק הזיכרון והכתובת המקסימאלית. נחפש כל פעם את LIST\_HEAD\_MAGIC בטווח זה:

```
LPVOID minAddress, maxAddress;  
  
minAddress = mbi->BaseAddress;  
maxAddress = (LPBYTE)minAddress + mbi->RegionSize - sizeof(DWORD);
```

minAddress שווה לכתובת הבסיס של בלוק הזיכרון. maxAddress שווה לסכום כתובת הבסיס וגודל הבלוק. הסיבה לחיסור sizeof(DWORD) מ-maxAddress היא שיש לחסר בית אחד כי אם נשאיר אותו התוצאה היא כתובת הבסיס של הבלוק הבא. מחסירים שלושה בתים נוספים היא כי אנחנו מחפשים ערך DWORD. גודל הערך הזה הוא 4 בתים, לכן אנחנו רוצים לרוץ על הבתים בבלוק הזיכרון עד לרביעי מהסוף, כי בדיקה של שלוש הכתובות אחריו כ-DWORD תוביל לחריגה מגבולות הבלוק.

בטוח שאתה זוכר נכון?

[www.DigitalWhisper.co.il](http://www.DigitalWhisper.co.il)

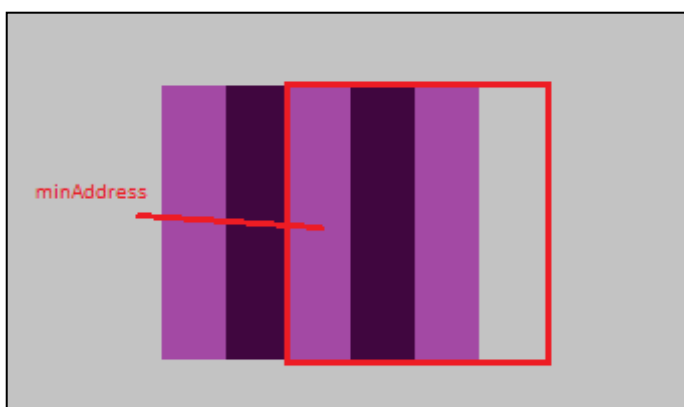
נניח ובלוק הזיכרון הוא בגודל 5 בתים ונראה כך (כל מלבן סגול הוא בית):



המצביע שלנו יעבור בית בית ויבדוק מהמיקום שלו ערך של 4 בתים:



התקדמות של המצביע אל בתים מעבר לבית ה-4 מהסוף לא רצויה:



עכשיו כשעניין הכתובות סגור, אפשר להגיע אל הלולאה. הלולאה תראה כך:

```
while (minAddress <= maxAddress)
{
    if (*(DWORD*)minAddress == LIST_HEAD_MAGIC) Roll(minAddress);
    minAddress = (LPBYTE)minAddress + 1;
}
```

בטוח שאתה זוכר נכון?

[www.DigitalWhisper.co.il](http://www.DigitalWhisper.co.il)



בכל פעם אנחנו בודקים את ערך ה-DWORD שנמצא בכתובת minAddress. כדי לקבל ערך DWORD יש להתייחס אל minAddress כמצביע ל-DWORD באמצעות (DWORD\*). הכוכבית הנוספת מביאה את הערך שנמצא בכתובת, אחרת היינו משווים את הכתובת ל-LIST\_HEAD\_MAGIC וזה לא רצוי. אם אכן מצאנו את הערך שאנחנו מחפשים אנחנו קוראים לפונקציה Roll עם הכתובת הרלוונטית. לאחר מכן מגדילים את minAddress ב-1 כדי להמשיך את פעולת הלולאה.

הפונקציה Roll מקבלת את הכתובת של הערך השווה ל-LIST\_HEAD\_MAGIC בזיכרון. הפונקציה תתייחס אליה כאל הכתובת של dwMagic ב-struct LINKED\_LIST\_HEAD. נגדיר שני מצביעים:

```
LINKED_LIST_HEAD *ListHead;  
LINKED_LIST_MEMBER *Member;
```

ListHead ישמש אותנו להתחלת התהליך והצבעה על רשימת הסיסמאות. Member יצביע בכל פעם על struct מסוג LINKED\_LIST\_MEMBER ובעזרתו נרוץ על הרשימה. כאמור, משתני ה-struct פרוסים ברצף בזיכרון לפי סדר ההצהרה עליהם, כשכתובת הבסיס של ה-struct היא הכתובת של המשתנה הראשון. המשתנה הראשון ב-struct הוא dwMagic והפונקציה מקבלת את הכתובת שלו כ-LPVOID Address. כלומר, הכתובת שעליה מצביע Address היא הכתובת שעליה נצביע עם ListHead בצורה הבאה:

```
ListHead = (LINKED_LIST_HEAD*)Address;
```

עכשיו כשיש לנו את המצביע לאיבר הראשון ברשימה, נצביע אליו עם Member:

```
Member = ListHead->pFirstMember;
```

נרוץ על הרשימה עם הלולאה הבאה:

```
while (Member->dwPassId != ROCKET_SYSTEM_PASSWORD) Member = Member->Next;
```

כמו שאמרתי, אנחנו מחפשים את הסיסמא למערכת הטילים, לכן, נרוץ עם Member ברשימה עד שנגיע ל-LINKED\_LIST\_MEMBER עם dwPassId מתאים. כשנצא מהלולאה Member יצביע לאיבר המתאים ברשימה.

כל מה שנשאר הוא לקחת את הסיסמא. נגדיר משתנה ונקצה לו זיכרון בגודל של  $dwPassLen + 1$ .  
 ב-`dwPassLen` בשביל תוכן הסיסמא ו-1 בשביל לסיים את המחרוזת עם `null character`:

```
char *password;
password = (char*)malloc(Member->dwPassLen + 1);
```

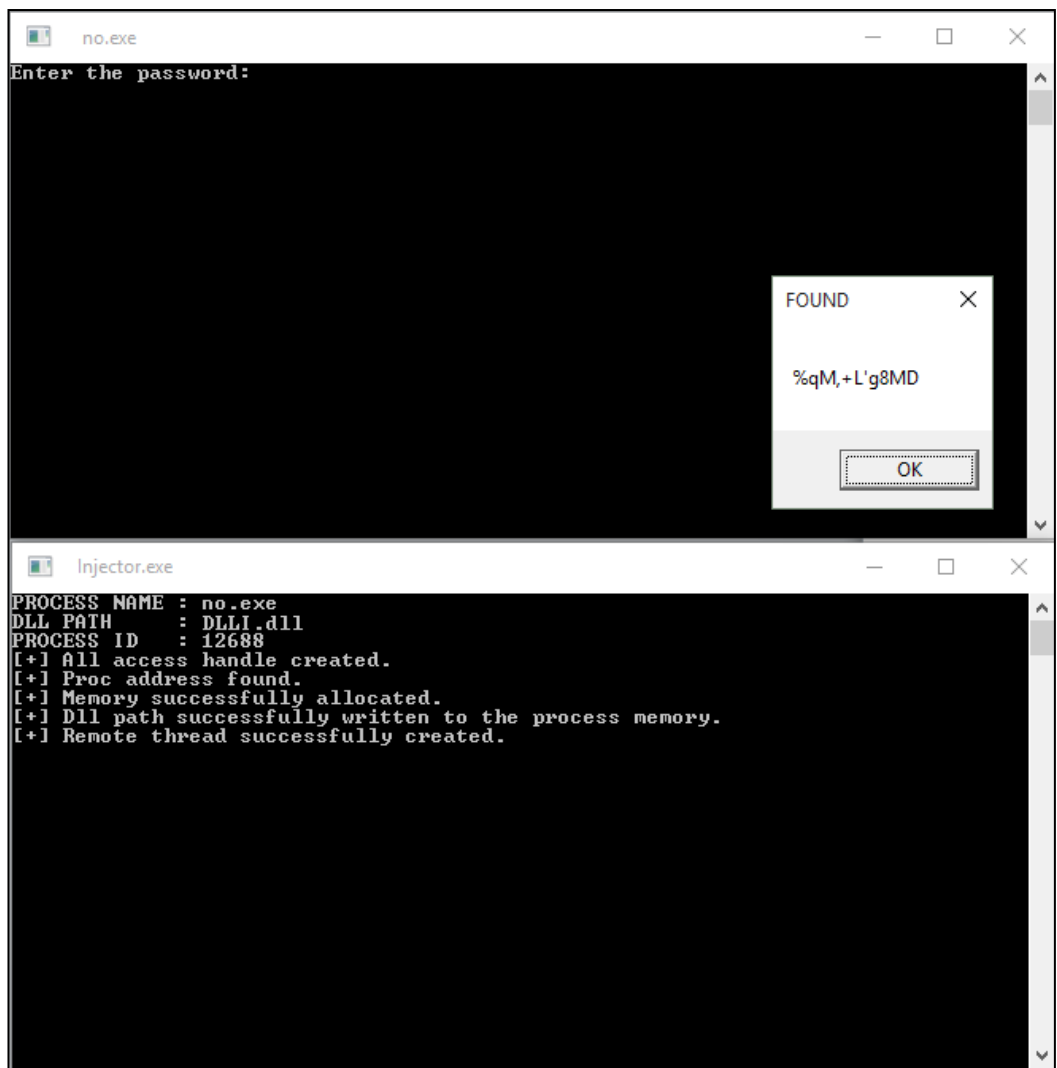
נעתיק את המידע שעליו מצביע `lpPassword` אל ה-`buffer` שלנו ונתחום את המחרוזת:

```
memcpy(password, Member->lpPassword, Member->dwPassLen);
password[Member->dwPassLen] = 0;
```

הפרמטר השלישי ב-`memcpy` הוא כמות הבתים שיש להעתיק. במקרה שלנו זה אורך הסיסמא שמצוין ב-`dwPassLen`. נקפיץ את הסיסמא כהודעה ובזה מסתיים הקוד:

```
MessageBoxA(NULL, password, "FOUND", MB_OK);
```

נריץ ונבדוק את התוצאה. ראשית נריץ את ה-`crackme`, ולאחר מכן נזריק לו את ה-DLL:



בטוח שאתה זוכר נכון?  
[www.DigitalWhisper.co.il](http://www.DigitalWhisper.co.il)

נכניס את התוצאה:



זו הייתה הדרך הראשונה. עכשיו אציג את הדרך ללא הזרקת DLL, ואסביר רק את ההבדלים בין הגרסאות.

## פיתרון 2: בלי DLL injection

בשיטה זו אנחנו לא נעבוד ישירות מול הזיכרון של התהליך, מכיוון שלכל תהליך מרחב כתובות פרטי משלו אנחנו לא נוכל פשוט לפנות לכתובות ולמשוך מהן ערכים. נאלץ להיעזר בפונקציות WINAPI כדי לגשת לזיכרון התהליך. אנחנו נסרוק את כולו ובזיכרון שנסרוק נחפש את המידע הדרוש. בשביל לנהל את הזיכרון שנסרוק, נשתמש ברשימה מקושרת של struct שיעזור לנו. ה-struct ייצג בלוק זיכרון והמבנה שלו יושפע ישירות מהפרמטרים שמקבלת הפונקציה ReadProcessMemory והם:

```
_In_ HANDLE hProcess,
_In_ LPCVOID lpBaseAddress,
_Out_ LPVOID lpBuffer,
_In_ SIZE_T nSize,
_Out_ SIZE_T *lpNumberOfBytesRead
```

ה-struct יראה כך:

```
typedef struct MemoryBlock
{
    HANDLE hProc;
    LPVOID BaseAddress;
    SIZE_T Size;
    char *Buffer;
    struct MemoryBlock *Next;
}MemoryBlock;
```

בטוח שאתה זוכר נכון?

[www.DigitalWhisper.co.il](http://www.DigitalWhisper.co.il)



hProc הוא ה-HANDLE לתהליך ממנו אנחנו קוראים את הזיכרון. BaseAddress הוא המצביע לכתובת הבסיס של הבלוק. Size הוא גודל הבלוק. Buffer הוא ה-buffer בו נאחסן את הזיכרון שנקרא. Next הוא מצביע ל-MemoryBlock הבא ברשימת הבלוקים. פונקציית ה-main שלנו תתחיל בקליטה של שם ה-process מהמשתמש ומציאת ה-PID בדיוק כמו ב-injector שבו השתמשנו קודם. נגדיר משתנה:

```
MemoryBlock *firstBlock;
```

המצביע הזה יצביע לאיבר הראשון ברשימה המקושרת שלנו. נקרא לפונקציה SearchValue שמקבלת pid של תהליך, סורקת אותו ומחזירה מצביע מסוג MemoryBlock\* לאיבר הראשון ברשימה מקושרת:

```
firstBlock = ScanProcess(pid);
```

הפונקציה ScanProcess מתחילה עם הגדרת משתנים:

```
SYSTEM_INFO si;  
MEMORY_BASIC_INFORMATION mbi;  
LPVOID minAddress, maxAddress;  
HANDLE hProc;  
MemoryBlock *firstBlock = NULL, *Block = NULL;
```

ב-si, mbi, minAddress ו-maxAddress שימוש זהה לשימוש בפיתרון עם ה-DLL. hProc יהיה ה-handle לתהליך ה-crackme. firstBlock יצביע לאיבר הראשון ברשימת איברי ה-MemoryBlock ועם Block נרוץ על הרשימה. נמצא את הערכים ל-minAddress ו-maxAddress. עכשיו נפתח handle לתהליך עם הפונקציה OpenProcess:

```
hProc = OpenProcess(PROCESS_QUERY_INFORMATION | PROCESS_VM_READ, NULL, pid);
```

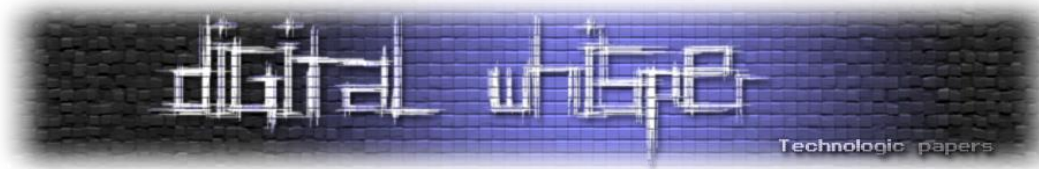
הפרמטר הראשון שמועבר לפונקציה הוא ההרשאות שאנחנו רוצים לקבל על התהליך. במהלך הפיתרון נשתמש בפונקציות שדורשות handle לתהליך. הפונקציות הן VirtualQueryEx ו-ReadProcessMemory. VirtualQueryEx דורשת handle שנפתח עם הרשאות PROCESS\_QUERY\_INFORMATION ו-ReadProcessMemory דורשת handle שנפתח עם הרשאות PROCESS\_VM\_READ ולכן אלה הערכים שאנחנו מעבירים ל-OpenProcess. עכשיו נכנס ללולאה שתראה כך:

```
while (minAddress < maxAddress)  
{  
    VirtualQueryEx(hProc, minAddress, &mbi, sizeof(mbi));  
    if (mbi.State == MEM_COMMIT && mbi.Protect == PAGE_READWRITE)  
    {  
        if (!firstBlock) {  
            firstBlock = CreateMemoryBlock(&mbi, hProc);  
            Block = firstBlock;}  
        else {  
            Block->Next = CreateMemoryBlock(&mbi, hProc);  
            Block = Block->Next;}  
    }  
    minAddress = (LPBYTE)mbi.BaseAddress + mbi.RegionSize;  
}
```

בטוח שאתה זוכר נכון?

[www.DigitalWhisper.co.il](http://www.DigitalWhisper.co.il)





הריצה באמצעות minAddress ו-maxAddress זהה לזו שבפיתרון עם ה-DLL. הפעם אנחנו משתמשים ב-VirtualQuery במקום VirtualQueryEx. הן עובדות בצורה זהה, אך VirtualQueryEx מאפשרת עבודה על זיכרון של תהליך אחר.

אנחנו בודקים את ההרשאות על בלוק הזיכרון, במידה והזיכרון בשימוש ומאפשר קריאה וכתובה ניצור בלוק ונוסיף אותו לרשימה המקושרת שלנו. בתחילת הפונקציה אתחלנו את firstBlock כ-NULL. לכן, אם זה הבלוק המתאים הראשון firstBlock עדיין NULL והרשימה ריקה. ניצור MemoryBlock באמצעות הפונקציה CreateMemoryBlock שמיד אציג ונצביע אליו עם firstBlock. נצביע ל-firstBlock עם Block כדי שנוכל לרוץ על הרשימה מרגע זה ונשמור את ההצבעה על האיבר הראשון באמצעות firstBlock.

במקרה בו firstBlock אינו NULL, Block כבר מצביע על חלק מהרשימה. בין אם על האיבר הראשון או איבר מתקדם יותר. Block->Next יהיה שווה ל-NULL ו-Block יצביע על struct עם מידע חיוני. כדי להמשיך את הריצה על הרשימה ולשמור על המידע החיוני ניצור MemoryBlock חדש ונצביע אליו עם Block->Next. אחרי שיצרנו את ה-MemoryBlock נצביע אל Block->Next עם Block וכך נתקדם ברשימה.

הפונקציה CreateMemoryBlock מקבלת מצביע ל-MEMORY\_BASIC\_INFORMATION struct ו-handle לתהליך של ה-crackme. שני הפרמטרים האלה מכילים את כל המידע הדרוש ל-MemoryBlock struct. הפונקציה נראית כך:

```
MemoryBlock* CreateMemoryBlock(MEMORY_BASIC_INFORMATION *mbi, HANDLE hProc)
{
    MemoryBlock *Block = (MemoryBlock*)malloc(sizeof(MemoryBlock));

    Block->hProc = hProc;
    Block->BaseAddress = mbi->BaseAddress;
    Block->Size = mbi->RegionSize;
    Block->Buffer = (char*)malloc(mbi->RegionSize);
    Block->Next = NULL;

    return Block;
}
```

בפונקציה אנחנו רק מקצים זיכרון לbuffer ולא מבצעים קריאה של הזיכרון, אין סיבה מיוחדת לכך. נחזור ללולאה. הלולאה מסתיימת בעדכון של minAddress בדיוק כמו בפיתרון עם DLL. לאחר סיום הריצה של הלולאה אמורה להיות לנו רשימה מקושרת של struct מסוג MemoryBlock שמכילה את המידע שצריך כדי לקרוא את הזיכרון מהתהליך. נחזיר את המצביע לראש הרשימה ונחזור לmain.

עכשיו כשאנחנו שוב ב-main נקרא לפונקציה ReadMemory שמקבלת מצביע ל-MemoryBlock struct, עוברת על כל האיברים ברשימה וקוראת אל המשתנה Buffer את הזיכרון שאותו הם מייצגים.



הפונקציה ReadMemory מתחילה מהגדרת מצביע מקומי לאיבר הראשון ברשימה כדי לא לשנות את ערך המצביע שמועבר אליה:

```
MemoryBlock *Block = firstBlock;
```

איברים ברשימה נוצרים כך שהאיבר הבא אחרים הוא NULL, לכן האיבר האחרון ברשימה הוא NULL. כדי לעבור על כל האיברים אנחנו נשתמש בלולאה הבאה:

```
while (Block)
{
    ReadProcessMemory(Block->hProc, Block->BaseAddress, Block->Buffer, Block->Size, NULL);
    Block = Block->Next;
}
```

כל עוד Block אינו NULL נקרא זיכרון באמצעות ReadProcessMemory אל המשתנה Buffer ב-struct MemoryBlock ונתקדם ברשימה. אין צורך בהחזרה של מצביע. הקריאה התבצעה על הרשימה שלנו מפני שהעברנו לפונקציה מצביע לרשימה ולא העתק שלה ולכן כל השינויים נעשו על הרשימה המקורית.

עכשיו כשקראנו את כל הזיכרון הגיע הזמן לחפש את הערך LIST\_HEAD\_MAGIC. נקרא לפונקציה SearchValue שבמקרה הזה מקבלת מצביע לאיבר הראשון ברשימת MemoryBlock וערך שאותו היא תחפש (מעט שונה מהגרסא עם ה-DLL):

```
pValue = SearchValue(firstBlock, LIST_HEAD_MAGIC);
```

SearchValue דומה מאוד בשני הפתרונות. הפעם במקום לבדוק בלוק אחד שהיא מקבלת כ- MEMORY\_BASIC\_INFORMATION היא רצה על רשימה של MemoryBlock ובודקת את ה-Buffer שלהם. נגדיר מצביע מקומי לאיבר הראשון ברשימה מאותה הסיבה שעשינו זאת ב-ReadMemory ומשתנה int שישמש אותנו ללולאת for. נרוץ על הרשימה באופן זהה לריצה ב-ReadMemory אך הפעם הפעולה על כל MemoryBlock תהיה שונה כמובן:

```
while (Block)
{
    for (i = 0; i <= Block->Size - sizeof(DWORD); i++)
    {
        if (*(DWORD*)(Block->Buffer + i) == value)
            return (LPVOID)((LPBYTE)Block->BaseAddress + i);
    }
    Block = Block->Next;
}
```

כל עוד Block שונה מ-NULL לא הגענו לסוף הרשימה והריצה ממשיכה. עבור כל בלוק נשתמש בלולאת for כדי לסרוק את Buffer שלו בחיפוש אחר LIST\_HEAD\_MAGIC (שהועבר לפונקציה כ-value). ההגדרה (i <= Block->Size - sizeof(DWORD)) נועדה למנוע חריגת זיכרון כמו במקרה הקודם. גם כאן נבדוק עבור כל כתובת את ערך ה-DWORD שעליו היא מצביעה. במקרה ומצאנו כתובת המצביעה על הערך LIST\_HEAD\_MAGIC נחזיר את הכתובת.

בטוח שאתה זוכר נכון?

[www.DigitalWhisper.co.il](http://www.DigitalWhisper.co.il)



עכשיו כשיש לנו את הכתובת של LIST\_HEAD\_MAGIC אפשר לקרוא ל-Roll. במקרה שלנו מקבלת את הכתובת של LIST\_HEAD\_MAGIC ואת ה-handle לתהליך ה-crackme. היא זקוקה לו כדי לבצע קריאות זיכרון נוספות מהתהליך, למרות שהמידע כבר אצלנו במשתני Buffer ברשימה הקוד פשוט יותר כך. הפונקציה מתחילה עם הגדרת המשתנים הבאים:

```
PASSWORD_LINKED_LIST_POINTER *ListPointer =  
(PASSWORD_LINKED_LIST_POINTER*)malloc(sizeof(PASSWORD_LINKED_LIST_POINTER));  
  
PASSWORD_LINKED_LIST_MEMBER *Member =  
(PASSWORD_LINKED_LIST_MEMBER*)malloc(sizeof(PASSWORD_LINKED_LIST_MEMBER));  
  
char *pass;
```

ListPointer הוא המצביע לאיבר הראשון ברשימה שמכיל את dwMagic. Member יאפשר לנו לרוץ על הרשימה ולחפש את הסיסמא. עד עכשיו הרשימה שעליה דיברנו היא רשימת MemoryBlock, בפונקציה Roll הרשימה היא רשימת הסיסמאות ששמורות ב-PASSWORD\_LINKED\_LIST\_MEMBER struct. ב-pass נשמור את הסיסמא שנמצא. נקרא זיכרון מהתהליך בכתובת של LIST\_HEAD\_MAGIC:

```
ReadProcessMemory(hProc, pValue, ListPointer, sizeof(PASSWORD_LINKED_LIST_POINTER),  
NULL);
```

אנחנו קוראים זיכרון בגודל ה-PASSWORD\_LINKED\_LIST\_POINTER struct מהכתובת של LIST\_HEAD\_MAGIC. LIST\_HEAD\_MAGIC הוא הערך שיושב ב-dwMagic הלא הוא המשתנה הראשון ב-PASSWORD\_LINKED\_LIST\_POINTER struct. כפי שהסברתי, כתובת הבסיס של struct היא הכתובת של המשתנה הראשון שלו. לכן, בהנחה והכתובת שיש לנו היא הכתובת הנכונה, קריאת זיכרון ממנה בגודל של ה-struct הרצוי היא בעצם קריאה של ה-struct. עכשיו כשיש לנו את המצביע לרשימה, נגדיר את האיבר הראשון ברשימה.

בשביל זה נקרא זיכרון בצורה דומה, הפעם ל-Member. הכתובת שממנה נקרא היא הכתובת שנמצאת ב-ListPointer->FirstMember. גם הפעם נקרא זיכרון בגודל המתאים ל-struct:

```
ReadProcessMemory(hProc, ListPointer->FirstMember, Member,  
sizeof(PASSWORD_LINKED_LIST_MEMBER), NULL);
```

עכשיו כשיש לנו את האיבר הראשון ברשימה אפשר להתחיל לרוץ עליה ולחפש את הסיסמא שה-dwPassId שלה הוא ROCKET\_SYSTEM\_PASSWORD, הרי אנחנו מחפשים את הסיסמא למערכת הטיילים.

כל עוד האיבר ברשימה שאליו אנחנו מצביעים עם Member לא מאחסן את הסיסמא המתאימה נתקדם על ידי קריאת זיכרון מהתהליך בכתובת של Member->NextMember אל Member.



ברגע ש-Member->dwPassId שווה ל-ROCKET\_SYSTEM\_PASSWORD הגענו לסיסמא שאנחנו מחפשים. נקצה ל-pass זיכרון בגודל Member->dwPassLen ונקרא אליו את הסיסמא. בנוסף, נתחום את המחזורות עם null character. לאחר מכן נדפיס את הסיסמא:

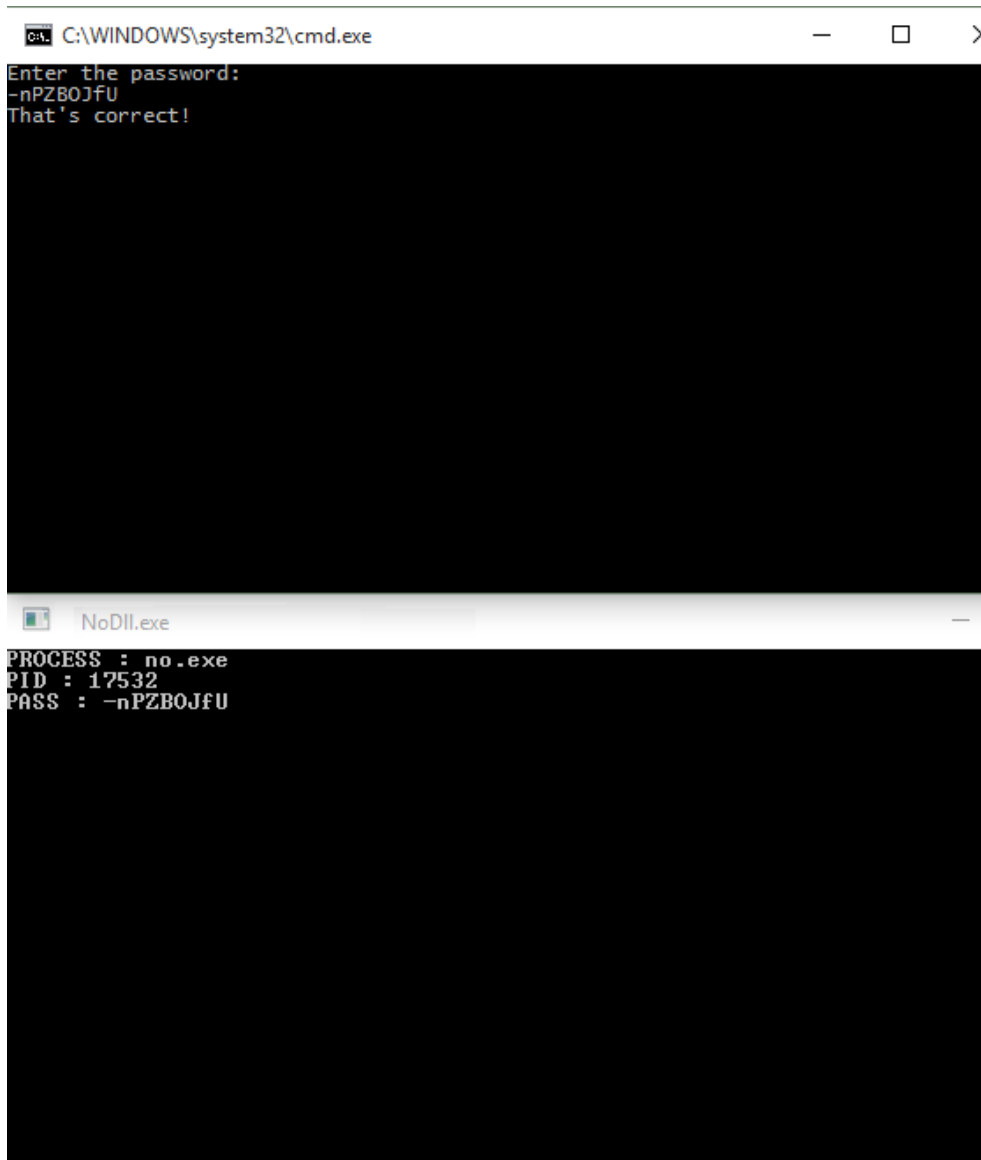
```
while (Member->dwPassId != ROCKET_SYSTEM_PASSWORD) ReadProcessMemory(hProc, Member->NextMember, Member, sizeof(PASSWORD_LINKED_LIST_MEMBER), NULL);

pass = (char*)malloc(Member->dwPassLen + 1);

ReadProcessMemory(hProc, Member->lpPassword, pass, Member->dwPassLen, NULL);
pass[Member->dwPassLen] = 0;

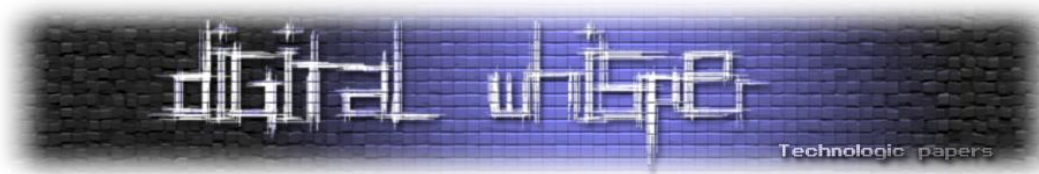
printf("PASS : %s\n", pass);
```

נריץ את ה-crackme ולאחר מכן את הקוד שכתבנו:



בטוח שאתה זוכר נכון?

[www.DigitalWhisper.co.il](http://www.DigitalWhisper.co.il)



זהו. פתרנו את ה-crackme בשתי דרכים, עם הזרקת DLL וללא הזרקת DLL. ההקדמה כנראה גרמה לקוד להזכיר Cheat Engine, אבל הפונקציות שכתבנו מאפשרות בסיס לקוד להיות הרבה יותר קרוב לתוצאה הסופית ממה שנדמה.

## Cheat Engine

נניח בצד את פתרון ה-crackme עם ה-DLL ונבחן את הפתרון השני. באמצעות הפונקציות שכתבנו בו וה-MemoryBlock struct יש לנו את היכולת לסרוק זיכרון של תהליך, ולמצוא בו מיקומים של ערכי DWORD. בתחילת המאמר דיברתי על כך ששימוש נפוץ ב-Cheat Engine הוא שינוי של ערכים מספריים במשחקים. עם הקוד שיש לנו אנחנו יכולים למצוא את הערכים האלה. כל שנשאר הוא לשנות אותם. נפתח פרויקט חדש, נעתיק אליו את ה-MemoryBlock struct ואת הפונקציות CreateMemoryBlock, ScanProcess ו-ReadMemory ו-GetPid (כתובה ב-injector).

נכתוב את main. התחלה, כמו תמיד, עם הגדרת משתנים:

```
DWORD pid, Value, newValue, cheatValue;  
char *pName, *input;  
MemoryBlock *firstBlock;  
Location *firstLocation;
```

pid יישמש אותנו למציאת התהליך אותו נסרוק. Value הוא הערך שנחפש ונרצה לשנות. newValue יאפשר לנו לצמצם את רשימת הערכים שנמצא על ידי הכנסה של ערך נוסף והשוואה שלו לערכים שנמצאים בכתובות שמצאנו בחיפוש ראשוני. cheatValue יכיל את הערך שנכניס כדי להחליף את הערך המקורי.

pName יכיל את שם התהליך שנסרוק. Input יישמש אותנו לקליטת קלט מהמשתמש. firstBlock יצביע על האיבר הראשון ברשימה של MemoryBlock. firstLocation יצביע על האיבר הראשון ברשימה של Location, struct שנגדיר עוד מעט.

נתחיל ממצאת ה-pid של התהליך לפי שם שהמשתמש מכניס:

```
pName = (char*)malloc(MAX_PATH + 1);  
fgets(pName, MAX_PATH, stdin);  
pName[strlen(pName) - 1] = 0;  
pid = GetPid(pName);
```

נסרוק את התהליך ונקרא את הזיכרון שלו:

```
firstBlock = ScanProcess(pid);  
ReadMemory(firstBlock);
```



הפעם ביצירת ה-handle לתהליך בפונקציה ScanProcess נקרא ל-OpenProcess בצורה שונה. בכלי הזה ברצוננו לכתוב לזיכרון התהליך כדי לשנות בו ערכים מסוימים. בשביל כתיבה לזיכרון התהליך נשתמש בפונקציה WriteProcessMemory שמקבלת כפרמטר handle לתהליך שנפתח עם הרשאות PROCESS\_VM\_OPERATION ו PROCESS\_VM\_WRITE. הקריאה ל-OpenProcess תיראה כך:

```
hProc = OpenProcess(PROCESS_QUERY_INFORMATION | PROCESS_VM_READ | PROCESS_VM_WRITE  
| PROCESS_VM_OPERATION, NULL, pid);
```

אנחנו משאירים את PROCESS\_QUERY\_INFORMATION ו PROCESS\_VM\_READ בשביל סריקת זיכרון התהליך. אפשר להחליף את קבוצת ההרשאות הזו ב-PROCESS\_ALL\_ACCESS והפעולות שנעשה עדיין יעבדו. נקלוט מהמשתמש את ערך ה-DWORD שיש למצוא:

```
printf("DWORD VALUE : ");  
input = (char*)malloc(MAX_PATH + 1);  
fgets(input, MAX_PATH, stdin);  
sscanf(input, "%d", &Value);
```

אין סיבה מיוחדת לשימוש ב-MAX\_PATH, אני בדרך כלל משתמש בו לקלט מחרוזות. הערך שלו הוא 260, האורך המקסימאלי לנתיב קובץ ב-Windows. עכשיו נגדיר את ה-Location struct, מבנה פשוט של רשימה מקושרת של מצביעים. כשנחפש ערך בזיכרון התהליך, נשמור את ההופעות שלו ברשימה כזאת:

```
typedef struct Location  
{  
    LPVOID Address;  
    struct Location *Next;  
}Location;
```

ופונקציה ליצירת Location:

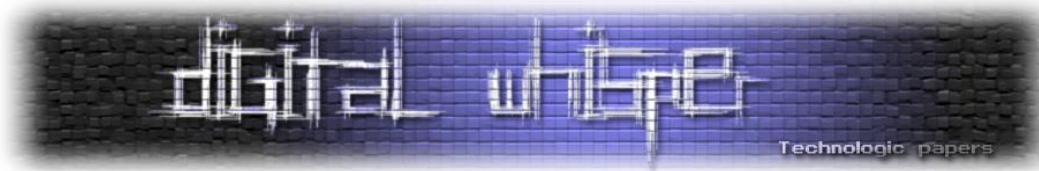
```
Location* CreateLocation(LPVOID Address)  
{  
    Location *l = (Location*)malloc(sizeof(Location));  
    l->Address = Address;  
    l->Next = NULL;  
    return l;  
}
```

נקרא לפונקציה FindLocations, המקבילה ל SearchValue במקרה שלנו, שמקבלת ערך DWORD ומצביע לאיבר ראשון ברשימת MemoryBlock. הפונקציה מחזירה מצביע לאיבר הראשון ברשימת Location:

```
firstLocation = FindLocations(firstBlock, Value);
```

נגדיר את המשתנים שישמשו אותנו בפונקציה:

```
Location *firstLocation = NULL, *tempLocation = NULL;  
MemoryBlock *Block = firstBlock;  
int i = 0;
```



firstLocation יצביע על האיבר הראשון ברשימת Location שנחזיר, tempLocation1 יישמש אותנו בריצה עליה. Block הוא מצביע מקומי לאיבר הראשון ברשימת MemoryBlock שמאפשר לנו לרוץ על הרשימה בלי לפגוע במבנה שלה. i יישמש אותנו בסריקה של Buffer בכל MemoryBlock. הפונקציה כולה תתנהל בזולאת while, בה נרוץ על רשימת ה-MemoryBlock ונחפש אחר הערך Value:

```
while (Block)
{
    for (i = 0; i <= Block->Size - sizeof(DWORD); i++)
    {
        if (*(DWORD*)(Block->Buffer + i) == Value)
        {
            if (!firstLocation)
            {
                firstLocation = CreateLocation((LPBYTE)Block->BaseAddress + i);
                tempLocation = firstLocation;
            }
            else {
                tempLocation->Next = CreateLocation((LPBYTE)Block->BaseAddress + i);
                tempLocation = tempLocation->Next;
            }
        }
    }
    Block = Block->Next;
}
```

כל עוד Block אינו NULL לא הגענו לסוף רשימת ה-MemoryBlock וזהו MemoryBlock שיש לסרוק את ה-Buffer שלו. נרוץ על ה-Buffer בזולאת for, את המבנה שלה הסברתי בשני הפתרונות ל-crackme. במידה ומצאנו את הערך Value נבדוק האם זו הפעם הראשונה שמצאנו אותו. במידה וזו הפעם הראשונה, firstLocation הוא NULL. ניצור Location חדש ונצביע עליו עם firstLocation ועם tempLocation, את ההצבעה של firstLocation לא נשנה יותר. אם firstLocation אינו NULL זו אינה הפעם הראשונה שמצאנו את הערך. ניצור Location חדש ונצביע עליו עם tempLocation->Next. לאחר מכן נצביע על tempLocation->Next עם tempLocation כדי להתקדם ברשימה. לאחר מכן נתקדם ברשימת ה-MemoryBlock באמצעות הצבעה על Block->Next עם Block.

לאחר שסיימנו לרוץ על רשימת ה-MemoryBlock נחזיר את המצביע לאיבר הראשון ברשימת ה-Location. נדפיס את תוכן רשימת ה-Location עם הפונקציה PrintLocationsValues שמקבלת מצביע לאיבר הראשון ברשימת Location ו-handle לתהליך הנסרק. הפונקציה תדפיס את הכתובת שבה הערך נמצא, תקרא את הכתובת שוב ותדפיס את הערך שנמצא בה (במקרים מסוימים הערך בכתובת מספיק להשתנות):

```
PrintLocationsValues(firstLocation, firstBlock->hProc);
```

נגדיר מצביע מקומי לאיבר הראשון ברשימת ה-Location ומשתנה DWORD שאליו נקרא את הערך שנמצא בכתובת שמצאנו:

```
Location *l = firstLocation;
DWORD value = 0;
```

בטוח שאתה זוכר נכון?

[www.DigitalWhisper.co.il](http://www.DigitalWhisper.co.il)



כל עוד | אינו NULL נקרא את הערך בכתובת, נדפיס אותו ואת הכתובת ונתקדם:

```
while (1)
{
    ReadProcessMemory(hProc, l->Address, &value, sizeof(DWORD), NULL);
    printf("ADDR : 0x%p\tValue : %d\n", l->Address, value);
    l = l->Next;
}
```

בחזרה ל-main. נקלוט מהמשתמש ערך DWORD נוסף. הערך הזה נועד לסינון התוצאות שנמצאות ברשימת ה-Location. למשל, למשתמש יש 1,000,000 מטבעות במשחק. המשתמש מכניס בפעם הראשונה את הערך 1,000,00. המשתמש משאיר את הכלי פתוח ומשנה את כמות המטבעות שלו על ידי רכישה כלשהי או רווח מסוים וכעת יש לו 900,000 מטבעות. המשתמש יכניס את הערך 900,000 והכלי יתמקד בכתובות שבהתחלה הכילו את הערך 1,000,000 וכעת מכילות את הערך 900,000. סיכוי גדול שאלה הן הכתובות שמעניינות את המשתמש:

```
printf("NEW VALUE : ");
input = (char*)malloc(MAX_PATH + 1);
fgets(input, MAX_PATH, stdin);
sscanf(input, "%d", &newValue);
```

קעת נדפיס את הכתובות בהן מופיע הערך החדש שהכילו בהתחלה את הערך הראשון. נקרא לפונקציה PrintLocationsValuesNew שדומה מאוד ל-PrintLocationsValues אך משווה את הערכים בכתובות שעליהן היא עוברת ומדפיסה אותם רק אם הם שווים לערך החדש. יש לשים לב שרשימת ה-Location שעליה עוברות שתי הפונקציות זהה ולא עברה שום שינוי. נקרא לפונקציה:

```
PrintLocationsValuesNew(firstLocation, firstBlock->hProc, newValue);
```

הפונקציה החדשה נראית כך:

```
Location *l = firstLocation;
DWORD value;

while (1)
{
    ReadProcessMemory(hProc, l->Address, &value, sizeof(DWORD), NULL);
    if (value == newValue) printf("ADDR : 0x%p\tValue : %d\n", l->Address, value);
    l = l->Next;
}
```

כאמור, התוספת היחידה היא השוואה בין value ל-newValue. הגיע רגע האמת, נקלוט מהמשתמש את הערך החדש שיחליף את הערך הקיים ובצע את הדריסה שלו. נתחיל מקליטת הערך החדש:

```
printf("CHEAT VALUE : ");
input = (char*)malloc(MAX_PATH + 1);
fgets(input, MAX_PATH, stdin);
sscanf(input, "%d", &cheatValue);
```

בטוח שאתה זוכר נכון?

[www.DigitalWhisper.co.il](http://www.DigitalWhisper.co.il)



נקרא לפונקציה שתדרוס את הערכים המקוריים:

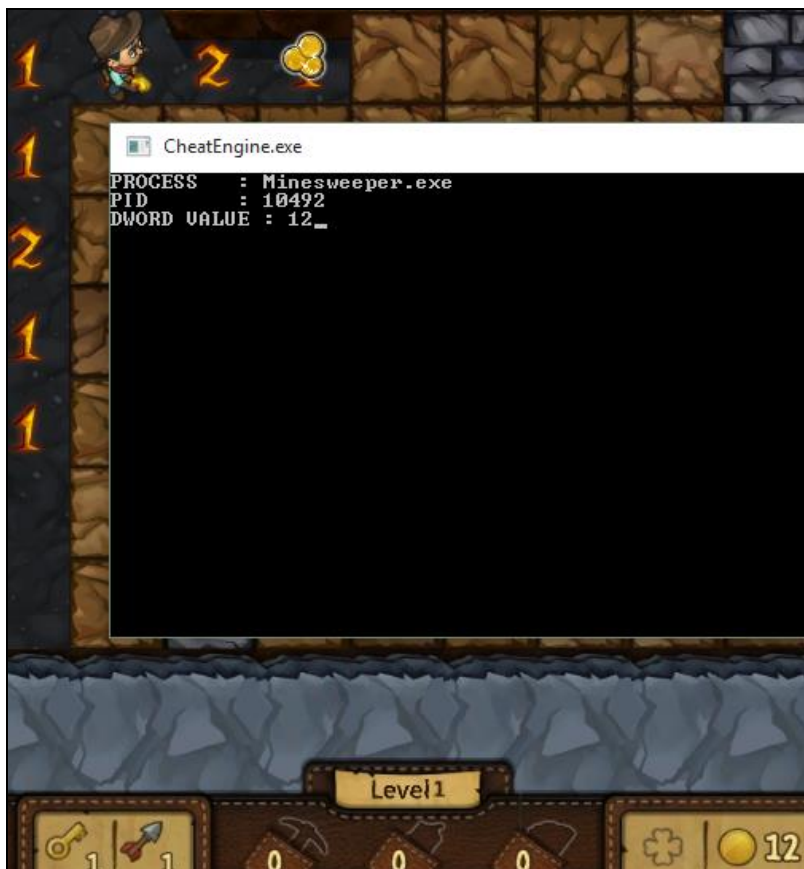
```
WriteNewValue(firstLocation, firstBlock->hProc, newValue, cheatValue);
```

הפונקציה דומה מאוד ל-PrintLocationsValuesNew אך הפעם אם קיים שוויון היא דורסת את הערך המקורי עם cheatValue. הפונקציה נראית כך:

```
Location *l = firstLocation;
DWORD value;
while (1)
{
    ReadProcessMemory(hProc, l->Address, &value, sizeof(DWORD), NULL);
    if (value == newValue) WriteProcessMemory(hProc, l->Address,
(LPVOID)&cheatValue, sizeof(DWORD), NULL);
    l = l->Next;
}
```

הדריסה של הערך המקורי נעשית באמצעות הפונקציה WriteProcessMemory שמקבל לתהליך, כתובת שאליה יש לכתוב, את הערך שיש לכתוב וגודל הערך שברצוננו לכתוב. הפרמטר האחרון מחזיר את כמות הבתים שנכתבו, נתעלם ממנו (במצב תקין הוא צריך להיות זהה לערך שהעברנו לפונקציה המציין את גודל הערך שברצוננו לכתוב).

הכלי מוכן. נבדוק אותו על מוד ה-Adventure של שולה המוקשים. נכנס למשחק ונריץ את הכלי:



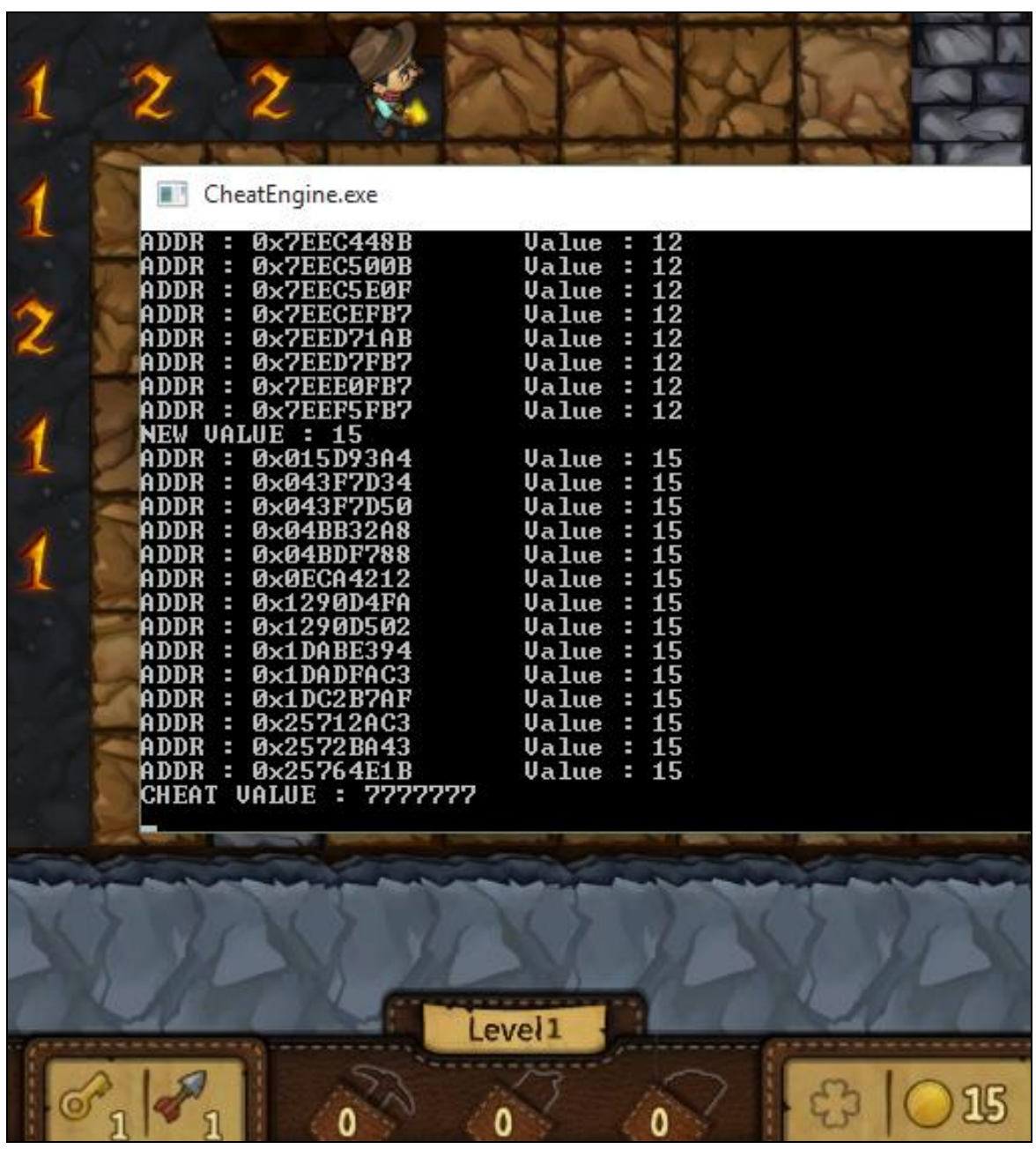
בטוח שאתה זוכר נכון?

[www.DigitalWhisper.co.il](http://www.DigitalWhisper.co.il)

יש לנו 12 מטבעות. בכלי נבחר את התהליך Minesweeper.exe, תהליך המשחק. נכניס את הערך הראשוני שברצוננו לחפש, 12. לאחר שהכנסנו את הערך 12 הכלי ידפיס את הכתובות שבו מצא את הערך. יש המון כתובות כאלה. כדי לצמצם את הכתובות נאסוף את המטבעות שנמצאים לידנו:



עכשיו כשיש לנו 15 מטבעות נכניס את הערך 15. הכלי יעבור על הכתובות שבהן מצא את הערך 12 וידפיס רק את אלו שעכשיו מכילות את הערך 15:

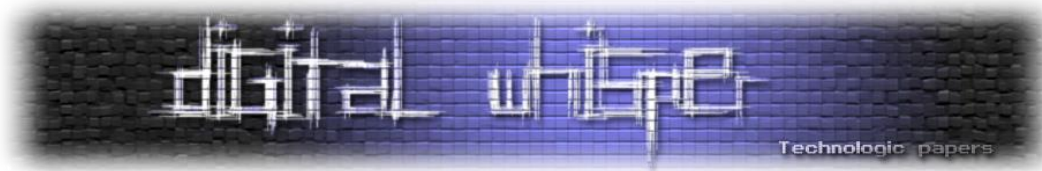


גם עכשיו יש מספר גדול יחסית של תוצאות, אך משמעותית קטן יותר מהתוצאות של הסריקה הראשונה שהניבה עשרות תוצאות. נכניס את הערך האחרון. זוהי כמות המטבעות שאנחנו רוצים. כאמור, ההתעסקות הזו היא בעיקר ניסוי וטעיה ויכולה להקריס את התהליך. בזה הסתיימה פעולת הכלי. למרות שעל המסך עדיין מוצג הערך 15 זו אינה באמת כמות המטבעות שיש לנו במשחק. כמות המטבעות כאמור שמורה כערך DWORD בזיכרון התהליך וההצגה של המטבעות על המסך היא כנראה מחרוזת. כדי לראות את השינוי בכמות המטבעות נאסוף עוד קצת מטבעות במשחק. המשחק יחשב את סכום כמות המטבעות שיש לנו (ערך ה-DWORD) וכמות המטבעות שנאסוף:

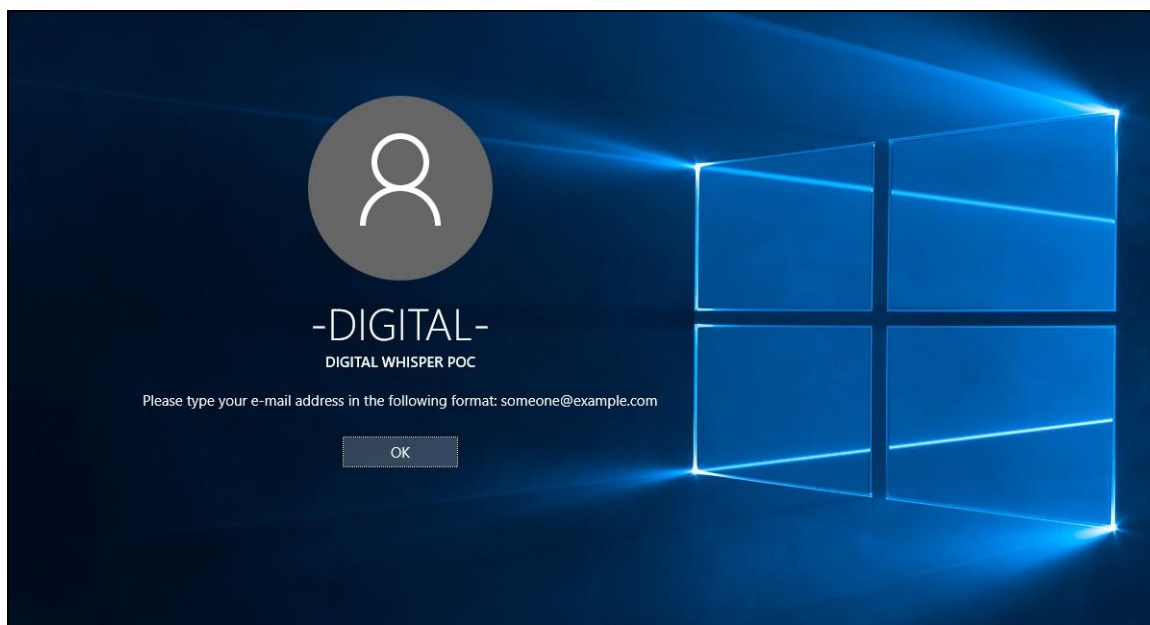


רימינו את המשחק 😊. נוכל להוסיף גם פצצות, פסילות וכל דבר שניתן לצבור במשחק.





אפשר גם להוסיף אפשרות לשינוי של ערכים מסוגים שונים, מחרוזות למשל:



## סיכום

בכתבה אמנם חיפשנו ערכי DWORD אך אפשר בקלות לחפש כל סוג ערך אחר (נסו את הפונקציה memcmp). הכלי שפיתחנו בעל פוטנציאל חזק ואידיאלי לערכים גלויים למשתמש. ניתן לקחת את הליבה של הכלי ולשכלל אותו כך שיתאים לצרכים אישיים. אני באופן אישי אשתמש בו בכמה משחקים, למרות שממוד ה-Adventure של שולה המוקשים הוא הוציא לי את הכיף. ניתן ליצור קשר במייל [hyprnir@gmail.com](mailto:hyprnir@gmail.com).

## תודות

- תודה לדור אריאלי שעזרה לי בכתיבה ונתנה לי ביקורת כנה.
- תודה לדימה פשול שכתב את ה-crackme ועזר לי בכתיבת המאמר.

בטוח שאתה זוכר נכון?

[www.DigitalWhisper.co.il](http://www.DigitalWhisper.co.il)

---

# אם תמונה שווה אלף מילים, כמה מילים שווה חולשה באינסטגרם?

מאת עדן אלון

---

## הקדמה

בזמן האחרון מספר לא מובטל של חברות וארגונים גדולים מאמצים שיטה חדשה בתחום המחקר ואבטחת המידע הנקראת - Bug Bounty. התוכנית מאפשרת לכל חוקר אבטחה לנסות למצוא פגיעות בשירותי החברה ובתמורה לכל דיווח אמין שייבדק וימצא נכון - יתוגמל החוקר בתשלום בהתאם לסוג חולשה שדווח. בזמני הפנוי אני חוקר לא מעט מערכות ושירותים נפוצים במטרה למצוא פגיעות ולדווח עליהם.

לאחר הרבה מחקרים מוצלחים, החלטתי להתקדם קצת ולנצל את ניסיוני עם שירותים יותר גדולים - פייסבוק, גימייל, וואלה ועוד... את המאמר על וואלה נשמור ליום שבו וואלה יחליטו לתקן את בעיות האבטחה שלהם ☺

המחקר שלי התחיל בחיפוש אחר XSS באינסטגרם והתפתח בסופו לחולשה לוגית קריטית המאפשרת שינוי סיסמה עבור כל חשבון אינסטגרם.

לאינסטגרם עבר לא רע בתוכנית ה-Bug Bounty, כאשר ב-7 במאי ב-2013 מצא חוקר אבטחת מידע בשם Sebastián Guerrero חולשה יפה ופשוטה מאוד שמאפשרת לצפות בתמונות של כל משתמש, כולל משתמשים פרטיים.

סבסטיאן מצא בשירות המפות של אינסטגרם באג ב-API שעוקף את ה-API הפומבי ומאפשר לגשת לכל תמונה בבקשת GET פשוטה. שירות המפות עצמו נזקק לאפשרות הזו על מנת לטעון את התמונות של משתמשים ולהציג אותם על המפה בהתאם למיקום שלהם.

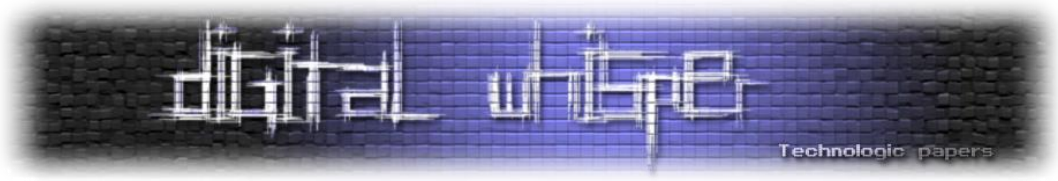
בשירות ה-API שהמפות עבדו איתו, (לא הפומבי) אינסטגרם לא חשבו על מקרה של משתמשים המוגדרים כ-Private ולכן כל בקשת תמונה החזירה תמונה עבור כל משתמש. לקריאה מורחבת על החולשה של סבסטיאן:

<https://www.nowsecure.com/blog/2013/06/28/how-i-hacked-your-instagram-account/>

---

אם תמונה שווה אלף מילים, כמה מילים שווה חולשה באינסטגרם?

[www.DigitalWhisper.co.il](http://www.DigitalWhisper.co.il)



## אז... איך אינסטגרם עובד?

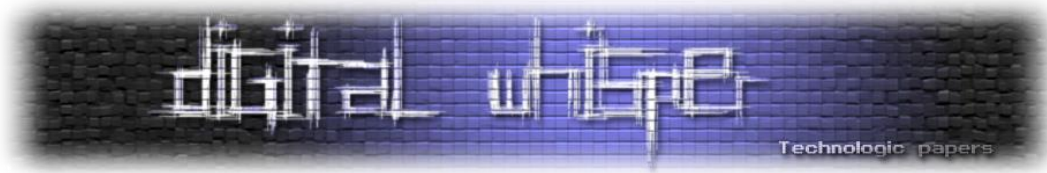
כיום אינסטגרם מתבססת בעיקר על האפליקציה שלה בפלטפורמת המובייל ולאחרונה השיקה אתר מותאם שמאפשרת שימוש גם מהדסקטופ וגם מהנייד.

האתר מוגבל מאוד מבחינת הפונקציות שבו לעומת האפליקציה ולכן התחלתי לחקור אותו. בכל זאת, כדאי להתחיל מקטן ולהתקדם לאט. הכיוון הראשון שהתחלתי בחיפושים אחריו היה פגיעות XSS. הפגיעה מאוד נפוצה ופשוטה לניצול לעומת שאר הפגיעות (RCE,SQLI ועוד).

לאחר כמה שעות שלא מצאתי דברים מעניינים\פגיעים באתר עצמו החלטתי לעבור למחקר של האפליקציה שמכילה הרבה יותר פונקציות ומגוון רחב יותר של פעולות. בחקירת האפליקציה של פלטפורמת המובייל (שתואר בהמשך) החלטתי ללכת על כיוון של שליחת הודעות פרטיות מסוג Direct. הפונקציה חדשה באינסטגרם ואולי יש בה חורים שטרם נסגרו.

השיטה בה אינסטגרם משתמשת כדי למנוע הרצה של קוד ברמת הדפדפן ידועה ונפוצה - תווים מיוחדים יומרו וישלחו בקידוד מסוג Unicode והאפליקציה תמיר אותם בהצגה בלבד. כך נראות הפקטות:

אם תמונה שווה אלף מילים, כמה מילים שווה חולשה באינסטגרם?



יש לא מעט כיוונים שטרם חקרתי כמו יצירת מיקום במפה עם שסלתיאור הכוללים XSS, חיפוש (באפליקציה), XSS בתגובה כאשר מקבל ההתראה יפתח את דף ההתראות וירוך שם XSS (לא בתצוגת התגובה עצמה) ועוד...

על מנת להבין ולקבל קצת רקע על האפליקציה ואיך היא עובדת השתמשתי במגוון כלים לביצוע Decompile לקובץ ה-APK של האפליקציה.

כיום יש עשרות כלים באינטרנט על מנת להמיר APK לקוד JAVA. בין היתר השתמשתי ב:

- <http://forum.xda-developers.com/showthread.php?t=1910873> - [TOOL] APK to Java RC2
- [http://www.neshkov.com/ac\\_decompiler.html](http://www.neshkov.com/ac_decompiler.html) - AndroChef Java Decompiler
- <http://bytecodeviewer.com/> - Bytecode Viewer - Java & Android APK Reverse

הכלים הללו נתנו לי קצת רקע בסיסי להבין איך האפליקציה עובדת, עם מי היא מדברת, ופרטים אודות בקשות ה-POST שהיא מוציאה בכל פעולת משתמש.

השלב הבא הוא להקליט את הפקטות, להבין את התקשורת וה API אותו האפליקציה מממשת. מכיוון שהאפליקציה מתקשרת SSL - נהיה חייבים להשתמש במכשיר פרוץ (Rooted) או לחילופין להתקין תעודת SSL משלנו שבעזרתה נוכל לקרוא ולפענח את ההצפנה.

שלל ניסיונות עם Fiddler ו-Burp נכשלו תחת ההודעה "No internet connection" (רק באפליקציה הזו). ולבסוף האפליקציה שהשתמשתי על מנת להתמודד עם הבעיה נקראת "Packet Capture":

- <https://play.google.com/store/apps/details?id=app.greyshirts.sslcapture>

האפליקציה יוצרת חיבור VPN דרכו תצא התעבורה של כל האפליקציות במכשיר, מקליטה את כל התעבורה ושומרת בצורה נוחה המאפשרת קריאה ישירה מתוך האפליקציה או לחילופין לקובץ pcap רגיל לחלוטין עם תעבורה לא מוצפנת.

בטעות (או שלא) - כאשר האפליקציה מזהה VPN, היא לא מוציאה בקשות ומחזירה שגיאה למשתמש שיש בעיה באינטרנט (כמו קודם - "No internet connection").



קצת מחקר של הקוד יכול להוביל ל-Flow מאוד גדול וארוך (בכל זאת, השימוש בכלי Decompiler אמור לתת רקע ולא קוד מושלם ויפה):

```
189     this.n.a(var1, var2);
190 }
191
192 public final void a(int var1, com.instagram.android.trending.d.c var2, int var3) {
193     this.n.a(com.instagram.android.trending.marquee.a.a(var3, var1), var2);
194 }
195
196 public final void a(com.instagram.common.o.a.k var1) {
197     if(this.isResumed()) {
198         Toast.makeText(this.getActivity(), com.facebook.ab.could_not_refresh_feed, 0).show();
199     }
200
201     this.k.notifyDataSetChanged();
202 }
203
204 public final void a(com.instagram.feed.d.ay var1, int var2) {
205     this.m.a(var1, var2);
206     this.b.b();
207     this.r.a(var2);
208 }
209
210 public final void a(String var1) {
211     com.instagram.android.feed.g.i.a((com.instagram.common.analytics.g)this);
212     com.instagram.t.d.h.a().a(this.getFragmentManager(), var1).a();
213 }
214
215 public final void a(List var1) {
216     com.instagram.base.a.b.a var3 = com.instagram.t.d.h.a().w(this.getFragmentManager());
217     Bundle var2 = new Bundle();
218     if(var1 != null && !var1.isEmpty()) {
219         var2.putStringArrayList(cc.a, (ArrayList)var1);
220     }
221
222     var3.a(var2).a();
223 }
```

מבחינת המחקר - הגעתי לנקודה בעייתית, לאחר מעט מחשבה והסתכלות על מחקרים קודמים שעשיתי - החלטתי לתת צ'אנס גם הפעם ולחפש אחר גרסה ישנה שעדיין כוללת את רוב הפונקציות שיש כיום.

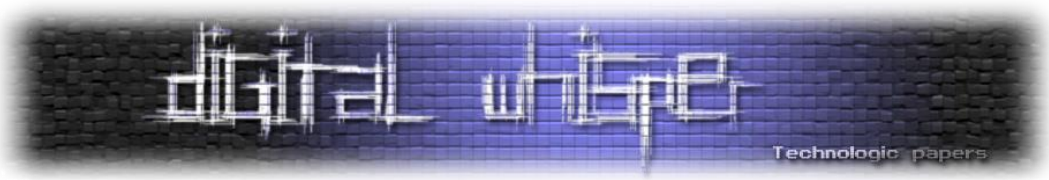
לאחר חיפוש קצר בגוגל אחר Instagram APK ראיתי גרסאות ישנות מאוד כמו 3.7 ועד לגרסה אחרונה 7.6.1 (נכון לכתיבת שורות אלו). לאחר בדיקה קצרה של גרסאות 4,5 ו-6 החלטתי ללכת על גרסה 5.0.0 שעובדת נהדר עם Packet Capture ומצד שני עדיין כללת לא מעט אפשרויות, וכמובן - ניתן לראות תעבורה לא מוצפנת. ניתן להשיג את הגרסה איתה עבדתי בקישור הבא:

<https://www.androidfilehost.com/?fid=23252070760975436>

---

אם תמונה שווה אלף מילים, כמה מילים שווה חולשה באינסטגרם?

[www.DigitalWhisper.co.il](http://www.DigitalWhisper.co.il)



## המעבר מ-XSS לחולשה לוגית - המימוש

כדי שיהיה נוח ונקי - התקנתי אימולטור של אנדרואיד למחשב בשם **Andy** (<http://www.andyroid.net>) שמאוד נוח ומהיר לעומת המתחרים. (BlueStacks, GenyMotion, Droid4X וכו...). בכניסה הראשונית לאינסטגרם שכחתי את הסיסמה והשם משתמש שלי (גם אני רגיל להתחבר באמצעות הפייסבוק/גוגל למגוון שירותים) ובחירתי ב"שכחתי סיסמה". בשלב זה אינסטגרם הציע לי 3 אופציות לאיפוס הסיסמה:

- איפוס באמצעות הודעת סמס.
- איפוס באמצעות אימייל.
- איפוס באמצעות חשבון פייסבוק.

האופציה האחרונה משכה את תשומת ליבי - התהליך מתאפשר רק למי שחשבון הפייסבוק שלו מקושר עם חשבון האינסטגרם (כך רוב המשתמשים).

מרגע זה החלטתי לעצור את הכיוון של XSS ולשנות כיוון למחקר על תהליך איפוס הסיסמה. (אגב, מחקר ה-XSS לא ננטש ובסוף לאחר החולשה הלוגית המשכתי בו ללא הצלחה מרובה...). השלב הבא הוא לבחון את הקוד JAVA שקבלתי בשלב הראשון בשילוב עם הסנפת התעבורה ולהבין היטב כיצד עובד התהליך.

דוגמה לפקטות:

### לפני פתיחת ההצפנה:

```

→ Instagram

#5 <--- 09-19 20:41:39

POST /api/v1/accounts/change_password/ HTTP/1.1
Content-Length: 435
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
Host: instagram.com
Connection: Keep-Alive
User-Agent: Instagram 5.0.0 Android (17/4.2.2; 160dpi; 800x1232; Andy OS Inc./AndyOS; AndyWin; AndyWin; andy; iw_IL)
Cookie: csrftoken=35f4c1500596b2dd5f8a4d79d1c73bb3; mid=VF2dmAABAeWp7Ddq5it3uny0jg
Cookie2: $Version=1
Accept-Encoding: gzip
Accept-Language: he-IL, en-US

signed_body=3b4aad804671baf3c694de469fa2ca21ac34970d4a171368da8696603040.%7B%22csrf_token%22%3A%2235f4c1500596b2dd5f8a4d79d1c73bb3%22%2C%22token%22%3A%2245a-1c93b91fde3bea68f462%22%2C%22new_password%22%3A%220522834978A%22%2C%22new_password1%22%3A%220522834978A%22%2C%22guid%22%3A%22808b4d1a-7663-4dcf-b40d-9d8f88c4807%22%2C%22user_id%22%3A%22177462491%22%2C%22device_id%22%3A%22android-39A43CF84A620306%22%2D%2C%22sig_key_version%22%3A%224

#6 ---> 09-19 20:41:40

HTTP/1.1 200 OK
Cache-Control: private, no-cache, no-store, must-revalidate
Content-Encoding: gzip
Content-Language: en
Content-Type: application/json
Date: Sat, 19 Sep 2015 17:41:40 GMT
Expires: Sat, 01 Jan 2000 00:00:00 GMT
Pragma: no-cache
Set-Cookie: csrftoken=5409907224f29e540231c3f18405e7ad; expires=Sat, 17-Sep-2016 17:41:40 GMT; Max-Age=31449600; Path=/
Set-Cookie: sessionid=1655C914e6fa7a76903327e63bc7fe573781e88d1e4af31e961459be4181fb5913db3Apo0rw7lx4XEqB2ry02ggfTdZcu105w13A978%22_token_ver%22%3A%222_auth_user_id%22%3A%22177462491%22_token%22%3A%22177462491%3A1a914eovPmXWzF1x2DQY120T2jBXA5N3Ae2d465d5d663484e718cfb7cc22cd30716382ca3eb7863e4f848f6ab63e%22%2C%22_auth_user_backend%22%3A%22accounts.backends.CaseInsensitiveModeBackend%22%2C%22last_refreshed%22%3A%221442684500.579558%22%2C%22platform%22%3A%22android%22%3A%22177462491%22%2C%22device_id%22%3A%22android-39A43CF84A620306%22%2D%2C%22sig_key_version%22%3A%224

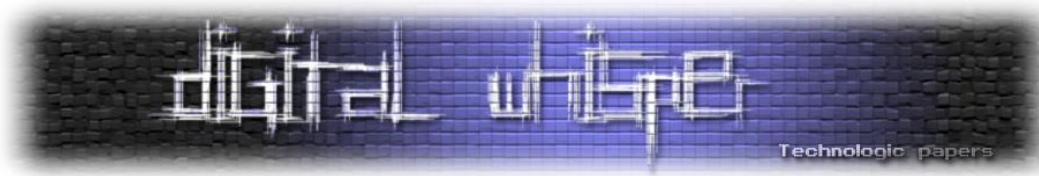
#7 <--- 09-19 20:41:41

GET /api/v1/direct_share/recent_recipients/ HTTP/1.1
Host: instagram.com
Connection: Keep-Alive
User-Agent: Instagram 5.0.0 Android (17/4.2.2; 160dpi; 800x1232; Andy OS Inc./AndyOS; AndyWin; AndyWin; andy; iw_IL)
Cookie: mid=VF2dmAABAeWp7Ddq5it3uny0jg
Cookie2: $Version=1
Accept-Encoding: gzip
Accept-Language: he-IL, en-US

```

אם תמונה שווה אלף מילים, כמה מילים שווה חולשה באינסטגרם?

[www.DigitalWhisper.co.il](http://www.DigitalWhisper.co.il)



## לאחר פתיחת ההצפנה:

Decoded as HTTP

HEURISTIC

<---

POST /api/v1/accounts/change\_password/ HTTP/1.1  
Content-Length: 435  
Content-Type: application/x-www-form-urlencoded; charset=UTF-8  
Host: instagram.com  
Connection: Keep-Alive  
User-Agent: Instagram 5.0.0 Android (17/4.2.2; 160dpi; 800x1232; Andy OS Inc./AndyOS; AndyWin; AndyWin; andy; iw\_IL)  
Cookie: csrftoken=35f4c1500596b2dd5f8a4d79d1c73bb3; mid=VF2dmAABAEEWP70dq5it3unyjg  
Cookie2: \$Version=1  
Accept-Encoding: gzip  
Accept-Language: he-IL, en-US

<---

signed\_body=3b4aadb04671bafc36947deb4e469fa2ca21ac34970da171368da8696603040.{"\_csrftoken":"35f4c1500596b2dd5f8a4d79d1c73bb3","token":"45a-1c93b91fde3bea68f462","new\_password2":"0522834978A","new\_password1":"0522834978A","guid":"80dbdd1a-7663-4dcf-b40d-9d8f88cc4b07","user\_id":"177462491","device\_id":"android-39A43CF84A62D3D6"}&ig\_sig\_key\_version=4

--->

HTTP/1.1 500 INTERNAL SERVER ERROR  
Cache-Control: private, no-cache, no-store, must-revalidate  
Content-Language: en  
Content-Type: text/html; charset=utf-8  
Date: Sat, 19 Sep 2015 17:41:20 GMT  
Expires: Sat, 01 Jan 2000 00:00:00 GMT  
Pragma: no-cache  
Set-Cookie: csrftoken=35f4c1500596b2dd5f8a4d79d1c73bb3; expires=Sat, 17-Sep-2016 17:41:19 GMT; Max-Age=31449600; Path=/  
Vary: Cookie, Accept-Language  
Content-Length: 25  
Connection: keep-alive

--->

Oops, an error occurred.

<---

POST /api/v1/fb/verify\_access\_token/ HTTP/1.1  
Content-Length: 426

תהליך איפוס הסיסמה מתחלק ל-2 שלבים:

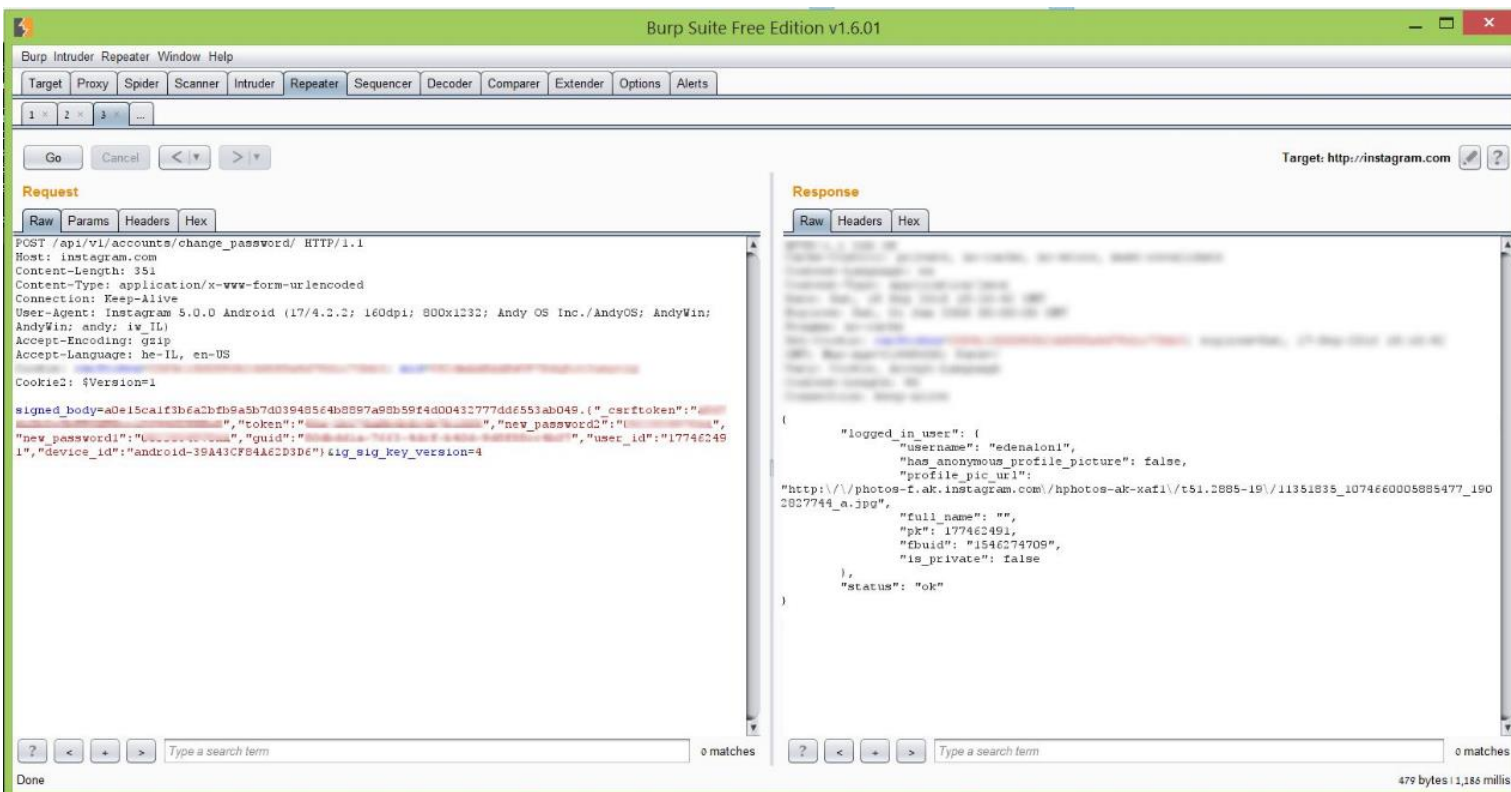
- **שלב ראשון:** תהליך האימות (Authentication) מול חשבון הפייסבוק - מתבצע באמצעות חיבור לאפליקציית פייסבוק לקבלת token של חשבון הפייסבוק וקישור שלו ל-ID של משתמש אינסטגרם.
- **שלב שני:** שלב איפוס הסיסמה עבור ה-ID שהתקבל מפייסבוק עם נתוני הפייסבוק שהתקבלו. שלב זה מתבסס על כך שכבר נעשה קישור ואימות מול המשתמש ואינו מבצע אימות נוסף בתהליך איפוס הסיסמה עצמו. כלומר - בשלב זה במידה ואערוך את הפרמטר של ה-ID של חשבון האינסטגרם שהתקבל לחשבון אחר ואבצע שינוי איפוס לסיסמה החדשה, הסיסמה עבור חשבון המשוך ל-ID המבוקש.

אם תמונה שווה אלף מילים, כמה מילים שווה חולשה באינסטגרם?

[www.DigitalWhisper.co.il](http://www.DigitalWhisper.co.il)



## להלן דוגמה לתשובה מהשרת לאחר שליחת בקשה לאיפוס סיסמה עבור ID מסוים:



בכך השגנו אפשרות לשינוי סיסמה לכל חשבון אינסטגרם. מדובר בחולשה לוגית בשלבי אימות האפליקציה מול המשתמש. במקרה או לא, באחד מאתרי הסחר הנפוצים בחולשות - 1337 (כיום - 0day.today) ישנה חולשה דומה מאוד שנמכרת בסכום לא מבוטל : (רמת האמון שלי באתר לא גבוהה אבל בכל זאת מעניין...)

<http://0day.today/exploit/description/23926>

### התמודדות מול הגנות באפליקציות

כיום לא מעט חברות ואפליקציות מתחילות להבין ולהפנים שהתחום הולך וגודל וכך גם הסכנות ותחום הסייבר נכנס חזק ומהר מאוד. מבלי להיכנס יותר מידי לתחום הרברסינג במובייל, ישנם לא מעט אפליקציות (חלק ציינתי כאן) שמאפשרות להגיע לקוד JAVA קריא יחסית, אך השאלה הגדולה כיצד ניתן באופן פשוט להתמודד עם קטעי קוד לא רצויים עבורנו בתור חוקרים. ישנה תוכנה לא רעה שעושה עבודה יפה בתחום - Virtuous Ten Studio.

התוכנה מאוד נוחה ומאפשרת לערוך את הקוד של האפליקציות החל מעריכה בסיסית ועד לעריכה מתקדמת מאוד עם שלל כלים ואפשרויות.

אם תמונה שווה אלף מילים, כמה מילים שווה חולשה באינסטגרם?

[www.DigitalWhisper.co.il](http://www.DigitalWhisper.co.il)



היתרון המשמעותי הוא היכולת לערוך ולקמפל חזרה ל-APK חדש היישר מה APK המקורי ללא צורך במספר כלים נוספים. מומלצת בחום לכל חוקר מוביל.

## חולשות לוגיות - סיכום

הכיוון לחקור ולחשוב על כיוונים פשוטים אך מתוחכמים שככל הנראה לא חשבו עליהם נובע מהמקום של החשיבה הפשוטה - Keep It Simple. החולשות הלוגיות נובעות בעיקר מטעויות אנוש בהם המפתחים לא דאגו מספיק טוב לאוטנטיקציה מלאה בכל השלבים, ממשקי ניהול חשופים ועוד...

לא מזמן נחשפנו אודות חולשה לוגית ביוטיוב, כאשר נשלחת בקשה למחיקת סרטון באמצעות משתמש מחובר (מאומת ותקין) לא מתבצעת ולידציה האם הסרטון המבוקש למחיקה אכן משויך לאותו חשבון ובכך התאפשר לכל משתמש למחוק כל סרטון באתר. לפרטים נוספים על הפגיעה ניתן לקרוא כאן:

<http://thehackernews.com/2015/04/hack-delete-youtube-video.html>

מאמר נוסף על חולשה בפייסבוק המאפשרת למחוק כל תמונה:

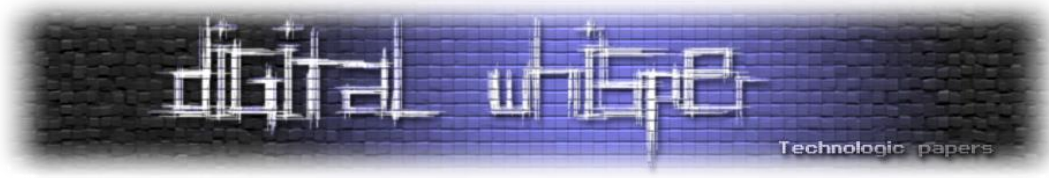
<http://thehackernews.com/2015/02/hacking-facebook-photo-album.html>

בקרב מאוד מקווה לפרסם מאמר נוסף המאפשר לשנות את פרטי החשבון באינסטגרם ובכך להוביל ב-Flow ספיציפי לאיפוסל'ינוי סיסמה. ☺ (פרומו: זה עובד!)

## אודות

עדן אלון: תעקבו באינסטגרם - edenalon1 ;)

אם יש שאלות או הצעות אשמח לענות - [adming4f@gmail.com](mailto:adming4f@gmail.com)



---

## הכרת SCTP - חלק ראשון

מאת עידו קנר

---

### הקדמה

בעולם התקשורת קיים מודל שבע השכבות של ה-OSI. כאשר מדברים על שכבה מספר אחת, כולם יודעים במה מדובר, אך ככול שעולים במספר השכבות, מכירים פחות ופחות מה קורה. יש כאלו שיודעים לזרוק שמות של פעולות ופרוטוקולים מסוימים, אך לא ניתן להכיר הכל.

השכבה הרביעית במספר, היא שכבה אשר הרבה מכירים, כאשר מרבית האנשים המתעסקים במקצוע יודעים לציין כי ישנם שני פרוטוקולים העובדים בשכבה זו, כאשר הפרוטוקול הראשון הוא TCP שמאוד נפוץ, ונחשב לאמין, והשני הוא UDP, אשר בנוי יותר ל-real-time ונחשב לפחות אמין.

העניין הוא שישנם עוד פרוטוקולים, אשר יודעים לשלב פעולות שונות, ומשום-מה אין הרבה אנשים אשר מכירים אותם, כדוגמת: RSVP, DCCP ואחרים. כל פרוטוקול שכזה בעל תפקיד מאוד מוגדר ברמת התעבורה. למשל RSVP יודע לשמור על משאבים מסוימים לתעבורה, בעוד ש-DCCP מספק גישה של UDP עם חלקים קטנים יותר המגיעים מעולם ה-TCP.

מאמר זה נכתב על מנת ללמד על אחד מפרוטוקולים אלו, אשר קיבל את השם: Stream Control Transmission Protocol או בשמו המקוצר SCTP.

המאמר בנוי בשני חלקים, כאשר החלק הראשון הוא הסבר המאפשר להבין על מה מדובר, והחלק השני מדבר על כיצד הפרוטוקול בעצם בנוי בפועל - כלומר הסבר של מבנה הפקטה עצמה.

מאמר זה אינו מגיע להיות שלם, ויוצא מנקודת הנחה כי הקורא מעולם לא הכיר או לא נכנס לעובי הקורה בנושא הפרוטוקול, אך כן בעל ידע והבנה ברשתות, ולכן מנסה להישאר בכיוון הלימוד של הפרוטוקול ללא הלימוד של עולם הרשתות (TCP/IP). בנוסף לאמור, אין המאמר מנסה ללמד על תתי פרוטוקולים אשר נמצאים בשימוש, והם אינם ייחודיים לפרוטוקול זה.

## חלק ראשון

על רגל אחת, וללא עשיית סדר בנושא, עולם הרשתות של TCP/IP הציג לנו רעיון מאוד מעניין - לבנות דברים בגישה שכל חלק יודע להתמחות במשהו אחד. לחלקים האלו, סיפקו את השם מודל שבע השכבות של OSI.

החלק הראשון ברשימה הוא חיבור פיזי, החלק השני ברשימה הוא כרטיס רשת, השלישי היא רשת ה-IP הרביעית זה כיצד המידע יעבור אלינו עד אשר מגיעים לחלק השביעי שהוא המידע שהתוכנה הסופית צריכה לקחת ולממש.

יחסית בהתחלה, נוצרו שני פרוטוקולים אשר מעבירים את המידע, אחד בשם TCP, אשר יודע לנהל את כל נושא התעבורה ברמה שלו, והשני הוא UDP, אשר מעביר את המידע בלי יותר מידי לבדוק דברים, ובכך מאפשר תעבורה מהירה יותר, אך יש איתו לא מעט בעיות, לדוגמא - שאת הבדיקות צריך להיעשות ברמת התוכנה הסופית (שכבה מספר 7), ולא ברמת התעבורה.

אילו בדיקות? למשל האם הסדר הגיע נכון, האם הגיעו כל החלקים, או האם התקשורת הגיע בכלל ליעד. כל פרוטוקול ברמה הגבוהה ביותר, צריך להכיל בתוכו מידע אשר יסייע לנושא, בעוד שב-TCP, המידע הזה נמצא בשכבה מספר 4, הכוללת גם ניהול פעולות congestion - כלומר כמה פעמים לחזור על בקשה עד אשר נכריז עליה שהיא לא הצליחה.

לרוב שתי גישות אלו מספיקות, אך, ככול שהזמן עבר, נמצאו עוד מקרי קצה אשר בהם צריכים לשלב תכונות שונות בין TCP לבין UDP, ונוצרו עוד פרוטוקולים חדשים, אשר מנסים לספק מענה שכזה.

## קיצורים ומונחי יסוד

כאשר מדברים על SCTP ישנם מספר קיצורים ומונחי יסוד, אשר חלקים גם מופיעים כאן במאמר:

פירוש	מונח
Hash Message Authentication Code - גישה לבדיקת תקינות המבוססת על פונקציות Hash קריפטוגרפיות (בגישה דומה ל-HMAC - RFC2104 אך לרוב לא זהה) באמצעות מפתח פרטי - סימטרי - כלומר משותף לכל הצדדים הנוגעים בדבר, על מנת לוודא את אמינות המידע בין הצדדים.	MAC
Retransmission Timeout	RTO
Round-Trip Time	RTT
Stream Control Transmission Protocol	SCTP
Round-Trip Time Variation	RTTVAR
Smoothed RTT	SRTT
Transmission Control Block	TCB
Type-Length-Value coding format	TLV
Transmission Sequence Number	TSN
Upper-Layer Protocol	ULP
כאשר מדברים על משתמש, מדברים על התוכנה (שכבה שבע) - User Application .	User
הודעה מצד המשתמש המוגשת ל-SCTP על ידי ה-ULP.	Message
איגוד מספר פעולות שונות תחת אותה קורת גג.	Multiplexing
פעולה אופציונלית של Multiplexing אשר יכולה להכיל יותר מהודעת המשתמש אחת באותה פקטת SCTP.	Bundling
משתנה האחראי על הגבלת גודל המידע בבתים שניתן לשלוח ליעד מסוים לפני שמקבלים אישור.	Congestion
"חתיכה", יחידה של מידע בתוך פקטת SCTP אשר מורכבת מ"ראש" המתאר את החתיכה והתוכן של החתיכה (Payload).	Chunk
מספר הסופי לחתיכה האחרונה, והגיע עליה אישור.	Cumulative TSN Ack Point



יעד אשר מכיל שגיאות, או אינו זמין לקבל הודעות.	<b>Inactive destination</b>
השימוש ב-Big Endian - שהבתים החשובים (MSB) נמצאים בהתחלה.	<b>Network Byte Order</b>
הודעת משתמש אשר נשלחה לפי הסדר, עם התחשבות לכל הודעות העבר של המשתמש שנשלחו.	<b>Ordered Message</b>
מספר TSN אשר שייך לחתיכה אשר נשלחה ליעד, אך טרם התקבל עליה אישור.	<b>Outstanding TSN</b>
הנתיב אשר נלקח על ידי פקטת SCTP אשר נשלחה ליעד מסוים לפי כתובת. כאשר, שליחה ליעד אחר, אינה מבטיחה נתיב שונה.	<b>Path</b>
נתיב MTU, היכולת להבטיח נתיב לפי גודל ה-MTU.	<b>PMTU</b>
Receiver Window - משתנה של SCTP המחזיק בתוכו את החישוב העדכני ביותר לחלון השליחה של היעד, ונשמר בבתים. זה מאפשר לשולח הבנה של הגודל הקיים אצל היעד המקבל את הבקשה.	<b>rwnd</b>
הנתיב המרכזי וברירת המחדל ליעד ולמקור ההודעות. ההגדרה מחזיקה בתוכה את כתובת המקור, ולא רק את כתובת היעד, היות והמימוש ירצה לרוב לדעת כיצד לחזור למקור על ידי שליטה בנתיב התעבורה עצמו.	<b>Primary Path</b>

ישנם עוד מונחים, אשר יכנסו בחלק הבא של המאמר.

## מהו SCTP?

הפרוטוקול של SCTP הוגדר על ידי IETF בשנת 2000 על מנת לאפשר העברת נתוני Signaling System 7 או SS7 בקיצור. SS7 משתמש בחיבור קווי, ובעל תשתית עצמאית משלו, ואינו קשור לרשת TCP/IP. הפרוטוקול החדש עבור SS7 של IETF קיבל את השם SIGTRAN, ובשביל להצליח ולספק לו מענה על שלל המרכיבים שלו, בעצם יצרו את SCTP.

השימושים ב-SS7 הם עבור עולם הטלפוניה. כאשר אחד השימושים העיקריים שלו, הוא שליחה וקבלה של SMS, ושימוש נפוץ נוסף הוא התמודדות עם טלפוניה בין מרכזיות אשר נמצאות בסביבות שונות, תחת אותה תשתית.

כאשר צריכים לממש את SS7 על גבי רשת ה-IP, ישנם הרבה סוגים שונים של בעיות אשר היו צריכים לפתור. הנה רשימה חלקית של הבעיות שהיו צריכים לפתור:

- כיצד להעביר הודעות?
- כיצד להעביר איתות?
- כיצד ניתן לעבוד עם מספר שרתים במקביל?
- כיצד ניתן להבטיח איכות תעבורה?
- כיצד ניתן לדחוס כמה שיותר מידע בבקשה אחת?
- כיצד ניתן להבטיח כי חלק מהמידע ידווח על שהגיע ליעד?
- כיצד להתעלם ממידע אשר אין צורך לדעת האם הוא הגיע?

על מנת לפתור חלק גדול מבעיות אלו, ניתן היה לשלב תעבורה שחלקה הוא TCP וחלקה הוא UDP, אך זוהי גישה מאוד בזבזנית, אשר אינה מאפשרת לדחוס כמה שיותר מידע בבקשה אחת (multiplexing). זו הסיבה שבעצם נוצר SCTP.

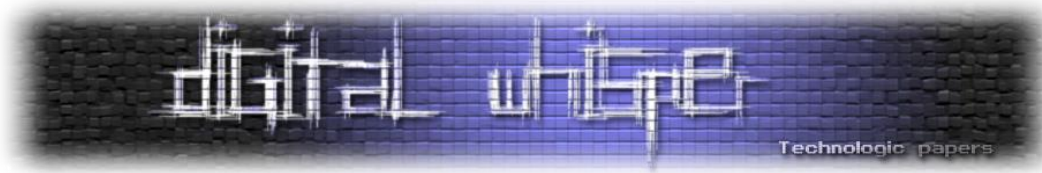
הבדל נוסף גדול ומשמעותי של SCTP אל מול TCP הוא בכך ש-TCP עובד על בסיס stream של מידע, בעוד ש-SCTP מבוסס הודעות (message).

### מה ההבדל בין Stream לבין Message?

תעבורה על בסיס הודעות מדברת על כך שהודעות זה משהו שיודע להתחלק במידע לחלקים קטנים, או קבוצות של מידע ובכך מי שמקבל את המידע יודע מתי להתחלה, אמצע והסוף של ההודעה. כאשר מדובר ב-stream, יש צורך לשלוח רצף של מידע, ובכך הרצף עצמו קובע את ההתחלה, אמצע וסוף של המידע.

למרות שב-TCP ניתן לדעת לפי כל פקטה מה הסדר, אי אפשר לדעת בפרוטוקול את הסדר של המידע, אלא רק את סדר השליחה שלו. כלומר אין מידע לגבי סדר התוכן, אלא רק לגבי מצב השליחה עצמו, בעוד שעל בסיס הודעות, כדוגמת SCTP, ניתן לדעת עוד ברמת הפרוטוקול את סדר המידע עצמו.

הפרוטוקול של SCTP יודע להכיל בתוכו מידע של מדיה למשל, בגישה של real-time, כאשר במידה והמידע לא הגיע, ממשיכים הלאה במקום לנסות ולקבל אותו בכל מחיר. אך בנוסף לכך, הפרוטוקול יודע להעביר הודעות אשר חשוב מאוד שיגיעו כמו שצריך, וחשוב לדעת להתעכב עליהם עד אשר יגיעו כמו שצריך.



אחרי התכנון, במימוש של SCTP ישנן התכונות הבאות:

- תמיכה ב-multihoming.
- יכולת עבודה ב"חתיכות" (chunks) בערוצים שונים, ללא צורך בתמיכת HOL (Head of Line).
- שליטה כיצד להתנהג עם מידע:
  - האם יהיה עליו פידבק.
  - האם לנסות ולשלוח אותו הלאה.
  - האם להתעלם מהמידע.
- הבטחת איכות תעבורה על ידי שימוש באלגוריתמים של Path Selection.
- יכולת להתמודד עם Jumbo Frames, כאשר מדובר ב-MTU גדול במיוחד (מעל 1,500 בתים).
- כלים ומימושים לסיוע באבטחת המידע ומניעת התקפות שונות, כדוגמת Replay attack.

### MultiHoming

כאשר מדברים על MultiHoming, מדברים על כך שמחפשים שרידות של תעבורה, על ידי שימוש ביותר משרת אחד בשביל להעביר את המידע לנקודה אחת, במידה ואחד מהשרתים אינו מבצע את הפעולה שלו בצורה טובה, או אינו נגיש.

שימוש נוסף ב-multihoming הוא שימוש בנתיבי רשת באיכות שונה על מנת להבטיח תעבורה טובה. למשל כמות קפיצות (hop) בין שרתים עד אשר המידע מגיע ליעד, מהירות התעבורה בין כל יעד וכיו"ב...

בגדול מאוד, הרעיון של MultiHoming אומר כי המקור ו/או היעד מכילים יותר משרת אחד, ובמידה והשרת הראשי או הראשון אינם מגיבים, המידע יתקבל לשרת השני. כמובן שניתן שיהיו יותר משני שרתים. ובנוסף, ישנם מצבים בהם אחד מהצדדים אינו תומך בגישה של MultiHoming ועדיין הצד שיודע לתמוך, ידע להתמודד עם המצב.

כאשר SCTP מדבר על MultiHoming, מדובר למעשה במצב בו מספקים מספר יעדים. כאשר תעבורת השרת המוציא מקבל את את הבקשה ורואה את היעדים השונים, חלקה העליון של הבקשה (ULP) כיעד הראשי להודעה. בברירת המחדל בנושא, תעבורת השרת תמיד תבחר יעד ראשי אחד, אלא אם מוגדר ברמת ה-SCTP יעד ברור.

השרת אשר מוציא את הבקשה תמיד צריך לדעת להעביר את הנתונים לשרת הנבחר, דבר הכולל גם הודעות תעבורה כדוגמת ACK-ים שונים, עליהם ידובר בהמשך. עניין זה כולל גם את היעד, אשר צריך לענות חזרה למקור, כאשר גם הוא יכול במקור להיות בעל תמיכה ב-multihoming.



כאשר יש מספר הודעות ממקורות שונים לבקשה מחולקת, הצד המקבל, צריך לענות לשולח, ולא לשרת הראשי כתגובה להודעות עצמן.

כאשר יש ניסיון שליחה ליעד אשר אינו פעיל, בין אם בשל שגיאות, ובין אם באמצע התקשורת, יהיה צורך להחליף את השרת אשר אינו באוויר כרגע, ותהיה החלפה של השרת על ידי ניסיון לשלוח ליעד חלופי, ואם זה נכשל, יהיה דיווח על השגיאה הלאה.

דיווח השגיאה יתבצע רק כאשר יגיע timeout או חוסר יכולת למצוא יעד פעיל.

## חתיכות

ב-SCTP פקטה מורכבת מ"ראש" או הכרזה מוכרת וידועה (common header) ומ"חתיכת" מידע (chunk).

הפרוטוקול יודע להכיל בתוכו מספר חלקים שכאלו, אשר מורכבים מהכרזה מוכרת וידועה ו"חתיכת מידע". הגבול לכמות החתיכות, הוא גודל ה-MTU.

חלק מחתיכות המידע חייבות להגיע שלמות, וחלק לא, בהתאם להרבה מאוד הגדרות בפרוטוקול. המידע אשר חייב להגיע בצורה שלמה, הוא למעשה הודעות שמורות של הפרוטוקול, למשל הודעה על התחלת השליחה (בסגנון לחיצת שלוש הידיים של TCP) בשם INIT, מנגנון של "עוגיות" (אדבר על כך בחלק אבטחת המידע), שליחת heartbeat, שנוי ה-window לטיפול ב-congestion ועוד. המידע הזה חייב להגיע בשלמותו בפעם אחת, ולכן אינו יהיה בחלקים.

גם מידע המגיע בחלקים, יכול להיות מוגדר כ"שלם". זה יקרה כאשר שולחים מידע על פעולות של הבקשה עצמה, כדוגמת פעולת ה-INIT למשל, והתשובה שלה שתהיה INIT ACK.

סוגי הודעות/חתיכות:

קוד	סוג הודעה	הסבר
0	DATA	המידע עצמו (payload)
1	INIT	התחלת שליחה
2	INIT ACK	אישור קבלת השליחה
3	SACK	אישור חלקי בדבר קבלת השליחה
4	HEARTBEAT	בקשת HEARTBEAT
5	HEARTBEAT ACK	אישור על בקשת ה-HEARTBEAT
6	ABORT	בטל פעולות
7	SHUTDOWN	בקשה לסיום מסודר של תשדורת
8	SHUTDOWN ACK	אישור על קבלת SHUTDOWN
9	ERROR	שגיאת פעילות (operation error) - אינה נחשבת למשהו שלא ניתן להמשיך דרכו, בניגוד ל-ABORT, אך יכולה לבוא בשימוש גם של ABORT לשם כך.
10	COOKIE ECHO	התחלה של עוגיות (ארחיב על הנושא בחלק אבטחת המידע)
11	COOKIE ACK	אישור קבלת עוגייה
12	ECNE	שמור עבור הכרזת congestion
13	CWR	שמור עבור הכרזת הפחתת חלון שליחה
14	SHUTDOWN COMPLETE	סיום פעולת הכיבוי
15-62	פנוי	
63	שמור	
64-126	פנוי	
127	שמור	
128-190	פנוי	
191	שמור	
192-254	פנוי	
255	שמור	



כאשר מדובר בהודעות מערכת INIT ACK, INIT COMPLETE ו-SHUTDOWN, החתיכות חייבות להגיע לבד, ואסור להן להגיע עם עוד חתיכות בנוסף.

כאשר המידע (Payload) אינו יכול להיכנס לפקטה בודדת, הפקטה יכולה ליצור מספר חלקים של אותו המידע ובכך ליצור מידע מחולק (fragmented).

ב-SCTP שולחים כאמור את מידע המשתמש בחתיכות, וניתן להגדיר התנהגות של חתיכה מסוימת. למשל ניתן להגיד כי יש להתעלם ממנה, או לשלוח עליה פידבק, ובכך בתוך ה-multiplexing, ניתן להגדיר התנהגות של כל חתיכה. בנוסף, המידע על החתיכה מכיל מספר סידורי עולה (TSN), מה המיקום שלה - למשל התחלה, סוף או בגלל שאין סימון שכזה, יודעים לפי מספר סידורי מה המיקום שלה באמצע.

### Path Selection

הבטחת מסלול התעבורה, בניגוד ל-MPLS או DiffSrv, אשר מתבצעים ברמת ה-IP ואף ברמת החיבור, כאן הבטחת המסלול מתבצעת בשכבה גבוהה יותר. המימוש של SCTP מבצע Path MTU Discovery אשר מוגדר ב-rfc 1191 ו-rfc 1981. הרעיון הוא ללכת רק בכיוון שבו ה-MTU הוא בגודל מסוים, ואינו חורג מהגודל הזה.

ההבדלים של SCTP מ-rfc 1191 הם:

1. SCTP תומך ביכולת להחזיק מספר כתובות IP כיעד (multihoming כאמור), וצריך לשמר ניתוב עבור כל כתובת IP בנפרד.
2. השולח צריך לבצע מעקב אחר PMTU המשויך לבקשה עם ה-PMTU הנמוך ביותר עבור כל החיבורים אל היעד. כאשר מחלקים את התעבורה לחלקים שונים (fragmentation), השיוך הזה של ה-PMTU יהיה בשימוש על מנת לחשב כל חלק. גישה זו, תאפשר את היכולת לבצע שליחה מחדש ליעד אחר ללא צורך בביצוע חילוק על בסיס כתובת IP.

### עוגיות ואבטחת מידע

לפרוטוקול SCTP ישנם מספר מנגנונים המאפשרים לו לדעת האם המידע שהגיע אמין, האם להתייחס אליו בכלל והאם בכלל היה צריך להגיע אותו מידע. דבר ראשון, עבור אמינות המידע, לכל חתיכה יש checksum אשר פעם היה מבוצע על ידי Adler32, אך קיבל שכתוב וב-rfc4960, אשר נכון לכתיבת מאמר זה הוא מימוש הפרוטוקול העדכני, החישוב נעשה על ידי CRC32c.

כאשר יש צורך לעבוד עם תקשורת מוצפנת, RFC 3436 מדבר על המימוש של TLS עם SCTP, ובנוסף יש תמיכה ב-IPsec עבור הפרוטוקול ב-RFC 3554. בנוסף ישנו hash קריפטוגרפי (MAC) אשר יוצר לרוב

הכרת - חלק ראשון

[www.DigitalWhisper.co.il](http://www.DigitalWhisper.co.il)

מפתח סימטרי באמצעות יצירת מספר פסודו רנדומלי, כפי שמתואר ב-RFC4086, ומימוש מאוד דומה ל-HMAC המתואר ב-RFC2104. השימוש ב-MAC, מתבצע בשביל למנוע מתקפות DoS על הבקשה, וההגנה נעשת על ידי שימוש ב"עוגיות".

עוגיות ב-SCTP הן עולם שלם, אשר נלקח במקור מ-RFC2522 - המוכר גם בשם Photuris Session-Key Management Protocol, אשר מאפשר לנהל session בצורה מאובטחת ברמת תעבורה ורמת ה-IP.

### מהי עוגיית Photuris Session-Key?

בקצרה (יחסית) - עוגייה יוצרת session קצר מועד מבוסס מפתחות בין שני צדדים, ללא החלפת המפתחות בניהם ברשת האינטרנט. המפתחות האלו מחליפים בצורה ישירה את הצורך בסמאות למשל, אשר לרוב מוגדרות בשביל לנהל תעבורה מאובטחת. הפרוטוקול משתמש באותם מפתחות פרטיים (shared secret) שהוגדו בעבר לצורך זיהוי הצדדים. העוגייה יוצרת סוג של הגנה מפני הצפת בקשות מכתובות IP אשר אינן מוכרות, או פורטים שנפתחים ב-UDP, על ידי כל שכל צד מעביר "עוגייה" לצד השני.

לאחר שליחת העוגייה, רשימה של מוסכמות על הדיאלוג מוחזרת לשולח, על מנת לחשב את המפתח המשותף. ערך כלשהו מוחלט על ידי הצדדים, ונוצר מפתח פרטי לתקשורת בין הצדדים. כל צד מעביר לצד השני ערך מספרי כלשהו. הערכים האלו הם בשימוש על מנת לחשב את המפתח המשותף בניהם. הצד המקבל של הערך, נשאר stateless עד אשר המפתח המשותף נוצר.

בנוסף, ישנם עוד תכונות משותפות העוברות בין הצדדים, על מנת לסייע ביצירת שיח בעל תכונות מאובטחות. לאחר החלפת המידע, מערכת החלפת המידע מזהה את הצדדים ומוודאת את אמינות המידע אשר נשלחו בפעימה הראשונה והשנייה.

בנוסף, המפתח הפרטי מאפשר ליצור מפתחות נפרדים עבור ה-session לכל כיוון, אשר אחראים על הזדהות "רגילה" ומקובלת גם ללא השימוש בעוגיות.

### עוגיית STCP

על מנת להתחיל ולשלוח מידע, SCTP חייב ליצור פעולה בשם INIT, אשר מודיעה על התחלת פעולה. רק כאשר פעולה זו מסתיימת, נוצרת הכרה בין הצדדים. צד המשתמש חייב בכל קצה להשתמש ב-Associate Primitive על מנת לאתחל את השיוך לקצה השני של SCTP. כאשר השיוך מסתיים, נפתח חיבור זרימה unidirectional להעברת המידע של כל הצדדים. פעולות אלו דורשות הגנה, היות והן רגישות מאוד. ההגנה מתבצעת על ידי שימוש בעוגיות.

תהליך האתחול מכיל בתוכו את הצעדים הבאים (נקודה A מנסה להגיע לנקודה Z ונקודה Z צריכה לאשר את הבקשה):

1. נקודה A שולחת חתיכה של INIT אל עבר נקודה Z. בתוך ה-INIT, A חייב לספק טאג הזדהות (verification tag), שנקרא TAG\_A לצורך ההדגמה, בתוך המיקום המתאים לטאגים (עליהם אדבר בחלק הבא). הטאג חייב להיות מספר רנדומאלי בין 1 ל-4294967295. כאשר השליחה של ה-INIT התבצעה, נקודה A מתחילה טיימר לטאג, ויוצרת מצב של COOKIE-WAIT.
2. נקודה Z צריכה להגיב מיד עם פעולה מסוג INIT ACK בתוך חתיכה הנשלחת חזרה. כתובת החזרה של INIT ACK חייבת להיות כתובת המקור בפעולת השליחה שאליה ה-INIT ACK מגיב. בתגובה, בנוסף למילוי הפרמטרים השונים, Z חייב להכיל את Tag\_A, אך גם ליצור Tag\_Z משל עצמו בנקודת האיתחול. בנוסף, Z חייב ליצור ולשלוח ביחד עם ה-INIT ACK עוד עוגייה. כאשר Z שולח את ה-INIT ACK עם כל הפרמטרים הנדרשים, אסור ל-Z ליצור משאבים כלשהם או להחזיק איזשהו מצב עבור השייך שבוצע, אחרת Z פגיע להתקפות.
3. כאשר A מקבל את ה-INIT ACK מ-Z, הנקודה חייבת לעצור את הטיימר T1-init ויצא ממצב של COOKIE-WAIT. לאחר מכן, A ישלח את מצב העוגייה שהוא קיבל ב-INIT ACK בפעולה של COOKIE ECHO, ויפעיל טיימר חדש ואז יכנס למצב של COOKIE-ECHOED. החתיכה של COOKIE-ECHO יכולה להיות משולבת עם עוד חתיכות נוספות של מידע, אבל היא חייבת להיות החתיכה הראשונה בסדרה, עד אשר יתקבל COOKIE ACK על ידי השולח, כאשר מידע נוסף לא ישלח עד אז על ידי היעד.
4. בקבלת המידע של COOKIE-ECHO, צד Z יענה עם COOKIE ACK לאחר בנייה של TCB ולעבור למצב של ESTABLISHED. COOKIE ACK יכולה להגיע כחלק מחתיכות המחכות לשליחה ו/או חתיכה מסוג SACK, אבל COOKIE ACK חייבת להיות החתיכה הראשונה בפקטה. המימוש יכול לבחור אם הוא יבצע שליחה בעת קבלת ההודעה של COOKIE ECHO תקני או מאוחר יותר.
5. בעת קבלת COOKIE ACK, צד A יעביר את המצב מ-COOKIE-ECHOED למצב של ESTABLISHED, צד A גם יעצור את הטיימר T1-Cookie. צד A יבחר גם אם לעדכן את צד ה-ULP על הצלחת החיבור, באמצעות הודעת Up.

חשוב להדגיש כי החתיכות של INIT ו-INIT ACK אינן מורשות להגיע ביחד עם עוד חתיכות וחייבות להיות בפקטה משל עצמן.



תרשים הבא מספק הצגה של פעולת ה-INIT:

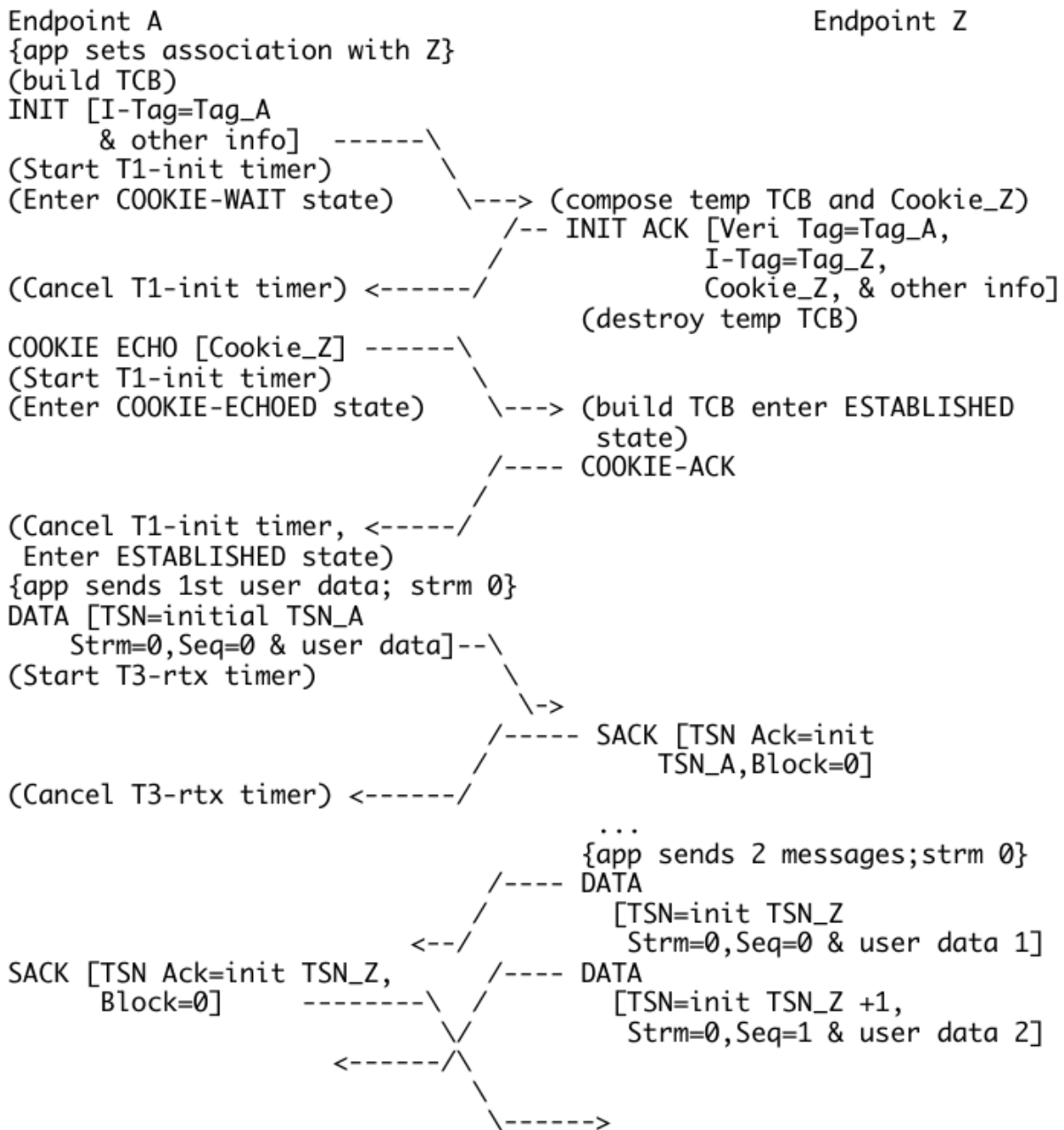
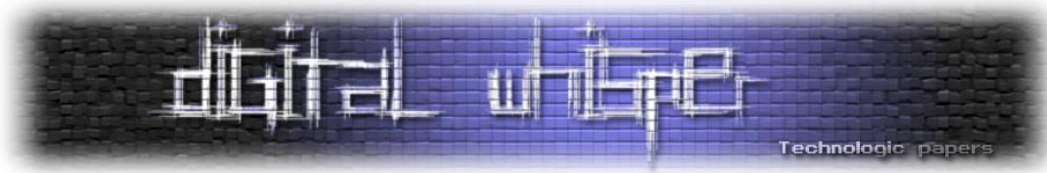


Figure 4: INITIATION Example

[נלקח מ-rfc4960]



## סיכום

פרוטוקול SCTP הוא אחד הפרוטוקולים המעניינים ביותר לתעבורה הקיימים כיום לדעתי, ומאפשר לבצע המון פעולות שונות ומשונות. בחלק זה, נגעתי כאן רק בקצה המזלג ברעיון, אשר מספק הבנה לגבי יכולות הפרוטוקול, אך פחות על מימושו והתמודדות עם נושאים שונים.

בזכות כך שהפרוטוקול הוא יחסית פרוטוקול חדש, בזמן תכנונו נלמד הרבה מאוד מבעיות שונות הקיימות ברשת האינטרנט, והתכנון מנסה לספק לבעיות אלו מענה באמצעות כלים שונים. למשל לספק יכולת של Path MTU Discovery, High Availability ו-DR באמצעות Multihoming, היכולת לקבוע נתיב תעבורה באמצעות Path MTU Discovery ועד לפתרון עבור בעיות אבטחה כדוגמת DoS ו- Replay Attack באמצעות עוגיות.

אך החלק החשוב ביותר הוא אספקת היכולת לבצע Multiplexing למידע, ובכך לשלוח כמה שיותר מידע של פעולות שונות בין שרתים שונים אשר מצפים להודעה. יכולת זו, יכולה להגיע כאמצעי זרימה, וכאמצעי של הודעה.

בחלק הבא אדבר יותר על המימוש, אך גם הוא יהיה קצר מאוד, ולא יחליף לימוד מקיף יותר של הפרוטוקול.

## ביבליוגרפיה

- [Stream Control Transmission Protocol \(RFC4960\)](#)
- [Path MTU Discovery - IPv4 \(RFC1191\)](#)
- [Path MTU Discovery for IP version 6 \(RFC1981\)](#)
- [HMAC: Keyed-Hashing for Message Authentication \(RFC2104\)](#)
- [Transport Layer Security over Stream Control Transmission Protocol \(RFC3436\)](#)
- [On the Use of Stream Control Transmission Protocol \(SCTP\) with IPsec \(RFC3554\)](#)
- [Security Attacks Found Against the Stream Control Transmission Protocol \(SCTP\) Current Countermeasures \(RFC5062\)](#)
- [Randomness Requirements for Security \(RFC4086\)](#)
- [Photuris: Session-Key Management Protocol \(RFC2522\)](#)
- [Stream Control Transmission Protocol](#)



---

## דברי סיכום

---

בזאת אנחנו סוגרים את הגליון ה-66 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](mailto:editor@digitalwhisper.co.il).

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

[www.DigitalWhisper.co.il](http://www.DigitalWhisper.co.il)

*"Talkin' bout a revolution sounds like a whisper"*

הגליון הבא ייצא ביום האחרון של חודש נובמבר 2015.

אפיק קסטיאל,

ניר אדר,

31.10.2015