

Digital Whisper

גליון 65, אוקטובר 2015

מערכת המגזין:

מייסדים:	אפיק קסטיאל, ניר אדר
מוביל הפרויקט:	אפיק קסטיאל
עורכים:	אפיק קסטיאל
כתבים:	ליאור אופנהיים, יניב בלמס, דימה פשול, רזיאל בקר 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 ו/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

דבר העורכים

ברוכים הבאים לדברי הפתיחה של הגיליון ה-65 של Digital Whisper!

כשמתעסקים באבטחת מידע הופכים להיות לאט לאט פרנואידים, אם משתמש "רגיל" סתם מתעצבן כאשר הוא חוטף Blue Screen, אנחנו בדרך כלל נהיה די סקרנים לגבי הסיבה, וננסה לגלגל אחורה בראש איזה דרייבר התקנו לאחרונה, ואולי אפילו נחפש ונחקור את ה-MiniDump על מנת להבין לעומק את הסיבה (ואז גם נגיד לעצמנו ש-"אחרי ה-Reboot הקרוב נשנה את הקונפיגורציה כך שסוף סוף נתחיל לקבל FullDump..."), אם משתמש רגיל ירגיש איטיות בעת הגלישה ברשת הבייתית שלו, אנחנו נתחבר לנתב שלנו ונבדוק את הסטטוס של ה-DHCP וננסה לברר האם שכן סורר הצליח לנחש את הסיסמה הסופר מורכבת שלנו...

תמיד אפשר לבוא אלינו בטענות, להגיד שאנחנו סתם חיים בסרט, שנשמור את הקונספירציות שלנו לעצמנו או לטעון שאנחנו פרנואידים... (ואנחנו כמובן נצטט את הלר ג'וזף, ונטען ש:"רק בגלל שאנחנו פרנואידים לא אומר שלא רודפים אחרינו...") ☺.

אבל עם כל הכבוד, האירועים שהיו החודש די לא עוזרים להרפות מהפרנויה: זאת כבר הפעם השלישית שענקית המחשבים לנובו נתפסת מנסה [להשחיל לנו כל מיני כלי ריגול על המחשבים שלהם](#) ובעזרתם להפוך גם אותנו להיות חלק מהצבא הסיני... (ומי יודע כמה פעמים היא ניסתה ולא נתפסה...). ולא די בזאת, מסתבר לאחרונה, שלא מעט אפליקציות ומשחקים שהועלו ל-Apple Store בזמן האחרון [כללו קוד זדוני](#), שגם הוא, איכשהו קשור לסינים...

אז פרנויה או לא פרנויה - החלטה שלכם, בכל מקרה הייתי פוקח עין אחת לפחות על ה-Netstat...

ובנוסף, החודש החלטנו לתת נסיון נוסף לפינה חדשה-ישנה: פינת החדשות. והפעם בניצוחו של **ים מסיקה**, שטרח, אסף, סיכם וערך לנו מספר אירועים מעניינים שפקדו אותנו החודש - **תודה רבה!** בתקווה שנצליח להחזיק איתה יותר מ-3 גליונות רצוף... ☺.

וכמובן, לפני שניגש לחלק האומנותי, ברצוננו להודות לכל מי שהשקיע ונתן מזמנו האישי ובזכותו הגיליון פורסם, תודה רבה **לליאור אופנהיים**, תודה רבה **ליניב בלמס**, תודה רבה **לדימה פשול**, תודה רבה **לרזיאל בקר!**

קריאה מהנה!

ניר אדר ואפיק קסטיאל.



תוכן עניינים

2	דבר העורכים
3	תוכן עניינים
4	אז מה קרה החודש?
9	How to turn your KVM into a tagging Key-Logging monster חלק ב' - בשורות רעות
14	אתגרי ה-CrackMe של רפאל 2015
53	הסודות החבואים ב-WebSocket
69	דברי סיכום

אז מה קרה החודש?

מאת ים מסיקה

מזריקים עוגיות (מחיבור לא מאובטח לחיבור מאובטח)

החודש היה רעש די מוגזם סביב מחקר די חביב. חבורת אנשים נחמדים חקרו את ההשלכות של Cookie Injection, וכיצד אפשר לנצל Cookie Injection בכמה שיטות דרכים מגניבות. ההייפ, בקצרה, היה על כך שהם מצאו שניתן לגרום לעוגיות שהוזרקו בתקשורת מעל HTTP להישלח לשרת אחר-כך גם כשמתבצעת תקשורת מול השרת מעל HTTPS.

[המחקר עצמו](#) שווה רפרוף, והוא פורסם עוד חודש שעבר במסגרת כנס USENIX השנתי בווישינגטון. הוא הובא לאור הזרקורים ב-24 בספטמבר, כאשר ה-CERT האמריקאי [חשף](#) שכל הדפדפנים לא בדיוק התמודדו כמו שצריך עם הבעיה, ושניתן לנצל אותה בכל הדפדפנים המודרניים עקב מימוש מעט צולע הקשור בצורה שבה דברים מוגדרים ב-RFC 6265 ("עוגיות בדפדפן").

ה-RFC המדובר [מציין](#) בסעיף 8.6 (תחת הפרק "שיקולי אבטחה"):

"עוגיות לא מספקות אמינות בין דומיינים. לדוגמה, חשבו על `foo.example.com` ועל `bar.example.com`. השרת ב-`foo.example.com` יכול להגדיר עוגייה ששייכת לדומיין `example.com` (ייתכן שתוך כדי דריסת עוגייה שקיימת עבור `example.com`) שנערכה על ידי `bar.example.com`, והדפדפן יכלול את העוגייה בבקשות ה-HTTP שישלחו ל-`bar.example.com` במקרה הגרוע ביותר, `bar.example.com` לא יוכל להבדיל בין העוגייה הזו לבין עוגייה שהוא הגדיר בעצמו. השרת ב-`foo.example.com` יוכל למנף את היכולת הזו על מנת ליצור התקפה כנגד `bar.example.com`".

הוא גם [מציין](#) בסעיף 8.5 (תחת אותו הפרק):

עוגיות אינן מופרדות על ידי פורט. אם עוגייה קריאה על ידי שירות המסופק בפורט מסוים, העוגייה גם קריאה על ידי שירות בפורט אחר באותו שרת. אם ניתן לכתוב עוגייה על ידי שירות המסופק בפורט מסוים, העוגייה יכולה להיכתב גם על ידי שירות שרץ בפורט אחר על אותו שרת.

...

עוגיות לא מופרדות על ידי סוגי שירות¹. למרות שלרוב ניתן להשתמש בהן בעזרת `http` ו-`https`, העוגיות השייכות לכתובת מסוימת עלולות להיות זמינות גם בסוגי שירות אחרים, כמו `ftp` ו-`gopher`.

¹ מתורגם מהמילה `schemes`, והכוונה כאן היא ל-`HTTP`, `HTTPS`, `FTP` וכדומה.



במחקר שפורסם בכנס ה-USENIX כתבו החוקרים ג'יאן ג'יאנג, ניקולס ויבר ועמיתיהם:

ניתן להגדיר לעוגייה דגל "secure", שמסמן שהעוגייה תשלח רק מעל חיבור HTTPS. למרות זאת, אין דגל מתאים לסמן כיצד העוגייה הוגדרה: תוקף שעושה Man in the middle אפילו לזמן קצר תחת HTTP, מסוגל להזריק עוגיות שיוצמדו לחיבורי HTTPS שיבואו אחר-כך.

מהמחקר עלה שאכן, מכיוון שה-RFC לא מציין שום צורה להבחין בין עוגיות שהוגדרו בצורות שונות, רוב מימושי הדפדפנים לא תמיד מאמתים את הדומיין שהגדיר את העוגיות. אז אם אנחנו נמצאים ברשת מסוימת ועשו עלינו מתקפת Man in the middle, התוקף יוכל להכניס לנו עוגיות המיועדות לאתר כלשהו, ולסמן את דגל ה-safe עבור אותה עוגייה. בכניסה יותר מאוחרת לאתר מעל HTTPS, הדפדפן ישלח לאתר את העוגיות שהושתלו בתקשורת לא מוצפנת.

ייתכן שפתרון מקיף יופיע בגרסאות הבאות של RFC 6265 (או RFC 6454, ה-RFC שמגדיר את העקרונות מאחורי Same Origin Policy). עד אז ההמלצות של CERT הן לעדכן את הדפדפנים שלכם (כל הדפדפנים תיקנו את הבעיה עד ה-16 בספטמבר), ומבעלי אתרים להפעיל [HSTS](#)² על ה-Top Level Domain שלהם.

התפוח הרקוב

שוק האפליקציות למערכות הטלפונים החכמים רק הולך וצומח, וישנם כמובן מי שימצאו בכך סיבה למסיבה. הפעם אני לא מדבר על חברות התוכנה שגורפות סכומים נאים, אלא דווקא על Maleware חדש-דנדש ש"הושחל" לתוכנות רבות בחנות האפליקציות של ענקית התוכנה Apple.

אחת מסביבות העבודה המוכרות ביותר שמשמשות מתכנתים לפיתוח אפליקציות ל-iOS (מערכת ההפעלה לטלפונים חכמים של Apple) היא Xcode. Xcode היא IDE שמפותחת על-ידי Apple, ומשומשת על ידי רוב המתכנתים בשוק זה. היא חנימית, ומאוד נפוצה בשוק - אבל שוקלת כ-3GB.

גרסת חיקוי של Xcode, היא זו שאחראית לבלגאן. הקוד הזדוני הוכנס לקובץ Mach-O ש"נארז מחדש" לתוך גרסאות מסוימות של Xcode. הגרסאות הארוזות מחדש הועלו לענן של Baidu, מנוע החיפוש הסיני. מכיוון שהשרתים של Baidu נמצאים בסין וההורדה מהם תהיה מהירה בהרבה מהשרתים של Apple, העדיפו מתכנתים סינים רבים להוריד את התוכנה מהשרתים הללו - בלי לדעת שהגרסה שאותה הם מורידים נגועה למעשה בקוד הזדוני.

בחזרה לעולם פיתוח התוכנה של Apple: CoreServices הוא ממשק שמסופק על ידי Apple וכולל הרבה שירותי מערכת בסיסיים שמפתחים יכולים להשתמש בהם. כאשר אפליקציית iOS עוברת הליך הידור, Xcode מחפשת את CodeService בנתיבים מוגדרים מראש ומוסיפה אותו לקוד של המפתחים.

² HSTS מאפשר לשרת להגיד לדפדפנים לתקשר עם האתר אך ורק דרך HTTPS, ולעולם לא דרך HTTP.

אז מה קרה החודש?

www.DigitalWhisper.co.il



[לפי פאלו-אלטו](#), XcodeGhost מממשת קוד זדוני בקובץ CodeServices משלה, ומעתיקה את הקובץ הזה לנתיב שבו Xcode תחפש את CodeServices. כך מתווסף קוד זדוני לכל אפליקציה שתהודר בעזרת Xcode, ללא ידיעת המפתח. בקובץ הזדוני ישנו קוד זדוני במחלקות של UIWindow ושל UIDevice, שבהן כמעט כל אפליקציה משתמשת.

התוצאה? אפליקציות רבות ל-iOS שבהן מושגל קוד זדוני שנראה כמו רכיב לגיטימי של Apple. למרות ש-Applle ידועה בבדיקות המחמירות שלה לתוכנות לפני שהן עולות לחנות האפליקציות שלה, הפעם, כנראה, משהו השתבש. מאות אפליקציות נגועות עלו לחנות האפליקציות, כולל אפליקציות גדולות ומוכרות, ביניהן WeChat ו-Angry Birds.

הקוד הזדוני אוסף מידע בעזרת הפונקציה AppleIncReserved, הכולל את הזמן, שם האפליקציה, השם וסוג המכשיר, מדינה ושפה, סוג הרשת אליה המכשיר מחובר ו-UUID של המכשיר. השלב הבא הוא הצפנת המידע, ושליחתו לשרתי השליטה והפיקוד מעל HTTP.

נפילה קצת פחות קשה החודש התרחשה עם Siri, העוזרת הקולית של מכשירי ה-iOS. במידה והמשתמש הפעיל את האפשרות להשתמש ב-Siri במסך הבית, היה ניתן להשתמש בה על מנת לעקוף את נעילת המכשיר, ולהגיע לתמונות, אנשי קשר ושליחת הודעות.

שם קוד: משטרת הקארמה

ארגון הביון הבריטי GCHQ נתפס עם המכנסיים למטה: [במסמכים סודיים שהודלפו](#) ונכתבו ב-2008, נמצא בין הסעיפים הביטוי "KARMA POLICE", שתפס תוך זמן קצר כותרות בעיתונים טכנולוגיים רבים. במסמך כתוב:

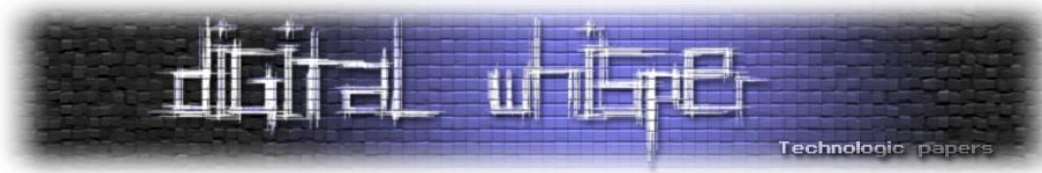
"משטרת KARMA POLICE היא לקשור כל יוזר הזמין ב-SIGINT עם כל אתר שבו הוא מבקר, וכך לספק או (א) אפיון גלישה ברשת עבור כל משתמש הנראה ברשת אינטרנט, או (ב) אפיון משתמש עבור כל אתר הנראה ברשת האינטרנט"

התוכנית מחלצת את נתוניה מהכבלים הבין-יבשתיים העוצרים בקורנוול שבדרום מערב אנגליה, בהם עוברת 25% מתעבורת האינטרנט הכלל-עולמית.

[מסמך הסיכום](#) של התוכנית מ-2009 מראה כי היא אכן יצאה לפועל. מהפעולה הושפעו, כך לפי המסמך, "224,446 כתובות IP שונות במהלך תקופה של שלושה חודשים". אם לא די בכך, לפי מסמכים נוספים הגיעה התוכנית מ-10 מיליארד רשומות בין 2007 לבין 2009, לכ-50 מיליארד רשומות בשנת 2012 - והמטרה המוצהרת הייתה להכפיל את הכמות הזו.

אז מה קרה החודש?

www.DigitalWhisper.co.il



המסמכים, שנחשפו על ידי המדליף אדוארד סנודן, מפרטים על [מערכות רבות](#) של ה-GCHQ, בהן:

- Black Hole - מסד הנתונים שבו נשמרו 217TB של מידע (לא מכווץ).
- Karma Police - יראה לכם איזה משתמש נכנס לאיזה אתר.
- Memory Hole - מספק מידע על חיפושים שנעשו במנועי חיפוש (מתי, איפה, IP).
- Mutant Broth - כלי שמאפשר להוציא מידע מה-Black Hole לפי זיהוי מסוים, כמו עוגיות.

D-Link והמפתח האבוד

ב-18 בספטמבר התעוררנו למאורע מביך מאוד - כלי תקשורת שונים הודיעו כי יצרנית ציוד הרשת הטיוואנית D-Link פרסמה את אחד מהמפתחות הפרטיים שלה, המשמש אותה לחתום על אחד ממוצריה, ברשת.

המפתח הפרטי נמצא באחת מחבילות הקוד הפתוח שהחברה הפיצה בחופשיות באינטרנט על ידי אדם ערני במיוחד, שקנה את מצלמת האבטחה DCS-5020L מבית D-Link, והוריד את הקוד של הקושחה מאתר היצרן. הקורא דיווח לאתר הטכנולוגיה ההולנדי [Tweakers](#), שדיווחו לחברת האבטחה ההולנדית Fox-It. זו אימתה את הממצאים, ומשם החלה המהומה.

במסגרת המחקר התגלו גם המפתחות הפרטיים של Starfield Technologies, של KEEBOX ושל Alpha Networks, אם כי כולם נשללו או פקעו טרם תאריך הגילוי. המפתח של D-Link היה זמין וחשוף לכל החל מה-27 בפברואר השנה, ועד ל-3 בספטמבר, בו פקע.

בזמן שלא ידוע האם נעשה שימוש במפתח הפרטי או לא, ישנה האפשרות שגורמים זדוניים עשו שימוש במפתח הפרטי כדי להקל על עצמם בהרצת תוכנות זדוניות על מכונות. זכורים לנו שימושים כאלו בעבר בהתקפות APT שונות, כמו ב-[Destover](#) שנחתמה לכאורה על ידי סוני.

בכל מקרה, על מנת לוודא שלא קיימות ומסתובבות תוכנות שנחתמו על ידי המפתח טרם ה-3 בספטמבר, [שללה מיקרוסופט](#) את ארבעת המפתחות שהתגלו מספר ימים אחרי המאורע, ב-24 בספטמבר 2015.

פעם שלישית לנובו

החברה הסינית [Lenovo](#) נתפסת החודש מתקינה Spyware על המחשבים של לקוחותיה. לקוחות רבים זועמים על החברה וזו מבוקרת באופן נרחב באמצעי התקשורת, בין היתר מכיוון שזו הפעם השלישית שהחברה נתפסת במעשה שכזה:

- פעם ראשונה, כזכור, הייתה בפרשת [Superfish](#) שבה התקינו Lenovo את ה-Spyware של חברת Superfish על המחשבים ששיווקו בין אוקטובר 2014 לדצמבר 2014. ה-Spyware אפשר לחברת Superfish להתקין את עצמם כ-CA מורשה במערכת ההפעלה של הלקוח ולהיות MiTM בכל

אז מה קרה החודש?

www.DigitalWhisper.co.il

ההתקשרויות המוצפנות מעל HTTPS. התרחיש הסתבך אף יותר כאשר ה־Private Key של Superfish נחשף, וגרם ל־Lenovo ול־Superfish הרבה כאב ראש תדמיתי.

- [בפעם השנייה](#) נתפסה Lenovo מכניסה קוד לקושחה של המחשב, כך שבכל העלאה של מערכת ההפעלה ישלח מידע על המחשב לחברה, ו"יעודכנו הדרייברים, הקושחה ותוכנות שהותקנו כברירת־מחדל", וגם "יסרקו קובצי זבל וימצאו דברים שמשפיעים על ביצועי המערכת". מכיוון שמדובר בקוד שכתוב על הקושחה של המחשב, הוא לא נחקק בהתקנה מחדש של מערכת ההפעלה, מן הסתם. [לימים](#) מצאו מומחי אבטחה פרצות אבטחה רבות בקוד, ו־Lenovo הפסיקה לדחוף אותו למחשבים שלה ואף פרסמה דרך להסיר אותו.

הפעם הענקית הסינית [מסתבכת בפעם השלישית](#), כשנמצא שהיא משתילה תוכנה נוספת בשם "תכנית המשוב ללקוחות לנובו" במחשבי Thinkpad של IBM. התוכנה, כך מדווח, שולחת מידע ל־Omniure, חברה למרקטינג ולניתוח נתוני אינטרנט. בהחלט לא Juicy כמו להתקין עוד CA או לצרוב את תוכנת הזבל שלך כקושחה, אבל אי אפשר להגיד שהמוניטין של לנובו משתפר עם השנים.

אולי עשו בצדק ארגוני הביון של ארצות הברית כשהחליטו שהם [לא משתמשים](#) יותר במחשבי Lenovo מחשש לריגול, ממש שנה לפני שגילינו את Superfish.

How to turn your KVM into a raging Key-Logging monster חלק ב' - בשורות רעות

מאת ליאור אופנהיים ויניב בלמס

הקדמה

שלום חברים וברוכים השבים לחלק השני בסדרת המאמרים שלנו, בה ננסה להפוך KVM שולחני נחמד ותמים למפלצת Key-Logging זועמת.

החלק הראשון הסתיים בכך שהצלחנו לקבל 64K של מידע בינארי בלתי קריא לחלוטין מתהליך עידכון הקושחה של ה-KVM. במהלך העידכון המידע הבינארי הזה מועבר דרך הכבל הסיריאלי ועושה את דרכו דרך סבך המעגלים האלקטרוניים עד שמגיע בסוף דרכו אל מעבד ה-Winbond 8052 כאסמבלי 8051 תיקני.

או שזה לפחות הניחוש הטוב ביותר שלנו כרגע...

הבעיה עם ניחוש כזה, איך לומר את זה בעדינות, היא די קשה... למרות שזה נשמע לנו הגיוני לגמרי, ולמרות שראינו הרבה סימנים לכך שזה באמת המצב, עדיין יש סיכוי לא רע שאנו טועים.

הבעיה כאן היא שאם אנחנו רוצים לנסות ולפענח את שיטת הקידוד של המידע הבינארי, חשוב מאוד להבין איך צריכה להיראות התוצאה הסופית.

במידה וההנחה שלנו נכונה, התוצאה הסופית תהיה אסמבלי 8051, אבל אם אנחנו טועים זה יכול להפוך את תהליך הפיענוח לסיוט מתמשך. אפשר אולי להשוות את זה לגרסה של משחק פוקר מבולבל, מוזר, בחדר חשוק לחלוטין עם קלפים ריקים על סכומים אינסופיים, עם מחלק שלא אומר לך את החוקים ומחייך כל הזמן³.

אנחנו חייבים למצוא איזשהי דרך לוודא שהמידע אכן מגיע אל הציפ שלנו. אבל איך?

³ מתוך "בשורות טובות" - טרי פראצ'ט וניל גיימן

לאחר 20-30 ימי משלוח והמתנה של כשעתיים (שהרגישה אינסופית) בסניף הדואר המקומי, התשובה הגיע בקופסא קטנה:



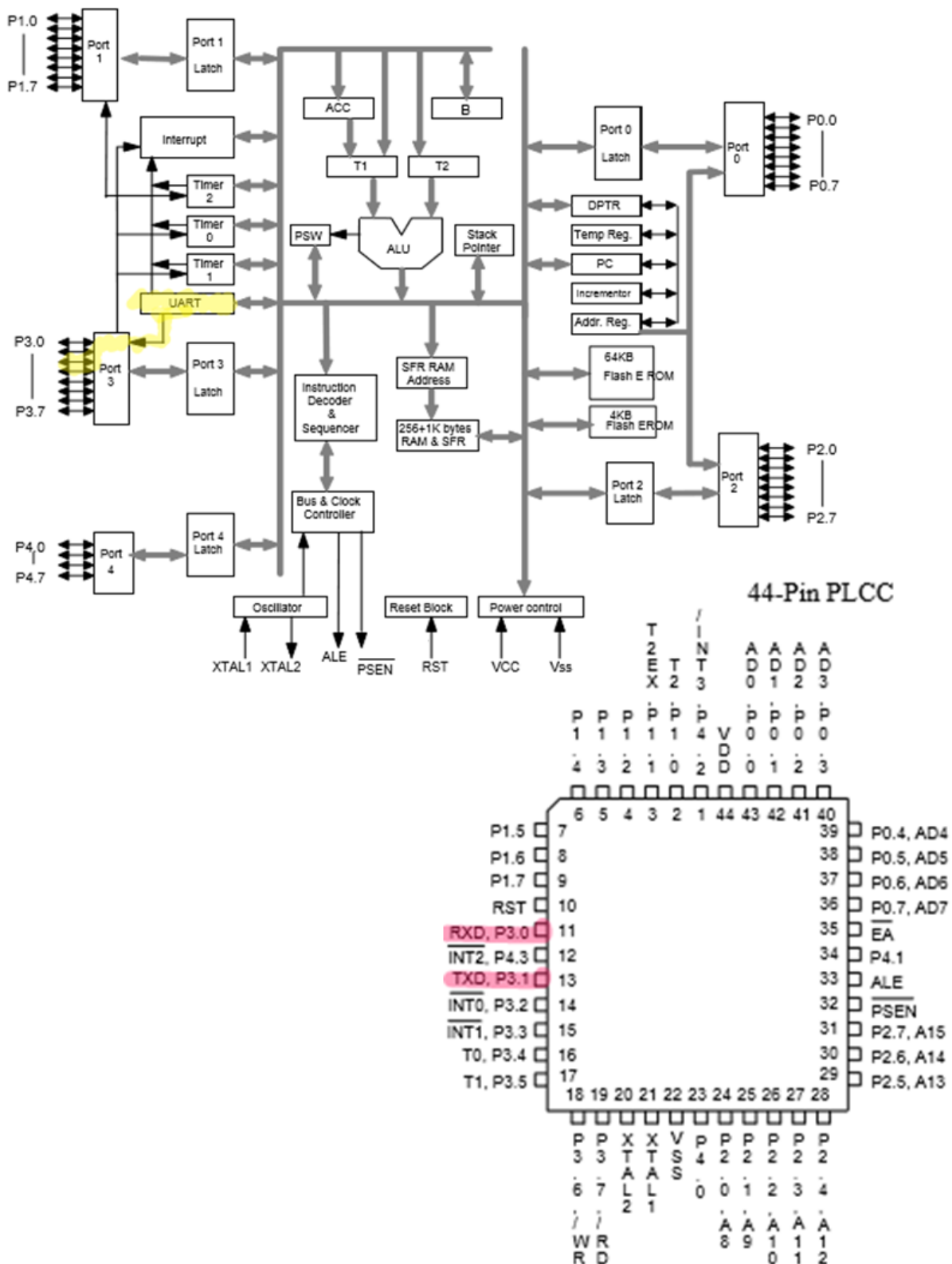
תכירו בבקשה את Logic-Analyzer - Logic מבית היוצר של חברה בשם Saleae.

לאילו מכם שלא מכירים Logic-Analyzer, הוא ציוד בדיקה אלקטרוני שמאפשר לחבר תפסים קטנים לנקודת מגע כלשהי במעגל אלקטרוני, ו-"להקליט" את האותות החשמליים שעוברים דרכה. לאחר מכן ניתן לתרגם את האותות האלו למידע בינארי

מצוין, אז בעצם עכשיו אנו יכולים לחבר את LOGIC לרגלי המעבד 8052 שלנו, לבצע שידרוג קושחה מחדש וכך לראות את המידע שנכנס אל תוך המעבד בזמן העידכון.

אבל לאיזה מבין 44 הרגליים של הציפ צריך להתחבר? ואיך בכלל הציפ הזה מתעדכן?

יש רק מקום אחד בו נוכל למצוא את התשובות האלו - חוברת ההגדרות של הציפ, הידועה בדר"כ בשמה הלועזי המקוצר - "Chip Spec":



How to turn your KVM into a raging Key-Logging monster - בשורות רעות

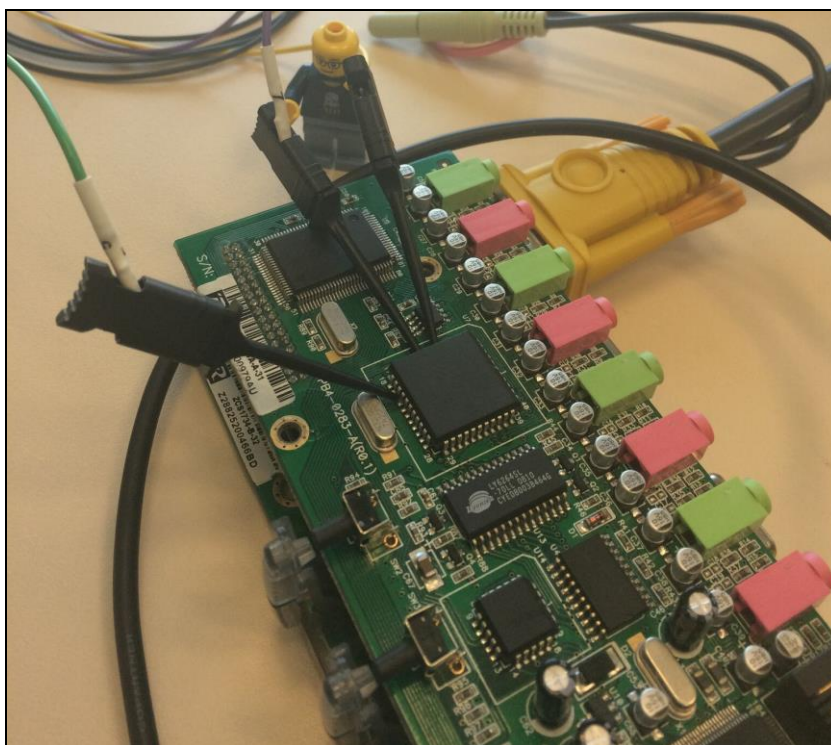
www.DigitalWhisper.co.il

אז מסתבר שלכל מעבד 8051/8052 יש ממשק UART מובנה. לאלו מכם שלא מכירים את המונח UART, מדובר במעין ממשק סיריאלי גנרי "Universal Async Receive Transmit" - שמשמש בעיקר פרוטוקולים כגון RS232 ואחרים.

בציפ הספציפי שלנו ישנן 2 רגליים שמשמשות כ-RX ו-TX של ממשק ה-UART בזמן העידכון - רגליים מספר 11 ו-13 בהתאמה.

אוקי, אז כל מה שנותר לנו לעשות הוא לחבר את את LOGIC לרגלי ה-UART ולהתחיל להקליט. בזמן שאנו מחברים את הכל ביחד, אנו מתחילים לתהות לגבי התוצאות האפשריות:

- **תוצאה אפשרית מס' 1** - כשלון מוחלט. לא הצלחנו להקליט כלום, או לפחות שום דבר בעל משמעות. זה אומר ששההנחה שלנו היא לחלוטין שגויה והעידכון כלל לא קשור למעבד ה-8052 מה שכנראה יגרום לנו להיכנס לדיכאון עמוק.
- **תוצאה אפשרית מס' 2** - ניצחון מוחלט. איזשהו רכיב אחר בסבך האלקטרוניקה שעל הלוח הזה מקבל את ה-BLOB שלנו, מפענח אותו, ואז מעביר אותו מפוענח אל תוך הציפ ואל ממשק ה-UI שלנו.



נותר לנו רק לצרוך את מעט האלכוהול שנותר, להחזיק אצבעות ולהתחיל את תהליך העידכון...

זהו, תהליך העידכון הסתיים בהצלחה. אנו טוענים את הכל לתוך ממשק ה-UI של LOGIC, שמאפשר לנתח את ההקלטה. מיד מתגלה לנו תבנית מעניינית.

How to turn your KVM into a raging Key-Logging monster חלק ב' - בשורות רעות

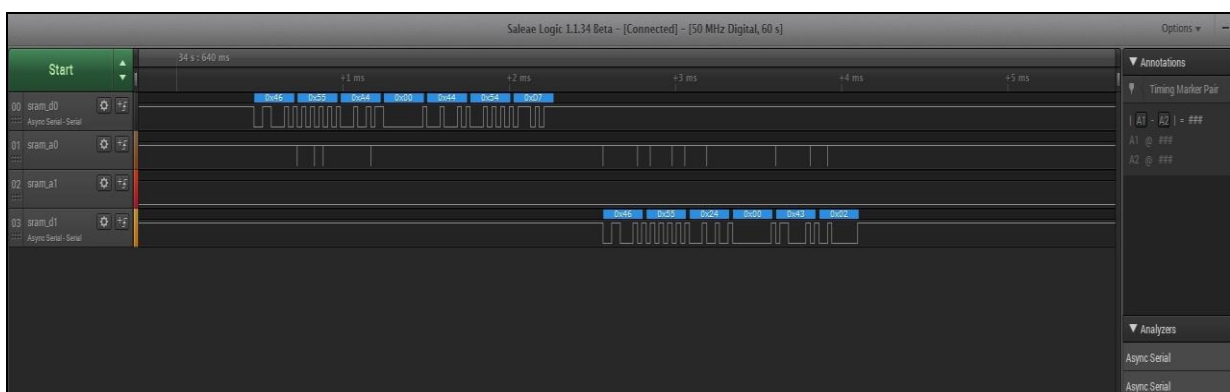
www.DigitalWhisper.co.il



רגלי ה-RX וה-TX משדרות ומקבלות אותות לסירוגין. נראה ממש כמו פרוטוקול סיראלי. כל שנותר לעשות הוא לנסות ו-"לתרגם" את האותות במודולציה הנכונה. לאחר מספר נסיונות מצאנו את המודולציה המתאימה - Asynchronous Serial - שהיא אחת המודולציות הפשוטות ביותר. עבור כל תקתוק שעון מצב בו יש מתח על הקו מסמן את הביט 1 ומצב בו אין מתח מסמן את הביט 0.

בשלב הבא ניתן להשתמש ב-UI שלנו כדי לתרגם את הביטים האלו לערכי ה-HEX שלהם.

כשמסתכלים על התוצאות שקיבלנו אנו נתקלים ברגשות מעורבים.



מה שאנחנו רואים הוא שבדיוק אותו פרוטוקול סיראלי שניתחנו במאמר הראשון הוא זה שמועבר אל תוך הציפ 8052. כלומר, תוכנת העדכון בסה"כ שולחת את כל ה-BLOB דרך הכבל הסיראלי, ישירות לתוך המעבד. מצד אחד, לא הצלחנו לפענח את ה-BLOB שלנו וככל הנראה הפיענוח מתרחש בתוך הציפ עצמו. מצד שני, עכשיו אנו בטוחים שהציפ שלנו הוא היעד של ה-BLOB וזה אומר שהוא כנראה מתורגם בסופו של דבר לאסמבלי 8051 בדיוק כמו שהנחנו בהתחלה.

עבודה יפה KVM, ניצחת אותנו במערכה נוספת, אבל המלחמה ממשיכה ואנו צוברים יותר ויותר ידע בכל שלב! עכשיו כל שנותר לנו לעשות הוא לנסות ולהבין כיצד ה-BLOB שלנו מקודד, ויותר חשוב, איך לקודד אותו בחזרה לאסמבלי 8051 תקין.

המשך יבוא...

נ.ב

עד עכשיו לא ידוע לנו על אף אחד שהצליח לפתור את האתגר שצירפנו למאמר הראשון, פרט לשד טזמני מסוים שעושה רושם שהוא בדרך הנכונה.

אז לצורך החדרת מעט מוטיבציה, אנו מציעים כוס בירה + צ'ייסר חינם לכל מי שיצליח לפתור את האתגר עד לפרסום המאמר האחרון בסידרה. בהצלחה!

How to turn your KVM into a raging Key-Logging monster חלק ב' - בשורות רעות

www.DigitalWhisper.co.il



אתגרי ה-CrackMe של רפאל 2015

מאת דימה פשול

הקדמה

המאמר בא לתאר ולתעד את הדרך בה ניגשתי ופתרתי את אתגרי ה-CrackMe של רפאל במאי השנה. האתגרים פורסמו באתר של רפאל, בקישור הבא:

<http://portal.rafael.co.il/rechallenge15/Documents/rechallenge15/index.html>

האתגרים פורסמו באותו פורמט של האתגרים של השב"כ משנת 2009 ונקראים באותם השמות (stage0.exe - stage3.exe). בכל אתגר על הפותר למצוא סיסמא המדפיסה הודעת ניצחון ו-E-mail איתו ניתן ליצור קשר ולהירשם כפותר האתגר הספציפי. לאורך תיעוד הפיתרון לאתגרים ניסיתי להביא כמה שיותר דוגמאות ותמונות מ-IDA והכלים בהם השתמשתי. בנוסף ניסיתי להביא פתרונות כמה שיותר מקצועיים הבאים לידי ביטוי בפתרון ע"י כתיבת קוד שפותר את האתגרים - כמובן שלא בכלם זה אפשרי. בזמן קריאת המאמר יש לשים לב לכמה דברים:

כתובות ספציפיות

לאורך המאמר נקובות כתובות ספציפיות בקבצים המנותחים אשר משתנות מריצה לריצה. יש לקחת בחשבון שתי משמעויות שנובעות מכך:

1. אם ומי מקוראי המאמר שירצה לשחזר את פתרון ה-Crackme עלול להיתקל בכתובות לא תואמות לאלה המצוינות במאמר.
2. לאורך המאמר ובפתרון כל Crackme התמונות המצורפות להמחשה יכולות להיות משתי הרצות שונות ב-IDA ולכן הכתובות לא יהיו תואמות בין שתי תמונות או יותר.

מה שכן אפשר לעשות בשביל לעקוב אחרי הכתובות הוא להשתמש ב-4 ספרות האחרונות כסימניות, למשל הכתובות הבאות כנראה יכילו את אותו המידע/קוד בין 3 הרצות שונות:

```
0x001360C2  
0x00FD60C2  
0x00C060C2
```

אתגרי ה-CrackMe של רפאל 2015

www.DigitalWhisper.co.il



מושגים וקיצורים

- BP - Breakpoint.
- XREF - Cross Reference (פונקציה ב-IDA המראה איפה הפונקציה / אובייקט / כתובת מאוזכרת לאורך הקוד).
- RUR - Run until execution returns from the current function (אפשרות ריצה ב-IDA המבצעת ריצה עד אחרי ה-retn הראשון).

קישורים

לאורך הקוד יש מילים מודגשות המכילות קישורים לרשימת פונקציות / מושגים הנמצאת בתחתית המאמר, עשו בהם שימוש.

אתגר 1 - stage0.exe:

ידע דרוש לפתרון:

- Registry
- אלגוריתמי קידוד מוכרים
- Windows API

חימום:

נפתח את הקובץ ב-IDA, נגיע ל-Main שנראה ככה:

```
01262040
01262040
01262040 ; Attributes: bp-based frame
01262040
01262040 ; int __cdecl main(int argc, const char **argv, const char **envp)
01262040 _main proc near
01262040
01262040 phkResult= dword ptr -0B0h
01262040 pcchString= dword ptr -0ACh
01262040 cbData= dword ptr -0A8h
01262040 pszString= byte ptr -0A4h
01262040 Data= byte ptr -54h
01262040 var_4= dword ptr -4
01262040 argc= dword ptr 8
01262040 argv= dword ptr 0Ch
01262040 envp= dword ptr 10h
01262040
01262040 push ebp
01262041 mov ebp, esp
01262043 sub esp, 0B0h
01262049 mov eax, security_cookie
0126204E xor eax, ebp
01262050 mov [ebp+var_4], eax
01262053 mov eax, 50h
01262058 mov [ebp+cbData], eax
0126205E mov [ebp+pcchString], eax
01262064 lea eax, [ebp+phkResult]
0126206A push eax ; phkResult
0126206B push 20019h ; samDesired
01262070 push 0 ; u10ptions
01262072 push offset SubKey ; "Software\\rafael"
01262077 push 80000001h ; hKey CurrentUser
0126207C call ds:RegOpenKeyEx
01262082 test eax, eax
01262084 jnz loc_126210B
```

אתגרי ה-CrackMe של רפאל 2015

www.DigitalWhisper.co.il

הקוד מנסה לפתוח את המפתח "HKEY_CURRENT_USER\Software\rafael" של ה-Registry בעזרת [RegOpenKeyExA](#), במידה והוא לא מצליח הוא יקפוצ' להודעת השגיאה, במידה והוא מצליח נגיע לקטע הקוד הבא:

```
0126208A lea     ecx, [ebp+cbData]
01262090 push    ecx                ; lpcbData
01262091 lea     edx, [ebp+Data]
01262094 push    edx                ; lpData
01262095 push    eax                ; lpType
01262096 push    eax                ; lpReserved
01262097 mov     eax, [ebp+phkResult]
0126209D push    offset ValueName ; "Password"
012620A2 push    eax                ; hKey
012620A3 call    ds:RegQueryValueExA
012620A9 test    eax, eax
012620AB jnz     short loc_126210B
```

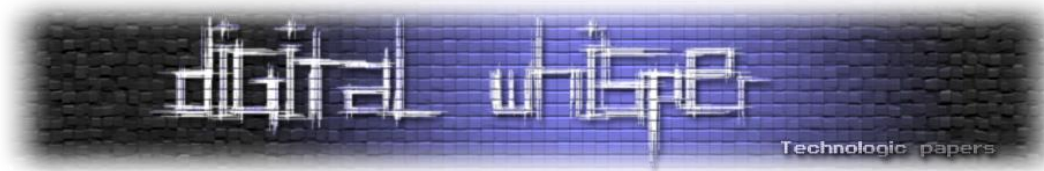
הקוד יבצע תשאול לגבי ה-Value של "Password" בעזרת [RegQueryValueExA](#), לאחר מכן יקודד אותו בעזרת הפונקציה [CryptBinaryToStringA](#) עם קידוד של [Base64](#):

```
012620AD mov     eax, [ebp+cbData]
012620B3 lea     ecx, [ebp+pcchString]
012620B9 push    ecx                ; pcchString
012620BA lea     edx, [ebp+pszString]
012620C0 push    edx                ; pszString
012620C1 push    40000001h        ; dwFlags
012620C6 push    eax                ; cbBinary
012620C7 lea     ecx, [ebp+Data]
012620CA push    ecx                ; pbBinary
012620CB call    ds:CryptBinaryToStringA
012620D1 test    eax, eax
012620D3 jz      short loc_126210B
```

לאחר מכן התכנית תשווה את מה שקיבלה לאחר הקידוד עם ה-String "U3RhZ2UjMHBhc3N3b3JkIQ==" בעזרת הפונקציה [strcmp](#). בשביל לפתור את האתגר הזה עלינו לבצע Decode ל-String הנ"ל - נקבל "Stage#0password!" את ה-String הזה עלינו להכניס לתוך ערך בשם "Password" בנתיב ה-Registry הבא - HKEY_CURRENT_USER\Software\rafael. לאחר הפעולות האלה ניתן להריץ את הקובץ:

```
C:\Windows\system32\cmd.exe
C:\Users\Alien\Downloads\All4Stages>Stage0.exe
Congratulations! You have solved this challenge!
To get the next challenge send an e-mail to the following address: iFv2h1ZnmNGGg@mail.com.
To get the points for the raffle you must explain in the mail how you solved this challenge.
C:\Users\Alien\Downloads\All4Stages>
```

כעת הקובץ ייגש לערך שהוספנו, יקודד אותו בעזרת base64 ויעשה השוואה בין הערך המקודד ל-String הקבוע בקוד (ההשוואה כמובן תהיה נכונה כעת).



אתגר 2 - stage1.exe

ידע דרוש לפתרון:

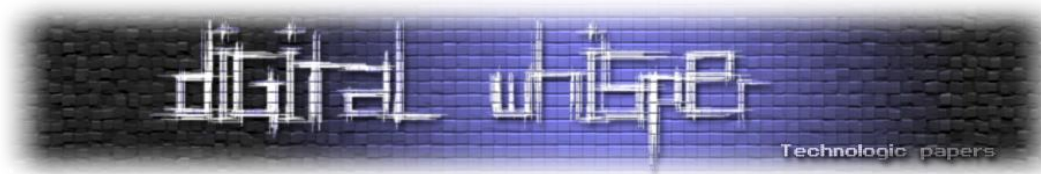
- x86 Assembly
- Anti-Debugging מבוסס זמן

ספרינט 100 מטר:

החלק הראשון של ה-Main נראה ככה:

```
00282000
00282000
00282000 ; Attributes: bp-based frame
00282000 ; int __cdecl main(int argc, const char **argv, const char **envp)
00282000 _main proc near
00282000
00282000 var_200= byte ptr -200h
00282000 var_1FF= byte ptr -1FFh
00282000 var_100= byte ptr -100h
00282000 var_FF= byte ptr -0FFh
00282000 argc= dword ptr 8
00282000 argv= dword ptr 0Ch
00282000 envp= dword ptr 10h
00282000
00282000 push    ebp
00282001 mov     ebp, esp
00282003 sub     esp, 200h
00282009 push    0FFh ; size_t
0028200E lea     eax, [ebp+var_FF]
00282014 push    0 ; int
00282016 push    eax ; void *
00282017 mov     [ebp+var_100], 0
0028201E call    _memset
00282023 push    0FFh ; size_t
00282028 lea     ecx, [ebp+var_1FF]
0028202E push    0 ; int
00282030 push    ecx ; void *
00282031 mov     [ebp+var_200], 0
00282038 call    _memset
0028203D push    offset aPleaseEnterAU$ ; "Please enter a user-name: ...\n"
00282042 call    _printf
00282047 push    100h
0028204C lea     edx, [ebp+var_100]
00282052 push    edx
00282053 push    offset Format ; "%s"
00282058 call    _scanf_s
0028205D lea     eax, [ebp+var_100]
00282063 push    eax ; char *
00282064 call    sub_281B90
00282069 add     esp, 2Ch
0028206C test    eax, eax
0028206E jnz     short loc_28208D
```

הקוד יבקש מאיתנו Username, לאחר מכן יעבור ל-sub_281B90 ויבצע [strncmp](#) של ה-Username שהכנסנו עם ה-String "root". ולכן נריך מחדש וה-String שנכניס לו ב-Username יהיה root.



במידה וה-Username נכון נעבור לקטע הקוד הבא המבקש לקבל את הסיסמא למשתמש, ולאחר מכן עובר ל-**sub_281DE0** בשביל לאמת את הסיסמא:

```

0028208D
0028208D loc_28208D:
0028208D lea     edx, [ebp+var_100]
00282093 push    edx
00282094 push    offset aHelloSPleaseEn ; "Hello %s, please enter the password: ..."
00282099 call    _printf
0028209E push    100h
002820A3 lea     eax, [ebp+var_200]
002820A9 push    eax
002820AA push    offset Format ; "%s"
002820AF call    _scanf_s
002820B4 add     esp, 14h
002820B7 lea     ecx, [ebp+var_100]
002820BD push    ecx ; char *
002820BE lea     edx, [ebp+var_200]
002820C4 push    edx ; void *
002820C5 call    sub_281DE0
002820CA neg     eax
002820CC sbb     eax, eax
002820CE and     eax, 0FFFFFFFh
002820D1 add     eax, 2
002820D4 mov     esp, ebp
002820D6 pop     ebp
002820D7 retn
002820D7 _main endp
002820D7

```

שרשרת הטעיות

תחילת הפונקציה **sub_281DE0** נראית ככה:

```

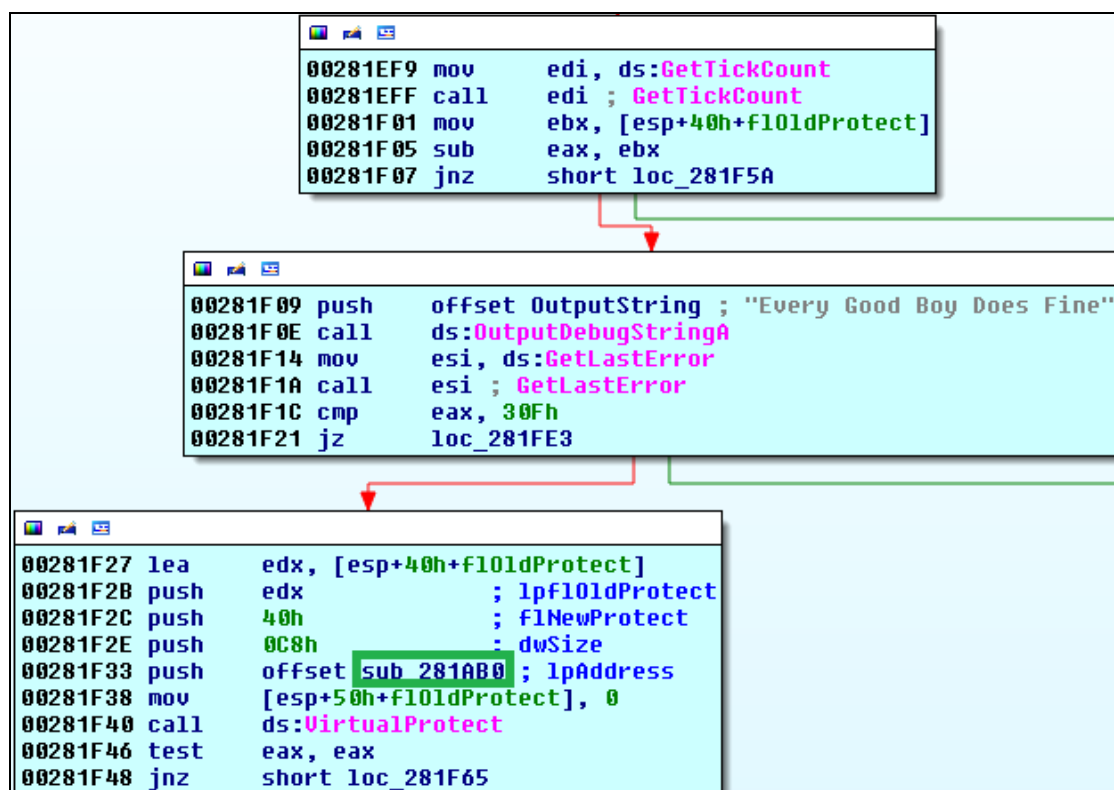
00281DE0
00281DE0
00281DE0 ; Attributes: bp-based frame
00281DE0
00281DE0 ; int __stdcall sub_281DE0(void *, char *)
00281DE0 sub_281DE0 proc near
00281DE0
00281DE0 var_34= dword ptr -34h
00281DE0 var_30= dword ptr -30h
00281DE0 var_2C= dword ptr -2Ch
00281DE0 var_28= dword ptr -28h
00281DE0 f10ldProtect= dword ptr -24h
00281DE0 var_20= dword ptr -20h
00281DE0 var_1C= dword ptr -1Ch
00281DE0 Time= qword ptr -18h
00281DE0 var_C= dword ptr -0Ch
00281DE0 var_8= dword ptr -8
00281DE0 var_4= dword ptr -4
00281DE0 arg_0= dword ptr 8
00281DE0 arg_4= dword ptr 0Ch
00281DE0
00281DE0 push    ebp
00281DE1 mov     ebp, esp
00281DE3 and     esp, 0FFFFFFFh
00281DE6 sub     esp, 34h
00281DE9 push    ebx
00281DEA push    esi
00281DEB push    edi
00281DEC xor     edi, edi
00281DEE push    edi ; Time
00281DEF mov     [esp+4h+var_20], 32CCFA3Ah
00281DF7 mov     [esp+4h+var_1C], 0A750D68h
00281DFF call    _time64
00281E04 mov     dword ptr [esp+4h+Time], eax
00281E08 lea     eax, [esp+4h+Time]
00281E0C add     esp, 4
00281E0F push    eax ; Time
00281E10 mov     dword ptr [esp+4h+Time+4], edx
00281E14 call    _localtime64
00281E19 add     esp, 4
00281E1C mov     [esp+4h+var_28], eax
00281E20 call    ds:GetTickCount
00281E26 mov     [esp+4h+f10ldProtect], eax
00281E2A xor     ecx, ecx
00281E2C mov     [esp+4h+var_30], edi
00281E30 mov     [esp+4h+var_34], edi
00281E34 mov     [esp+4h+var_2C], edi
00281E38 jmp     short loc_281E40

```

אתגרי ה CrackMe של רפאל 2015

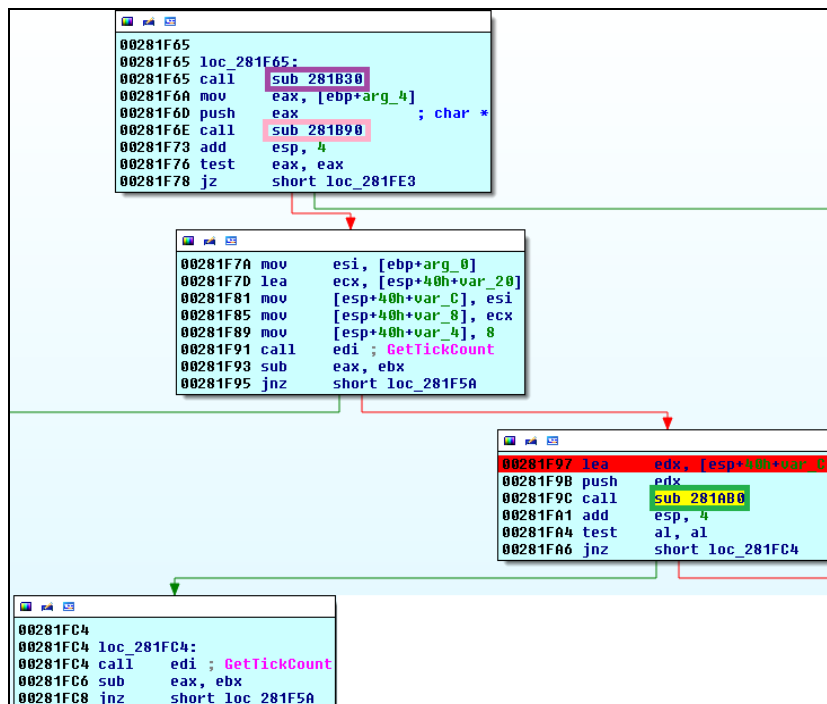
www.DigitalWhisper.co.il

הפונקציה מבצעת שלל קריאות לפונקציות שאינן רלוונטיות לפיתרון האתגר ואינן משמשות אותו בשום צורה, למעט 2 פונקציות והן [VirtualProtect](#) ו-[GetTickCount](#). לאחר הקריאה הראשונה ל-[GetTickCount](#) תבצע [sub_281DE0](#) מספר חישובים שאינם רלוונטיים, לאחר מכן תעבור לקריאה שניה ל-[GetTickCount](#), אחרי קריאה זאת יתבצע חיסור בין הזמן הראשון שהתקבל והזמן השני שהתקבל מ-[GetTickCount](#), במידה ותוצאת החיסור אינה 0, הקוד יברח לנו ולא יבצע בדיקה של הסיסמא, בדיקה זאת חוזרת על עצמה לאורך הקוד 3 פעמים בעזרת 4 קריאות ל-[GetTickCount](#). בשלב הבא יתבצע שימוש ב-[OutputDebugStringA](#), לעיתים פונקציה זאת יכולה לשמש כ-Anti-Debugging, המקרה הזה אינו כזה ומודפס לנו ה-String: "Every Good Boy Does Fine", לאחר ההדפסה שוב מתבצעת בדיקת זמנים ע"י [GetTickCount](#).

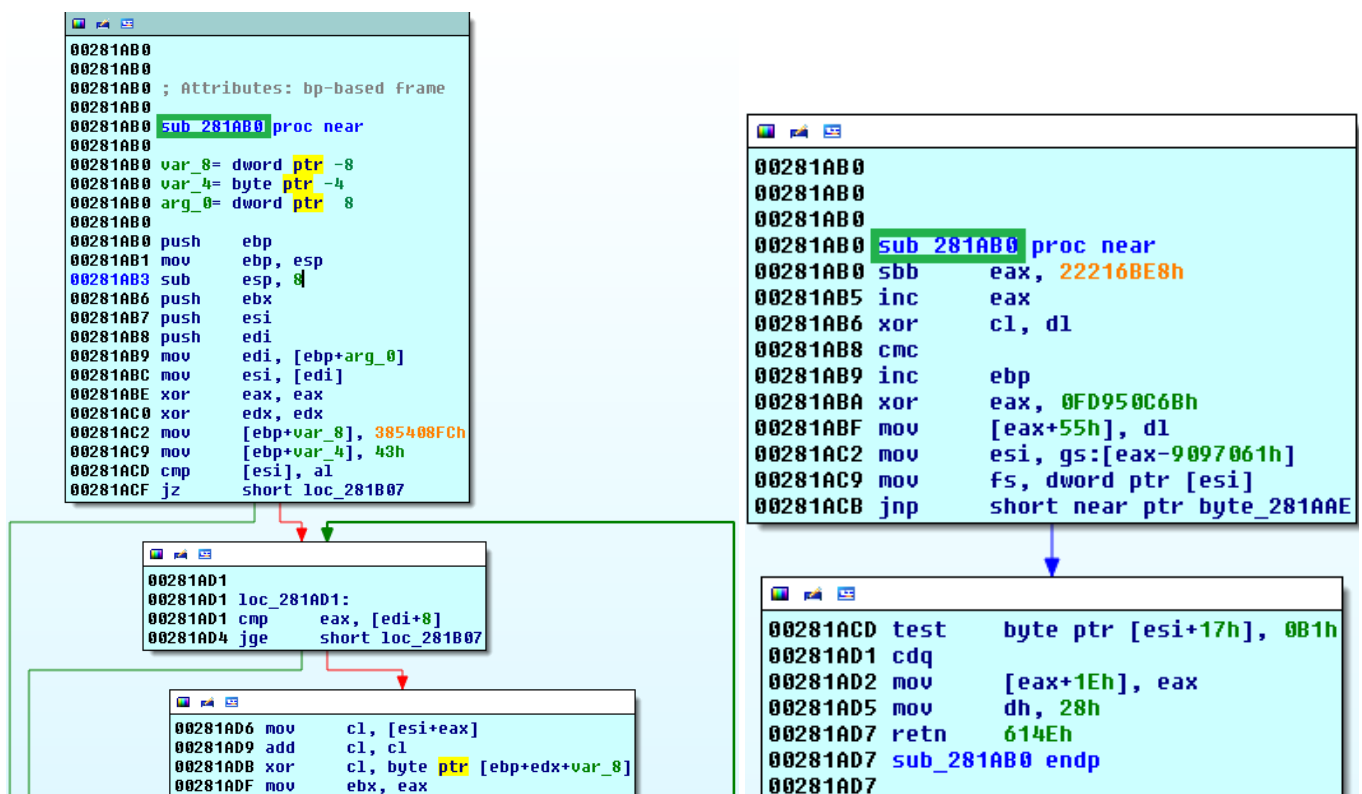


בחלק הבא יתבצע [VirtualProtect](#) אשר ישנה את הרשאות הזכרון לפונקציה של [sub_281AB0](#) ל-RW (Read Write) בשביל שהקוד יוכל בעתיד לפענח ולשנות את הפונקציה בזמן הריצה מאחר וברירת המחדל היא Read (אם נציץ בתוכה נראה שהיא אינה נראית תקינה).

נעבור לקטע הקוד הבא:

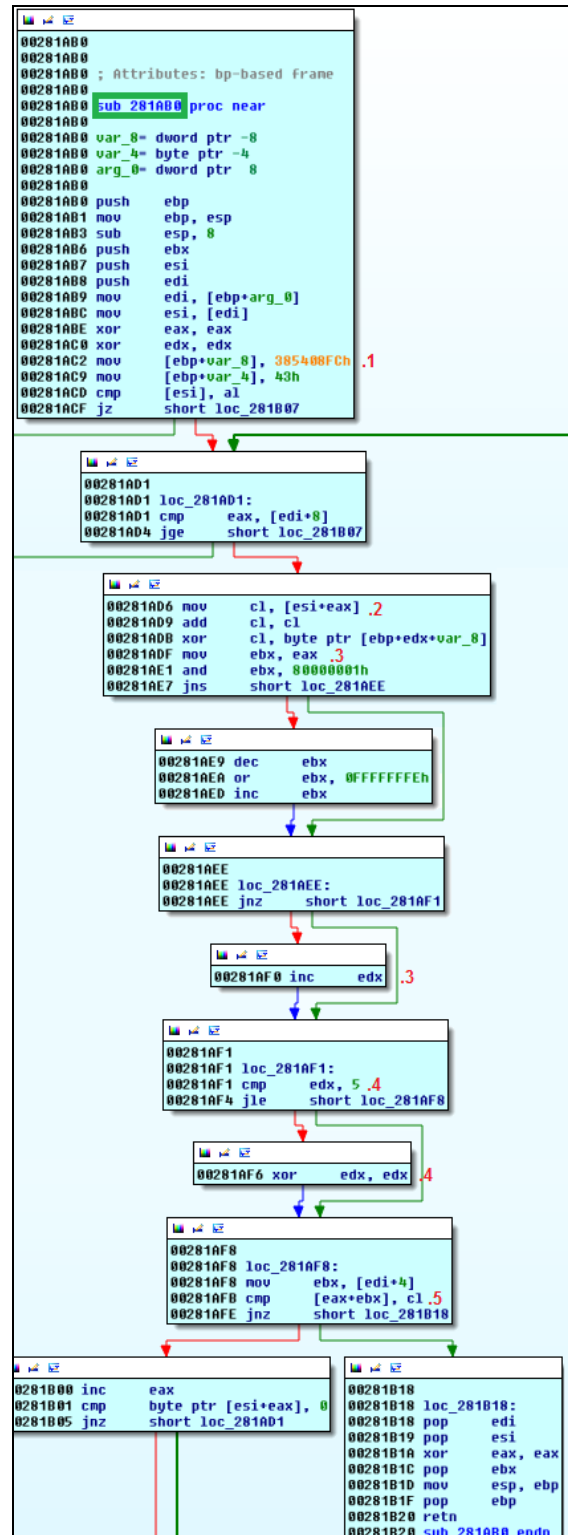


הפונקציה sub_281B30 תבצע פיענוח ל-sub_281AB0:



לפני

הפונקציה **sub_281B90** מבצעת השוואה נוספת של "root" אל מול ה-Username שהוכנס. לפני הכניסה ל-**sub_281AB0** תתבצע עוד בדיקת השוואת זמנים ואם נעבור אותה ניכנס לפונקציה **sub_281AB0** שהיא פונקציית בדיקת הסיסמא. הפונקציה נראית ככה:



אתגרי ה CrackMe של רפאל 2015

www.DigitalWhisper.co.il



הפונקציה כוללת 5 נקודות ציון ובמהלך ריצתה עוברת על ה-Byte-ים של הסיסמא שהוכנסה אחד-אחד:

1. אתחול המפתח (0xFC08543843)

2. האלגוריתם המתמטי:

א. `mov cl, [esi+eax]` - העברת ה-byte שנבדק ל-cl

ב. `cl = cl * 2 - add cl, cl`

ג. `xor cl, byte ptr [ebp+edx+var_8]` - ביצוע xor של cl עם ה-Byte עליו מצביע המפתח edx הוא

ה-Offset שלנו.

3. במידה ומספר לולאת הריצה שלנו מתחלק ב-2 נגדיל את ה-offset אשר מצביע למפתח (edx).

4. במידה וה-Offset שווה ל-5 הוא יתאפס.

5. השוואה של cl מול תא במערך של Byte-ים ב-offset של מספר הריצות:

`cl == Byte_array[Counter]`.

מערך ה-Byte-ים:

Stack[00000510]:0046F5D8	db	3Ah	; :
Stack[00000510]:0046F5D9	db	0FAh	; -
Stack[00000510]:0046F5DA	db	0CCh	; :
Stack[00000510]:0046F5DB	db	32h	; 2
Stack[00000510]:0046F5DC	db	0B0h	; :
Stack[00000510]:0046F5DD	db	0D6h	; +
Stack[00000510]:0046F5DE	db	50h	; P
Stack[00000510]:0046F5DF	db	0A7h	; :
Stack[00000510]:0046F5E0	db	0BFh	; +
Stack[00000510]:0046F5E1	db	0C7h	; :
Stack[00000510]:0046F5E2	db	0FDh	; 2
Stack[00000510]:0046F5E3	db	55h	; U

הסקריפט הבא ב-Python מוצא את הסיסמא הדרושה:

```
xorkey = "\xFC\x08\x54\x38\x43"
cmpkey = "\x3a\xfa\xcc\x32\xb0\xd6\x50\xa7"

decoded = ""
i = 0
run_count = 0
for c in cmpkey:
    xored = 0
    xored = ord(c) ^ ord(xorkey[i])
    xored = xored / 2
    decoded += chr(xored)
    if (run_count % 2) == 0:
        i += 1
    if i == 5:
        i = 0
    run_count += 1

print decoded
```

אתגרי ה-CrackMe של רפאל 2015

www.DigitalWhisper.co.il

שמירתו לקובץ והרצתו, תניב את הפלט הבא:

```
C:\Windows\system32\cmd.exe

C:\Users\Alien\Desktop>decode-stage1-rafael.py
cyb3rw4r
C:\Users\Alien\Desktop>_
```

לשם הספורט אני אציין כי יש עוד בדיקת [GetTickCount](#) לאחר בדיקת הסיסמא ולפני הדפסת הודעת ההצלחה.

אם כן, נראה שסיימנו...

```
C:\Windows\system32\cmd.exe

C:\Users\Alien\Downloads\All4Stages>stage1.exe
Please enter a user-name: ...
root
Hello root, please enter the password: ...
cyb3rw4r
Good news!

<C> Rafael Advanced Defense Systems Ltd. 2015
Congratulations!
You have solved this challenge!
For the next challenge, please send an e-mail to the following address: 0s8C4Ds7
MKA@gmail.com.
To be eligible to participate in the raffle, you must explain in the mail, how y
ou solved this challenge.

C:\Users\Alien\Downloads\All4Stages>
```



אתגר 3 - stage2.exe:

ידע דרוש לפתרון:

- פורמט קובץ PE
- Exception Handling
- Exception Anti-Debugging
- x86 Assembly

מאוררי Hash-ים או Hashing Fans

נצלול ישר ל-main שנראה כך:

```
01362980
01362980
01362980 ; Attributes: bp-based frame
01362980
01362980 ; int __cdecl main(int argc, const char **argv, const char **envp)
01362980 _main proc near
01362980
01362980     argc= dword ptr  8
01362980     argv= dword ptr  0Ch
01362980     envp= dword ptr  10h
01362980
01362980     push    ebp
01362981     mov     ebp, esp
01362983     push    offset unk_139BEC8
01362988     push    offset aEnterPassword ; "Enter password:"
0136298D     call    _printf
01362992     add     esp, 8
01362995     call    sub_136CDA6
0136299A     push    eax                ; FILE *
0136299B     push    4Fh                ; int
0136299D     push    offset byte_139BF18 ; char *
013629A2     call    _fgets
013629A7     add     esp, 0Ch
013629AA     call    sub_1362780
013629AF     push    5F048AF0h
013629B4     call    sub_13628B0
013629B9     add     esp, 4
013629BC     mov     dword_139BF6C, eax
013629C1     push    offset sub_1362450
013629C6     call    dword_139BF6C
013629CC     int     2Dh                ; Windows NT - debugging services: eax = type
013629CE     xor     eax, eax
013629D0     pop     ebp
013629D1     retn
013629D1 _main endp
013629D1
```

sub_13628B0 הוא מימוש ידני של [GetProcAddress](#) אשר מקבל Hash של שם הפונקציה ועל פיו מוצא את כתובת הפונקציה (נקרא לו MGPA מעכשיו - ר"ת של Manual GetProcAddress⁴). ניתן לדלג על הפונקציה הזאת בכל פעם ורק להתבונן בערך ההחזר (eax) של הפונקציה בשביל לבדוק איזה מצביע (לפונקציה) הוחזר מהפונקציה.

⁴https://books.google.co.il/books?id=FOC8EPYy834C&pg=PA413&lpg=PA413&dq=manual+GetProcAddress&source=bl&ots=BtknkBlfah&sig=YCE3B2G3pOm_vS_resJXLq_XSk&hl=en&sa=X&ved=0CCgQ6AEwAjgKahUKewjAx6yG5PnHAhWFxxQKHTOCCxM#v=onepage&q=manual%20GetProcAddress&f=false

מִיד לֵאחֶר הַתְּקִנָּה ה-Exception Handler⁵.

חיצונית כלשהי, אלא רוצים לתת לקובץ לרוץ בצורה טבעית.

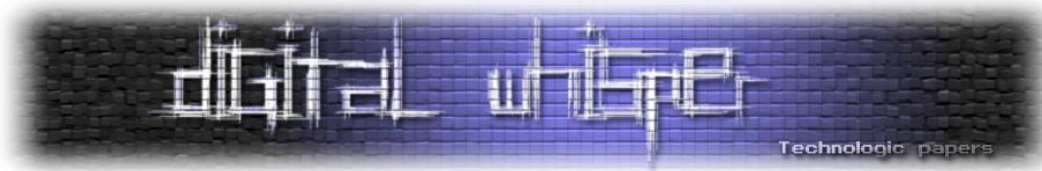
מתבצעת קריאה ל-NtQueryInformationProcess עם הקבוע 0x7 (ProcessDebugPort) בצורה הבאה:

```
NtQueryInformationProcess(x, y, ProcessDebugPort, z);
```

לאחר מכן נגיע לקטע קוד הבא:

```
013624F3
013624F3 loc 13624F3:
013624F3 movsx cx, byte ptr aTestingThis_0+3
013624FB movzx edx, cx
013624FE push edx
013624FF movsx ax, byte ptr aTestingThis_0+2
01362507 movzx ecx, ax
0136250A push ecx
0136250B movsx dx, byte ptr aTestingThis_0+1
01362513 movzx eax, dx
01362516 push eax
01362517 movsx cx, byte ptr aTestingThis_0 ; "testing-this\n"
0136251F movzx edx, cx
01362522 push edx
01362523 mov eax, [ebp+var_C]
01362526 movzx ecx, word_1396630[eax*2]
0136252E push ecx
0136252F call sub_1362650
01362534 add esp, 14h
01362537 mov [ebp+var_D], al
0136253A mov edx, [ebp+var_8]
0136253D push edx ; FILE *
0136253E movsx eax, [ebp+var_D]
01362542 push eax ; int
01362543 call _fputc
01362548 add esp, 8
```

⁵ <http://fumalwareanalysis.blogspot.co.il/2011/09/malware-analysis-3-int2d-anti-debugging.html>



loc_13624F3 יחזור על עצמו 0x1C00 פעמים. בכל ריצה הוא יקח את ארבעת ה-Byte-ים הראשונים של הסיסמא שהוכנסה וידחוף אותם לפונקציה **sub_1362650** יחד עם Word ממערך של word-ים קבוע אשר נמצא ב-word_1396630. הקריאה לפונקציה נראית ככה:

```
sub_1362650(const_word_array[counter] , password[0], password[1], password[2], password[3]);
```

sub_1362650 נראה ככה:

```
01362650 sub_1362650 proc near
01362650
01362650 var_8= dword ptr -8
01362650 var_4= dword ptr -4
01362650 arg_0= word ptr 8
01362650 arg_4= word ptr 0Ch
01362650 arg_8= word ptr 10h
01362650 arg_C= word ptr 14h
01362650 arg_10= word ptr 18h
01362650
01362650 push ebp
01362651 mov ebp, esp
01362653 sub esp, 8
01362656 movsx eax, [ebp+arg_C]
0136265A movsx ecx, [ebp+arg_C]
0136265E imul eax, ecx
01362661 xor eax, 5
01362664 mov [ebp+arg_8], ax
01362668 movsx edx, [ebp+arg_C]
0136266C movsx eax, [ebp+arg_8]
01362670 imul edx, eax
01362673 sub edx, 3
01362676 mov [ebp+arg_C], dx
0136267A movsx ecx, [ebp+arg_0]
0136267E movsx edx, [ebp+arg_4]
01362682 sub ecx, edx
01362684 shl ecx, 2
01362687 movsx eax, [ebp+arg_8]
0136268B shl eax, 2
0136268E mov edx, 10h
01362693 sub edx, eax
01362695 movsx eax, [ebp+arg_C]
01362699 lea edx, [edx+eax*4]
0136269C imul edx, 3
0136269F sub ecx, edx
013626A1 movsx eax, [ebp+arg_10]
013626A5 shl eax, 2
013626A8 sub ecx, eax
013626AA movsx edx, [ebp+arg_8]
013626AE mov eax, 4
013626B3 sub eax, edx
013626B5 movsx edx, [ebp+arg_C]
013626B9 add eax, edx
013626BB imul eax, 0Ch
013626BE add ecx, eax
013626C0 mov [ebp+var_4], ecx
013626C3 mov eax, [ebp+var_4]
013626C6 cdq
013626C7 mov ecx, 0Ch
013626CC idiv ecx
013626CE mov [ebp+var_8], eax
013626D1 mov eax, [ebp+var_8]
013626D4 add eax, 20h
013626D7 movsx edx, [ebp+arg_4]
013626DB xor eax, edx
013626DD mov esp, ebp
013626DF pop ebp
013626E0 ret
013626E0 sub_1362650 endp
013626E0
```

אפשר להיכנס לניתוח האלגוריתם ואח"כ להבין איך להתקדם הלאה אך אני מעדיף להראות דרך יותר יצירתית לפתרון ה-Crackme הזה. מה שחשוב להבין מהפונקציה הזאת היא שהיא מעבדת את המידע שהיא מקבלת ככה שלא ניתן לשחזר את המידע המקורי (פונקציית Hash) ועל כל ריצה היא מחזירה Byte. זאת אומרת שהגדרת הפונקציה נראית ככה:

```
BYTE hashFunc(WORD w, BYTE a, BYTE b, BYTE c, BYTE d);
```

נבצע זום אאוט לפונקציה **loc_13624F3**, קטע הקוד מקבל את ערך ההחזר של **sub_1362650** וכותב אותו לקובץ שנפתח בתחילת הפונקציה **sub_1362450** בעזרת **Fputc**.

נמשיך וגניע לקטע הקוד **loc_1362550**:

```

01362550
01362550 loc_1362550:
01362550 mov     ecx, [ebp+var_8]
01362553 push    ecx                ; FILE *
01362554 call    _fclose
01362559 add     esp, 4
0136255C push    0A498EAB6h        ; SetErrorMode hash
01362561 call    manual_getProcAddress
01362566 add     esp, 4
01362569 mov     funcSetErrorMode, eax
0136256E push    1
01362570 call    funcSetErrorMode
01362576 push    0EC0E4E8Eh        ; LoadLibraryA hash
0136257B call    manual_getProcAddress
01362580 add     esp, 4
01362583 mov     funcLoadLibraryA, eax
01362588 push    offset aTmp0_x_0 ; "tmp0.X"
0136258D call    funcLoadLibraryA
01362593 mov     [ebp+var_4], eax
01362596 push    7C0DFCAAh        ; GetProcAddress hash
0136259B call    manual_getProcAddress
013625A0 add     esp, 4
013625A3 mov     funcGetProcAddress, eax
013625A8 push    offset aTmp0_x_1 ; "tmp0.X"
013625AD call    _remove
013625B2 add     esp, 4
013625B5 cmp     [ebp+var_4], 0
013625B9 jz      short loc_1362634
    
```

```

0136258B push    offset aQualify ; "qualify"
013625C0 mov     edx, [ebp+var_4]
013625C3 push    edx
013625C4 call    funcGetProcAddress
013625CA mov     funcQualify, eax
013625CF cmp     funcQualify, 0
013625D6 jz      short loc_1362634
    
```

קטע הקוד מבצע MGPA לפונקציות [SetErrorMode](#), [LoadLibraryA](#), [GetProcAddress](#) לאחר מכן קורא להן. הקריאות המעניינות אותנו הן [LoadLibraryA](#) ו-[GetProcAddress](#).

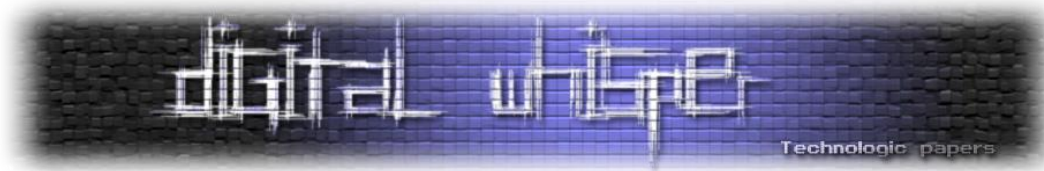
הקוד מעביר ל-[LoadLibraryA](#) את הקובץ "tmp0.X" - הפונקציה מצפה לקבל Dll, במידה והקובץ לא יהיה בפורמט של PE הפונקציה לא תתבצע בהצלחה. ניתן להבין כי הקובץ הנכתב צריך להיות בפורמט של [PE](#). אחרי טעינת הקובץ הקוד מבצע קריאה ל-[GetProcAddress](#) ומבקש את הפונקציה "qualify" שכנראה מיוצאת ע"י "tmp0.X".

אם כן, הרעיון הוא להכניס 4-byte-ים ככה שאחרי מעבר של הפונקציה [sub_1362650](#) על כל מערך ה-Word-ים נקבל קובץ [PE](#) המייצא את הפונקציה "qualify".

כל word במערך ה-word-ים מייצג byte של הקובץ הסופי, מהתבוננות קצרה ב-Header של קובץ [PE](#) ניתן לבצע המרה בין ה-Word הקיים במערך בתא במסוים לבין ב-Byte שהוא אמור לייצג. ההמרה נראית ככה (6-byte-ים ראשונים):

```

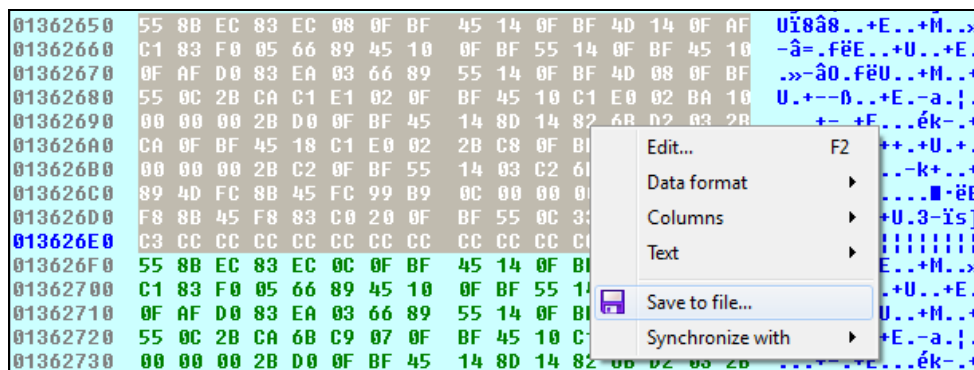
.data:01396630 word_1396630 dw 142h = 0x4D
.data:01396630
.data:01396632 dw 181h = 0x5A
.data:01396634 dw 223h = 0x90
.data:01396636 dw 73h = 0x00
.data:01396638 dw 7Ch = 0x03
.data:0139663A dw 73h = 0x00
    
```



דרך הפעולה בה בחרתי היא ביצוע BruteForce מחושב, הצעדים לביצוע הם כדלהלן:

1. שיחזור אלגוריתם ה-Hash.
2. מציאת ערכים קבועים במקומות קבועים ב-Format של PE עליהם אפשר להסתמך בשביל ה-BruteForce.

הצעד הראשון: מה שנחמד מאוד בפונקציה **sub_1362650** הוא שהיא פונקציה שאינה תלויה בשום משתנה או מצביע בקוד. היא מקבלת ומבצעת בעזרת ה-Stack והאוגרים ולכן ניתן לבצע לה Dump לדיסק ולטעון אותה דרך תוכנית שאנחנו נכתוב מהדיסק, להעביר לה פרמטרים והיא תחזיר לנו תשובות, כמה נוח.



הצעד השני: כתיבת תכנית אשר בודקת סיסמאות ובדיקתה.

ביצוע כמה ריצות נסיון של תכנית ה-BruteForce והוספת ערכים קבועים שערכם אחרי ההמרה ידוע מראש, למשל ארבעת ה-Byte-ים הראשונים:

0x0142	=	0x4D
0x0181	=	0x5A
0x0223	=	0x90
0x0073	=	0x00

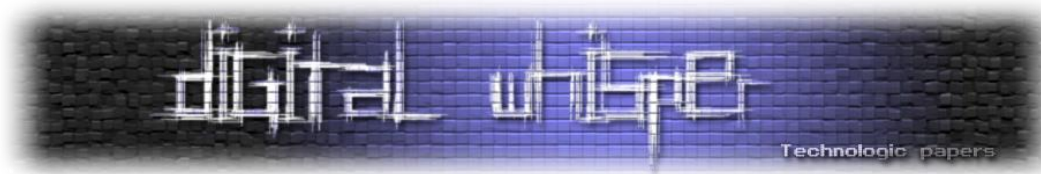
אחרי הוספתם עדיין יתפסו הרבה False Positives ב-BruteForce. לאחר התבוננות בתוצאות התכנית והשוואה מול קבצי PE אחרים ניתן לראות כי ה-Byte שמייצג space (0x20), למשל ב-String המוכר: "This program cannot be run in DOS mode", אינו נכון ומיוצג ע"י "!" (0x21). מכאן אפשר לקחת את ה-Offset של ה-0x21 byte וללכת לאותו offset במערך ה-word-ים ולהציב את ההמרה כתנאי נוסף בתכנית - לאחר מעבר בפונקציית ה-Hash ה-Word 0x0013 צריך להיות שווה ל-0x20.

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	Ascii
00000000	4D	5A	90	00	03	00	00	00	04	00	00	00	FF	FF	00	00	MZ
00000010	B8	00	00	00	00	00	00	00	40	00	00	00	00	00	00	00@
00000020	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000030	00	00	00	00	00	00	00	00	00	00	00	00	F0	00	00	00
00000040	0E	1F	BA	0E	00	B4	09	CD	22	B8	01	4C	CD	22	54	68I", LI"Th
00000050	69	73	21	70	72	6F	67	72	61	6D	21	63	61	6E	6E	6F	is!program!canno
00000060	74	21	62	65	21	72	75	6E	21	69	6E	21	44	4F	53	21	t!be!run!in!DOS!
00000070	6D	6F	64	65	2F	0D	0D	0A	25	00	00	00	00	00	00	00	mode/...%

לא תקין

אתגרי ה-CrackMe של רפאל 2015

www.DigitalWhisper.co.il



Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	Ascii
00000000	4D	5A	90	00	03	00	00	00	04	00	00	00	FF	FF	00	00	MZ .L...J...ÿÿ..
00000010	B8	00	00	00	00	00	00	00	40	00	00	00	00	00	00	00@.....
00000020	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000030	00	00	00	00	00	00	00	00	00	00	00	00	E0	00	00	00à....
00000040	0E	1F	BA	0E	00	B4	09	CD	21	B8	01	4C	CD	21	54	68	...!..I!..Li!Th
00000050	69	73	20	70	72	6F	67	72	61	6D	20	63	61	6E	6E	6F	is.program.canno
00000060	74	20	62	65	20	72	75	6E	20	69	6E	20	44	4F	53	20	t.be.run.in.DOS.
00000070	6D	6F	64	65	2E	0D	0D	0A	24	00	00	00	00	00	00	00	mode....\$.....

תקין

מכאן נובע כי יש לנו 5 תנאים, 5 תנאים שהם מספקים את התכנית והופכות אותה למספיק חכמה, נשאר רק לכתוב פונקציה אשר מגדילה כל פעם את ה-dword המוכנס אליה, ושומרת עליו בין הערכים המתקבלים בקלט (רק תווים שיש להם קידוד ASCII ומתקבלים בקלט של קונסול, 0x20 - 0x7E).

Flow התוכנית:

1. קבלת סיסמת התחלה (0x202020).
 2. הרצת הסיסמא בפונקציית ה-Hash אל מול 5 התנאים שהצבנו.
 3. במידה והסיסמא עומדת בכל התנאים - יש לנו סיסמא.
 4. במידה ולא עומדת בתנאי - מעבר לסיסמא הבאה ע"י הגדלת הסיסמא ב-1 תוך כדי שמירתה בגבולות תווי ASCII. (אחרי 0x202020 תבוא 0x202021 ואחרי 0x20207E תבוא 0x202120 עד הסיסמא המקסימלית שהיא 0x7E7E7E7E)
- התכנית שלנו שתבצע את ה-Bruteforce נראית ככה (כתוב ב++C):

```
#include "stdafx.h"
#include "Windows.h"

typedef DWORD (*hashFunc) (WORD w, BYTE a, BYTE b, BYTE c, BYTE d);

#define ASCII_MIN 32
#define ASCII_MAX 126
#define PASS_INIT 0x20202020
#define PASS_MAX 0x7E7E7E7E
#define PASS_NO_MORE_AVAILABLE 0xFFFFFFFF
#define FUNC_FILE "func_raw"

DWORD incPassword(DWORD dwPass)
{
    if ((dwPass & 0xff) < ASCII_MAX)
    {
        dwPass += 1;
    }
    else if (((dwPass >> 8) & 0xff) < ASCII_MAX)
    {
        dwPass += 0x100;
        dwPass -= 94;
    }
    else if (((dwPass >> 16) & 0xff) < ASCII_MAX)
    {
        dwPass += 0x10000;
        dwPass -= 11776;
        dwPass -= 94;
    }
    else if (((dwPass >> 24) & 0xff) < ASCII_MAX)
    {
        dwPass += 0x1000000;
        dwPass -= 3014656;
        dwPass -= 11776;
    }
}
```

אתגרי ה-CrackMe של רפאל 2015

www.DigitalWhisper.co.il



```
        dwPass -= 94;
    }
    else if (dwPass == PASS_MAX)
    {
        return PASS_NO_MORE_AVAILABLE;
    }
    return dwPass;
}

int _tmain(int argc, _TCHAR* argv[])
{
    DWORD dwPasswordCounter = 0;
    HANDLE hFunc = CreateFileA(FUNC_FILE, FILE_READ_ACCESS, NULL, NULL, OPEN_EXISTING, NULL,
    NULL);
    if (hFunc != INVALID_HANDLE_VALUE)
    {
        DWORD dwFileSize = GetFileSize(hFunc, NULL);
        LPVOID lpBuffer = VirtualAlloc(NULL, dwFileSize, MEM_COMMIT, PAGE_EXECUTE_READWRITE);
        DWORD dwBytesRead;
        if (ReadFile(hFunc, lpBuffer, dwFileSize, &dwBytesRead, NULL))
        {
            DWORD dwPass = PASS_INIT;
            DWORD dwResult;
            hashFunc hash = (hashFunc)lpBuffer;

            while (dwPass != PASS_NO_MORE_AVAILABLE)
            {
                dwResult = hash(0x0142, (dwPass >> 24) & 0xff, (dwPass >> 16) & 0xff, (dwPass >> 8)
                & 0xff, dwPass &
                0xff);

                dwResult &= 0xff;
                if (dwResult == 0x4D)
                {
                    dwResult = hash(0x0181, (dwPass >> 24) & 0xff, (dwPass >> 16) & 0xff, (dwPass >> 8)
                    & 0xff, dwPass & 0xff)

                    dwResult &= 0xff;
                    if (dwResult == 0x5A)
                    {
                        dwResult = hash(0x0223, (dwPass >> 24) & 0xff, (dwPass >> 16) & 0xff, (dwPass >>
                        8) & 0xff, dwPass & 0xff);

                        dwResult &= 0xff;
                        if (dwResult == 0x90)
                        {
                            dwResult = hash(0x0073, (dwPass >> 24) & 0xff, (dwPass >> 16) & 0xff, (dwPass >>
                            8) & 0xff, dwPass
                            & 0xff);

                            dwResult &= 0xff;
                            if (dwResult == 0x00)
                            {
                                dwResult = hash(0x0013, (dwPass >> 24) &
                                0xff, (dwPass >> 16) & 0xff,
                                (dwPass >> 8) & 0xff, dwPass & 0xff);
                                dwResult &= 0xff;
                                if (dwResult == 0x20)
                                {
                                    printf("this is it - 0x%4x\n",
                                    dwPasswordCounter++);
                                }
                            }
                        }
                    }
                }
            }
        }
        dwPass = incPassword(dwPass);
    }
}
```

אתגרי ה CrackMe של רפאל 2015

www.DigitalWhisper.co.il

```

    }
    printf("%d passwords found", dwPasswordCounter);
    getchar();
}
return 0;
}

```

התכנית מוצאת 2209 סיסמאות שונות.

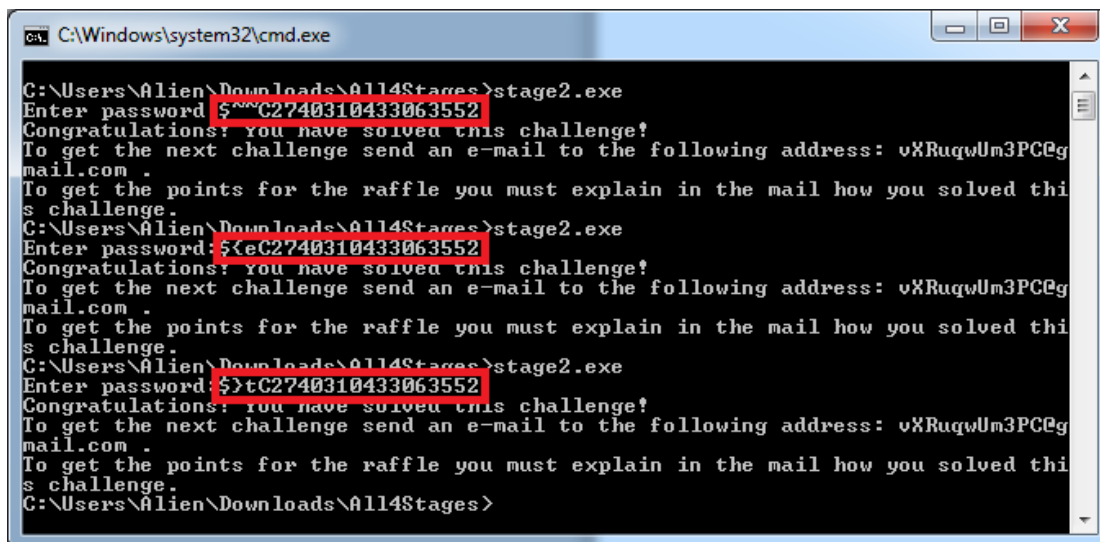
נשאר החלק האחרון, הטעינה של "tmp0.X" ע"י [LoadLibraryA](#) והקריאה ל-"qualify" מה-Export-ים של "tmp0.X". ניתן לקוד לטעון את "tmp0.X" ונשים BP בכתובת שהתקבלה ע"י [GetProcAddress](#) ממציאת הפונקציה "qualify" כמה צעדים בתוך הפונקציה יראו לנו שיש ביצוע של [strcmp](#) בין הסיסמא שהכנסנו ב-Offset של 4 byte לבינ ה-string "2740310433063552" ניתן להסיק כי הסיסמא שלנו תהיה בנויה ככה: XXXX2740310433063552 כאשר XXXX הם ארבעת התווים שיגרמו ל-"tmp0.X" להפוך ל-Dll (ארבעת תווים אלה הם המשתנים היחידים בסיסמא, התכנית הנ"ל מבצעת BruteForce ומציאה שלהם).

מדגמית נבדקו 3 סיסמאות בצורה מוצלחת (יש לשים לב כי מחליפים Byte שהוא אינו ה-Byte ה-3 בסיסמא כי אין לו השפעה באלגוריתם ה-Hash לכן הסיסמא הבאה: "C?~\$" תתפוס לגבי כל ערך של "C" בין 0x20 ל-0x7E):

```

$~C2740310433063552
${eC2740310433063552
${tC2740310433063552

```



```

C:\Windows\system32\cmd.exe
C:\Users\Alien\Downloads\All4Stages>stage2.exe
Enter password $~C2740310433063552
Congratulations! you have solved this challenge!
To get the next challenge send an e-mail to the following address: vXRuqwUm3PC@
mail.com .
To get the points for the raffle you must explain in the mail how you solved thi
s challenge.
C:\Users\Alien\Downloads\All4Stages>stage2.exe
Enter password ${eC2740310433063552
Congratulations! you have solved this challenge!
To get the next challenge send an e-mail to the following address: vXRuqwUm3PC@
mail.com .
To get the points for the raffle you must explain in the mail how you solved thi
s challenge.
C:\Users\Alien\Downloads\All4Stages>stage2.exe
Enter password ${tC2740310433063552
Congratulations! you have solved this challenge!
To get the next challenge send an e-mail to the following address: vXRuqwUm3PC@
mail.com .
To get the points for the raffle you must explain in the mail how you solved thi
s challenge.
C:\Users\Alien\Downloads\All4Stages>

```



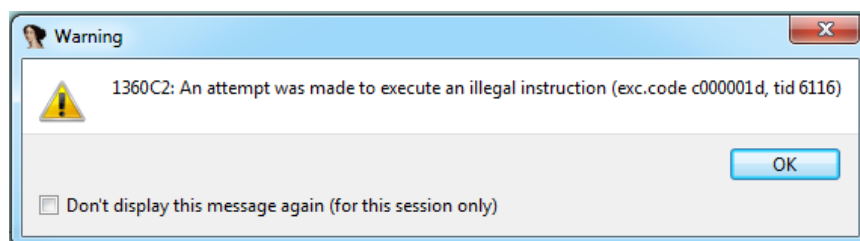
אתגר 4 - stage3.exe:

ידע דרוש לפתרון

- Exception Handling
- תהליך קימפול וצורת קימפול של Visual Studio
- Thread-ים ושליטה ב-Thread-ים ב-IDA
- x86 Assembly
- Windows API
- Exception Anti-Debugging
- Timing Anti-Debugging
- General Anti-Debugging
- Dynamic Function Pointers

איך להיאבד בדרכך

נתחיל מלשים BP בתחילת הפונקציה לה-IDA קורא main_ ונריץ את הקובץ, נראה כי IDA אינו מגיע בכלל לפונקציה וזורק Exception הרבה לפני.

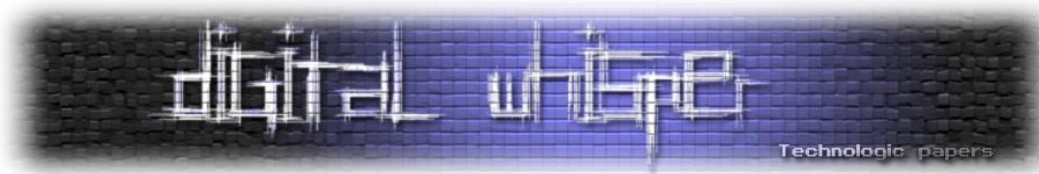


אכן כשנגיע ל-Instruction נראה כי הוא אינו תקין:

```
.text:001360B3 loc_1360B3: ; CODE XREF: .text:0013609F↑j
.text:001360B3 ; .text:001360AB↑j ...
.text:001360B3 movzx ecx, byte ptr [ebp-10h]
.text:001360B7 cmp ecx, 1
.text:001360BA jnz short loc_1360BD
.text:001360BA ;
.text:001360BC db 2Dh
.text:001360BD ;
.text:001360BD loc_1360BD: ; CODE XREF: .text:001360BA↑j
.text:001360BD mov edx, [ebp-8]
.text:001360BD ;
.text:001360C0 db 89h
.text:001360C1 db 55h ; U
.text:001360C2 ;
.text:001360C2 lock jmp short sub_1360CE
.text:001360C5 ;
.text:001360C5 ; START OF FUNCTION CHUNK FOR sub_1360CE
.text:001360C5 loc_1360C5: ; CODE XREF: sub_1360CE+26↓j
.text:001360C5 mov eax, [ebp-10h]
.text:001360C8 add eax, 1
.text:001360CB mov [ebp-10h], eax
.text:001360CB ; END OF FUNCTION CHUNK FOR sub_1360CE
.text:001360CE
```

אתגרי ה-CrackMe של רפאל 2015

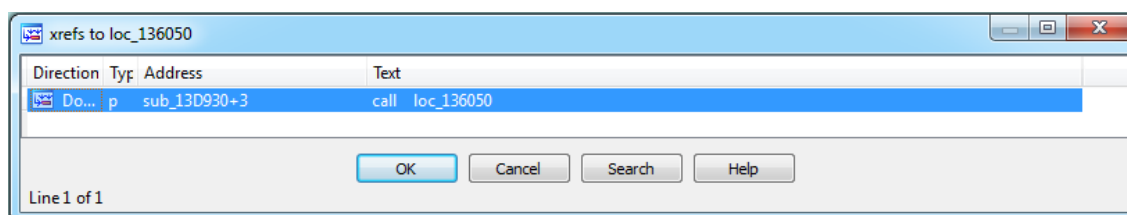
www.DigitalWhisper.co.il

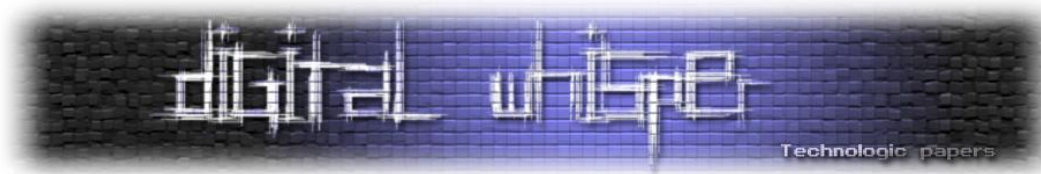


נתחיל בלהבין איך הגענו לקטע הקוד (loc_1360BD) הזה בכלל. נבצע XREF-ים עד שנבין איזה הסתעפות מביאה אותנו ל-Instruction הלא תקין.

```
.text:00136050 loc_136050: ; CODE XREF: sub_13D930+31p
.text:00136050 push    ebp
.text:00136051 mov     ebp, esp
.text:00136053 sub     esp, 20h
.text:00136056 mov     dword ptr [ebp-8], 0
.text:0013605D mov     dword ptr [ebp-14h], 0
.text:00136064 push    offset aNtQueryInforma ; "NtQueryInformationProcess"
.text:00136069 push    offset ModuleName       ; "ntdll.dll"
.text:0013606E call    ds:GetModuleHandleA
.text:00136074 push    eax
.text:00136075 call    ds:GetProcAddress
.text:00136078 mov     [ebp-1Ch], eax
.text:0013607E push    0
.text:00136080 push    4
.text:00136082 lea     eax, [ebp-14h]
.text:00136085 push    eax
.text:00136086 push    1Fh
.text:00136088 call    ds:GetCurrentProcess
.text:0013608E push    eax
.text:0013608F call    dword ptr [ebp-1Ch]
.text:00136092 mov     [ebp-18h], eax
.text:00136095 cmp     dword ptr [ebp-18h], 0
.text:00136099 jz      short loc_1360A1
.text:0013609B mov     byte ptr [ebp-10h], 0
.text:0013609F jmp     | short loc_1360B3
.text:001360A1 ;
.text:001360A1 loc_1360A1: ; CODE XREF: .text:00136099fj
.text:001360A1 cmp     dword ptr [ebp-14h], 0
.text:001360A5 jnz     short loc_1360AF
.text:001360A7 mov     byte ptr [ebp-10h], 1
.text:001360AB jmp     short loc_1360B3
.text:001360AD jmp     short loc_1360B3
.text:001360AF ;
.text:001360AF loc_1360AF: ; CODE XREF: .text:001360A5fj
.text:001360AF mov     byte ptr [ebp-10h], 0
.text:001360B3 loc_1360B3: ; CODE XREF: .text:0013609Ffj
; .text:001360ABfj ...
.text:001360B3 movzx   ecx, byte ptr [ebp-10h]
.text:001360B7 cmp     ecx, 1
.text:001360BA jnz     short loc_1360BD
.text:001360BA ;
.text:001360BC db      20h
.text:001360BD ;
.text:001360BD loc_1360BD: ; CODE XREF: .text:001360BAfj
.text:001360BD mov     eax, [ebp-8]
.text:001360BD ;
.text:001360C0 db      89h
.text:001360C1 db      55h ; U
.text:001360C2 ;
.text:001360C2 lock jmp short sub_1360CE
```

נראה שהגענו ל-Instruction הזה מהפונקציה loc_136050. יש להבין עכשיו איך הגענו אליה, בשביל זה נבצע XREF לראות מאיפה היא נקראה.





לפונקציה sub_13D930 נבצע גם כן XREF, נגיע לטבלת פונקציות שנראית ככה:

rdata:0013E16C	dd offset	_pre_cpp_init
rdata:0013E170	dd offset	sub_13D9FF
rdata:0013E174	dd offset	sub_13DA2D
rdata:0013E178	dd offset	sub_13DA43
rdata:0013E17C	dd offset	sub_13DA15
rdata:0013E180	dd offset	sub_13DA21
rdata:0013E184	dd offset	sub_13D9DB
rdata:0013E188	dd offset	sub_13D9E7
rdata:0013E18C	dd offset	sub_13D9F3
rdata:0013E190	dd offset	sub_13D8D0
rdata:0013E194	dd offset	sub_13D8F0
rdata:0013E198	dd offset	sub_13D910
rdata:0013E19C	dd offset	sub_13D930
rdata:0013E1A0	dd offset	sub_13D940
rdata:0013E1A4	dd offset	sub_13D950
rdata:0013E1A8	dd offset	sub_13D970
rdata:0013E1AC	dd offset	sub_13D990
rdata:0013E1B0	dd offset	sub_13D9D0

טבלת הפונקציות הזאת היא חלק מרוטינת האיתחול של קוד המקומפל ב-Visual Studio הממולאת בידי המשתמש ע"י פונקציות שהוא מגדיר כפונקציות אתחול הצריכות לרוץ לפני הקוד העיקרי. הטבלה נקראת מהפונקציה `_initterm`⁶.

אם אנחנו יוצאים מנקודת הנחה שאיננו מכירים את התנהגות זאת יש שתי דרכים להבין מאיפה הפונקציות נקראות:

1. הנחה של BP בתוך הפונקציה הראשונה בטבלה, הגעה ל-Entry Point של הקובץ (CTRL+E), הנחת BP, הרצה אל ה-BP בתחילת הקובץ, סימון Toggle Function Tracing והרצה עד שניתקל ב-BP השני.

במעבר למסך ה-Trace יתקבל משהו בסגנון הזה:

00001D3C	.text:security_init_cookie+37	call	ds:GetSystemTimeAsFileTime	__security_init_cookie call kernel32.dll:kernel32_GetSystemTimeAsFileTime
00001D3C	.text:security_init_cookie+43	call	ds:GetCurrentProcessId	__security_init_cookie call kernel32.dll:kernel32_GetCurrentProcessId
00001D3C	.text:security_init_cookie+48	call	ds:GetCurrentThreadId	__security_init_cookie call kernel32.dll:kernel32_GetCurrentThreadId
00001D3C	.text:security_init_cookie+53	call	ds:TickCount	__security_init_cookie call kernel32.dll:kernel32_GetTickCount
00001D3C	.text:security_init_cookie+5F	call	ds:QueryPerformanceCounter	__security_init_cookie call kernel32.dll:kernel32_QueryPerformanceCounter
00001D3C	.text:security_init_cookie+9A	retn		__security_init_cookie returned to start+5
00001D3C	.text:tmainCRTStartup+7	call	__SEH_prolog4	__tmainCRTStartup call __SEH_prolog4
00001D3C	.text:__SEH_prolog4+44	retn		__SEH_prolog4 returned to __tmainCRTStartup+C
00001D3C	.text:tmainCRTStartup+1B	call	ds:HeapSetInformation	__tmainCRTStartup call kernel32.dll:kernel32_HeapSetInformation
00001D3C	.text:tmainCRTStartup+38	call	ds:InterlockedCompareExchange	__tmainCRTStartup call kernel32.dll:kernel32_InterlockedCompareExchange
00001D3C	.text:tmainCRTStartup+8A	call	_initterm_e	__tmainCRTStartup call _initterm_e
00001D3C	.text:tmainCRTStartup+BF	call	_initterm	__tmainCRTStartup call _initterm

ניתן לראות בבירור את הקריאה האחרונה ל-`_initterm`.

2. הנחה של BP בתוך הפונקציה הראשונה בטבלה וביצוע RUR מיד ננחת בקטע קוד המסומן ע"י IDA כקטע קוד של `_initterm`.

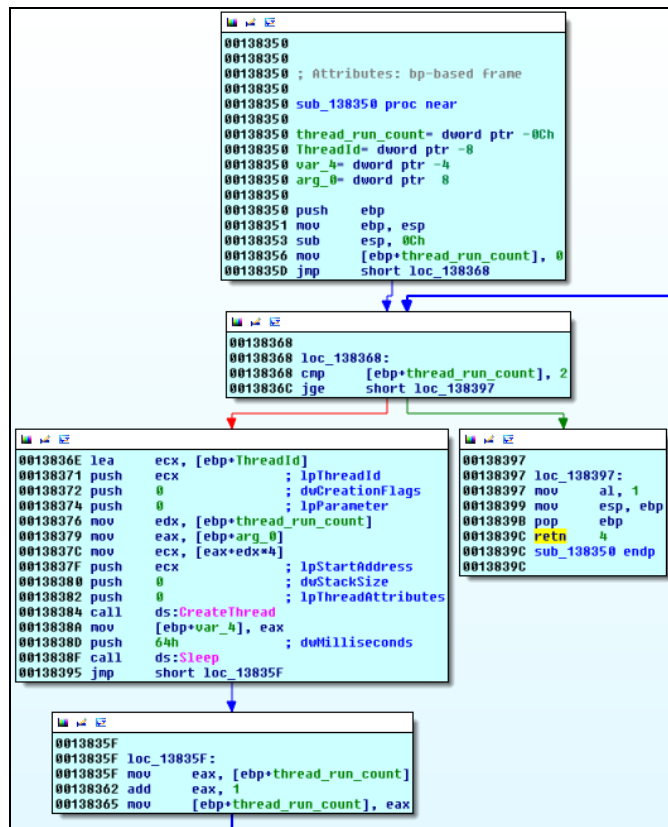
msvcrt100.dll:71052630	loc_71052630:		; CODE XREF: msvcrt100.dll:msvcrt100__initterm+191j
msvcrt100.dll:71052630	cmp	esi, [ebp+0Ch]	
msvcrt100.dll:71052633	jnb	short loc_71052642	
msvcrt100.dll:71052635	mov	eax, [esi]	
msvcrt100.dll:71052637	test	eax, eax	
msvcrt100.dll:71052639	jz	short loc_7105263D	
msvcrt100.dll:7105263B	call	eax	
msvcrt100.dll:7105263D			
msvcrt100.dll:7105263D	loc_7105263D:		; CODE XREF: msvcrt100.dll:msvcrt100__initterm+121j
msvcrt100.dll:7105263D	add	esi, 4	
msvcrt100.dll:71052640	jmp	short loc_71052630	
msvcrt100.dll:71052642			
msvcrt100.dll:71052642			
msvcrt100.dll:71052642	loc_71052642:		; CODE XREF: msvcrt100.dll:msvcrt100__initterm+C1fj
msvcrt100.dll:71052642	pop	esi	
msvcrt100.dll:71052643	pop	ebp	
msvcrt100.dll:71052644	retn		

⁶ <http://www.codeguru.com/cpp/misc/misc/applicationcontrol/article.php/c6945/Running-Code-Before-and-After-Main.htm>

איך לא להיאבד בדרך - שרשרת האתחול

פונקציית אתחול 1 (sub_136100):

אז אחלה, הבנו איפה נפלנו ואיך הגענו לנפילה. עכשיו אפשר להניח BP-ים בכל הפונקציות הנמצאות בטבלת הפונקציות ולהתחיל להריץ. נריץ כל פונקציה וננסה להבין האם היא רלוונטית בשבילנו, אפשר להבין כי הפונקציה הרלוונטית הראשונה בשבילנו היא פונקציה sub_13D8F0 בתוכה נמצאת קריאה לפונקציה נוספת, אם נעקוב אחריה נגיע לפונקציה הבאה:



הפונקציה מקבלת מערך של מצביעים לפונקציות ומריצה את 2 הפונקציות הראשונות במערך. ניתן לחזור לפונקציה הקודמת ולעקוב אחרי Offset ששנחף למחסנית (off_142E98) בשביל למצוא את המערך המועבר.

```

.data:00142E98 off_142E98 dd offset sub_136100 ; DATA XREF: sub_13D8F0+3f0
.data:00142E9C dd offset loc_134E40
  
```

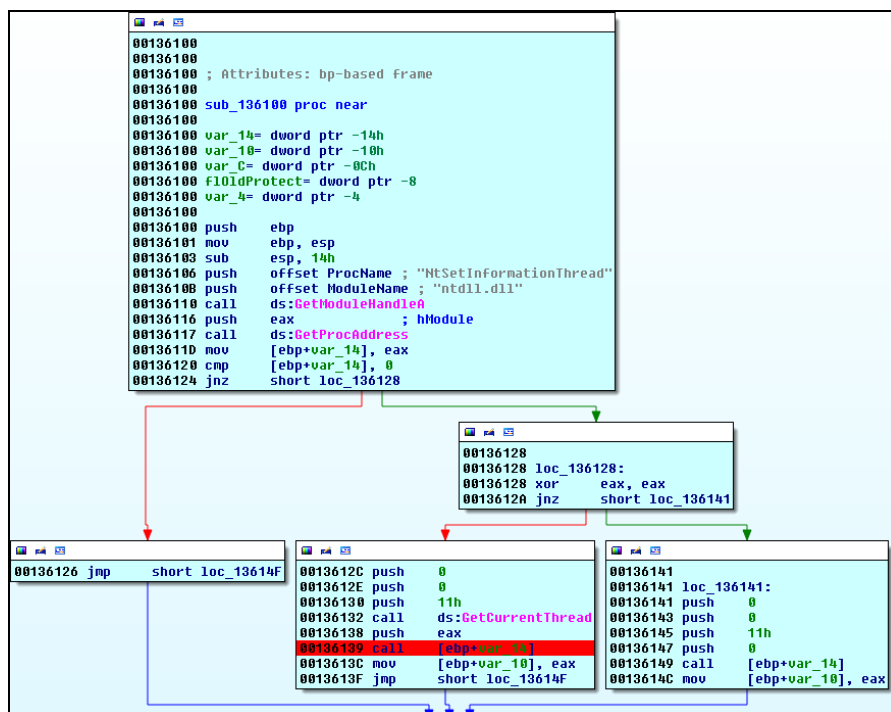
לשם שליטה ב-Thread-ים שיווצרו ע"י הקוד נניח BP בתחילת כל Thread חדש שיווצר וכאשר ייווצר ה-Thread נעבור אליו ונבצע השהייה (Suspend) ל-Thread העיקרי ככה שלא יברח לנו ויאתחל עוד Thread-ים מה שעלול לשלוח אותנו לארץ האבדון בה Thread-ים אינם נשלטים ע"י IDA.

Threads		
Decimal	Hex	State
5884	16FC	Suspended
2992	BB0	Ready

אתגרי ה CrackMe של רפאל 2015

www.DigitalWhisper.co.il

לצורך העניין התמונה הנ"ל ממחישה את העניין, ה-Thread העיקרי הרץ מ-sub_138350, בעל הערך 5884, וה-Thread שאני מבצע לו Debug כרגע והוא רץ מהפונקציה sub_136100 בעל הערך 2992. ניכס לפונקציה הראשונה (sub_136100):



הפונקציה מתחילה בקבלת הכתובת של הפונקציה [NtSetInformationThread](#) ע"י קריאה ל-[GetProcAddress](#). לאחר מכן אם [GetProcAddress](#) החזירה תשובה חיובית הפונקציה [NtSetInformationThread](#) תיקרא בצורה הבאה:

```
NtSetInformationThread(GetCurrentThread(), 0x11, 0, 0);
```

מתוך [ntnative.h](#):

```

typedef enum THREADINFOCLASS { //Query Set
    ThreadBasicInformation, // 0 Y N
    ThreadTimes, // 1 Y N
    ThreadPriority, // 2 N Y
    ThreadBasePriority, // 3 N Y
    ThreadAffinityMask, // 4 N Y
    ThreadImpersonationToken, // 5 N Y
    ThreadDescriptorTableEntry, // 6 Y N
    ThreadEnableAlignmentFaultFixup, // 7 N Y
    ThreadEventPair, // 8 N Y
    ThreadQuerySetWin32StartAddress, // 9 Y Y
    ThreadZeroTlsCell, // 10 N Y
    ThreadPerformanceCount, // 11 Y N
    ThreadAmILastThread, // 12 Y N
    ThreadIdealProcessor, // 13 N Y
    ThreadPriorityBoost, // 14 Y Y
    ThreadSetTlsArrayAddress, // 15 N Y
    ThreadIsIoPending, // 16 Y N
    ThreadHideFromDebugger, // 17 N Y
    ThreadIsCriticalInformation // 18 Y Y //RtlSetThreadIsCritical, sizeof = 4, XP
} THREADINFOCLASS;
    
```

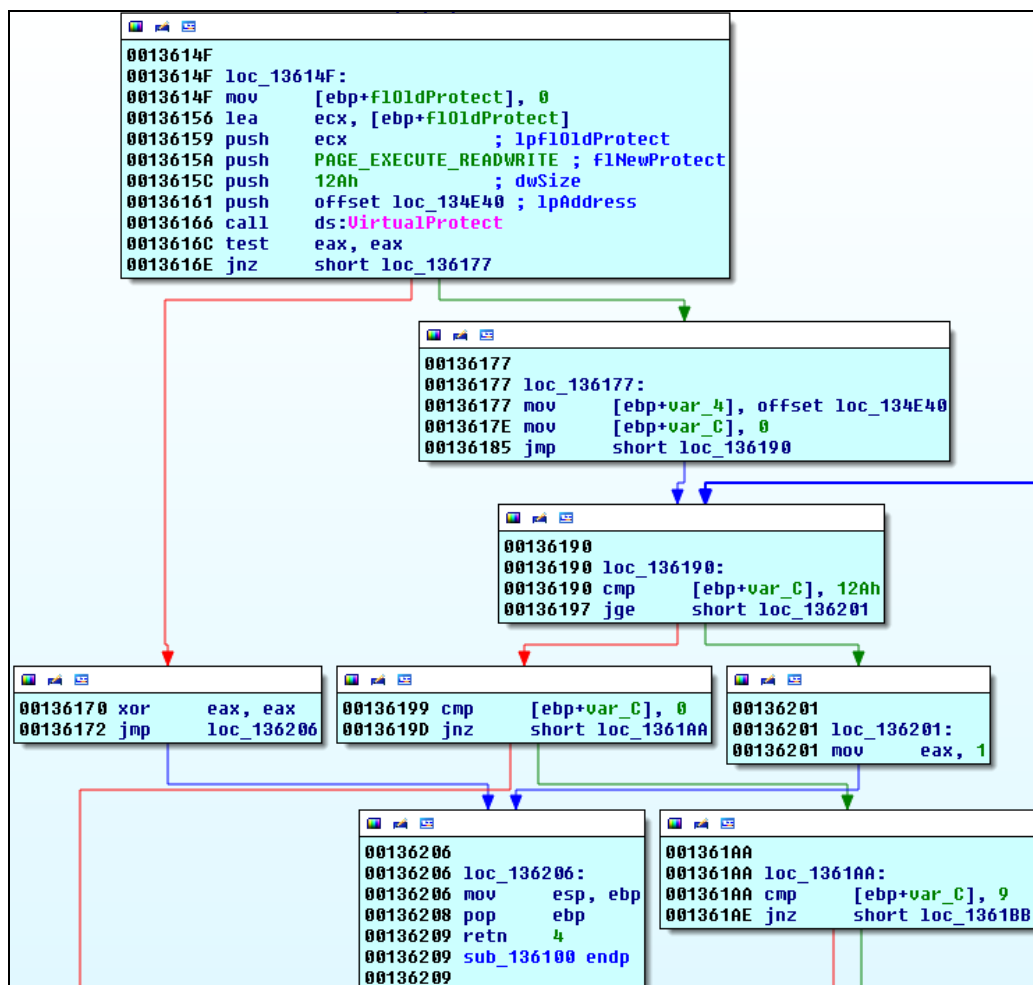
אתגרי ה-CrackMe של רפאל 2015

www.DigitalWhisper.co.il

כשנמיר את הפרמטר 0x11 עפ"י ה-Enum הנתון, נקבל שהפרמטר המועבר הוא ThreadHideFromDebugger (0x11 = 17) מה שהפונקציה תעשה כאשר היא מקבלת פרמטר כזה הוא ניתוק ה-Context של ה-Thread מה-Debugger ככה שה-Thread ירוץ אך לא נוכל לעשות איתו כל מה ש-Debugger מאפשר לנו (למשל עצירה, שינוי מידע, התבוננות באוגרים וכו') וה-Debugger לא יכיר ב-Thread ולא יראה אותו ברשימת ה-Thread-ים. בשביל שה-Thread לא ינותק ניתן לשנות את הערך המוכנס ל-Stack מ-0x11 ל-0x2 למשל. ה-Stack שלנו יראה כך:

007EF7DC	FFFFFFFE
007EF7E0	00000002
007EF7E4	00000000
007EF7E8	00000000

בצורה כזאת ה-Thread לא יתנתק מה-Debugger. בהמשך נבצע את אותה הפעולה בכל פעם שתיהיה קריאה לפונקציה [NtSetInformationThread](#) עם הפרמטר 0x11. לאחר שנפרדנו מה-Anti-debugging הראשון שהקוד הציע נמשיך עם ריצת הקוד ונגיע לקריאה ל-[VirtualProtect](#) שנראית ככה:



הפרמטר הראשון ש-[VirtualProtect](#) מקבלת הוא פרמטר של כתובת בזיכרון אותה הפונקציה תשנה, הכתובת שמועברת היא הכתובת של הפונקציה השניה שמועברת במערך הפונקציות לפונקציה sub_138350 שנמצאת ב-loc_134E40.

הפרמטר השלישי אשר מועבר ל-[VirtualProtect](#) הוא הקבוע 0x40 (המתפרש כ- PAGE_EXECUTE_READWRITE) בפועל מה שקורה הוא שהקוד משנה את ההגנה על מקטע הזיכרון ממה שיש לו כרגע ל- RWX(execute, read, write) (התנהגות זאת נפוצה בקרב Packer-ים וקוד אשר משנה את עצמו מכיוון ש-Section-ים של מידע של PE לרוב מתקמפלים עם הרשאות R או RW על ה-Section לעומת - Section-ים של קוד אשר מתקמפלים לרוב עם הרשאות RX) אפשר לנחש כי קטע הקוד מפה והלאה מבצע פיענוח של הפונקציה בשביל שיוכל להריץ אותה ב-Thread הבא.

נשחרר את ה-Thread לדרכו ואכן נראה כי הוא מבצע פיענוח לקוד שנמצא בכתובת המועברת בפרמטר ל-[VirtualProtect](#).

<pre>.text:01104E40 loc_1104E40: .text:01104E40 .text:01104E40 and cl, [ebx+6A006AECh] .text:01104E46 add [edx+0], ebp .text:01104E49 adc [edi+88h], esi .text:01104E4C loopne near ptr loc_1104E5D+1 .text:01104E4E add Handles[ebx], esp .text:01104E54 int 3 .text:01104E55 sub eax, 1113564h .text:01104E5A rol byte ptr [esi+10h], 1 .text:01104E5D loc_1104E5D: .text:01104E5D add esp, ecx .text:01104E5F sub eax, 1113534h .text:01104E64 jo short near ptr loc_1104EAF+3 .text:01104E66 adc [ecx], al .text:01104E68 int 3 .text:01104E69 sub eax, 11135D0h .text:01104E6E jo short loc_1104EBF .text:01104E70 adc [ecx], al .text:01104E72 int 3 .text:01104E73 sub eax, 1113540h .text:01104E78 ror byte ptr [edx+10h], 1 .text:01104E7B add esp, ecx .text:01104E7D sub eax, 111357Ch .text:01104E82 inc eax .text:01104E83 inc ebp .text:01104E84 adc [ecx], al .text:01104E86 int 3 .text:01104E87 sub eax, 1113544h .text:01104E8C lock push esi .text:01104E8E adc [ecx], al .text:01104E90 int 3 .text:01104E91 sub eax, 1113600h .text:01104E96 inc eax .text:01104E97 push esi .text:01104E98 adc [ecx], al .text:01104E9A int 3 .text:01104E9B sub eax, 11135F8h</pre>	<pre>.text:01104E40 loc_1104E40: .text:01104E40 .text:01104E40 push ebp .text:01104E41 mov ebp, esp .text:01104E43 push 0 .text:01104E45 push 1 .text:01104E47 push 0 .text:01104E49 call ds:CreateMutexA .text:01104E4F mov Handles, eax .text:01104E54 mov dword_1113564, offset sub_11046D0 .text:01104E5E mov dword_1113534, offset sub_1104C70 .text:01104E68 mov dword_11135D0, offset sub_1104F70 .text:01104E72 mov dword_1113540, offset sub_1104AD0 .text:01104E7C mov dword_111357C, offset sub_1104540 .text:01104E86 mov dword_1113544, offset sub_11056F0 .text:01104E90 mov dword_1113600, offset sub_1105640 .text:01104E9A mov dword_11135F8, offset sub_1102B20 .text:01104EA4 mov dword_1113584, offset sub_11043D0 .text:01104EAE mov dword_1113570, offset sub_11041A0 .text:01104EB8 mov dword_11135F0, offset sub_1105750 .text:01104EC2 mov dword_111354C, offset sub_1103A50 .text:01104EC8 mov dword_1113560, offset sub_1104000 .text:01104ED6 mov dword_1113554, offset sub_1103900 .text:01104EE0 mov dword_111355C, offset sub_1102DC0 .text:01104EEA mov dword_1113530, offset sub_1103E10 .text:01104EF4 mov dword_1113558, offset sub_1102B80 .text:01104EFE mov dword_1113568, offset sub_1103600 .text:01104F08 mov dword_1113574, offset sub_11033C0 .text:01104F12 mov dword_111353C, offset sub_1105120 .text:01104F1C mov dword_1113594, offset sub_1102930 .text:01104F26 mov dword_1113580, offset sub_1105000 .text:01104F30 mov dword_1113578, offset sub_1103530 .text:01104F3A mov dword_1113588, offset sub_1102A70 .text:01104F44 mov dword_11135C8, offset sub_11044E0 .text:01104F4E mov dword_1113590, offset sub_11028B0 .text:01104F58 mov eax, Handles .text:01104F5D push eax .text:01104F5E call ds:ReleaseMutex .text:01104F64 xor eax, eax</pre>
--	--

אחרי

תכלית הפונקציה היא אתחול מערך של פונקציות, לפני ההתחלה היא קוראת ל-[CreateMutexA](#) ובסיום ל-[ReleaseMutex](#), את הסיבה נבין רק בהמשך הניתוח אבל אפשר להסיק שזאת דרך לסנכרן את הקוד. ניתן לקוד לרוץ עד שיעצר ב-BP הבא הנמצא בפונקציה הבאה (sub_13D910) בטבלת פונקציות האתחול של _initterm.



פונקציית אתחול 2 (sub_13D910):

הפונקציה מבצעת את אותן הפעולות כמו פונקציית האתחול הראשונה עם שוני בפונקציה השניה שמועברת אליה בפרמטרים אך תכליתה זהה לקודמת לה - והיא השלמת אתחול אותו מערך של פונקציות.

פונקציית אתחול 3 (sub_13D930):

פונקציה זאת קוראת ל-loc_136050 הנראית ככה:

```
.text:00136050 loc_136050: ; CODE XREF: sub_13D930+31p
.text:00136050 push ebp
.text:00136051 mov ebp, esp
.text:00136053 sub esp, 20h
.text:00136056 mov dword ptr [ebp-8], 0
.text:0013605D mov dword ptr [ebp-14h], 0
.text:00136064 push offset aNtQueryInforma ; "NtQueryInformationProcess"
.text:00136069 push offset ModuleName ; "ntdll.dll"
.text:0013606E call ds:GetModuleHandle@
.text:00136074 push eax
.text:00136075 call ds:GetProcAddress
.text:0013607B mov [ebp-1Ch], eax
.text:0013607E push 0
.text:00136080 push 4
.text:00136082 lea eax, [ebp-14h]
.text:00136085 push eax
.text:00136086 push 1Fh
.text:00136088 call ds:GetCurrentProcess
.text:0013608E push eax
.text:0013608F call dword ptr [ebp-1Ch]
.text:00136092 mov [ebp-18h], eax
.text:00136095 cmp dword ptr [ebp-18h], 0
.text:00136099 jz short loc_1360A1
.text:0013609B mov byte ptr [ebp-10h], 0
.text:0013609F jmp short loc_1360B3
.text:001360A1 ;
.text:001360A1 loc_1360A1: ; CODE XREF: .text:00136099fj
.text:001360A1 cmp dword ptr [ebp-14h], 0
.text:001360A5 jnz short loc_1360AF
.text:001360A7 mov byte ptr [ebp-10h], 1
.text:001360AB jmp short loc_1360B3
.text:001360AD ;
.text:001360AD jmp short loc_1360B3
.text:001360AF ;
.text:001360AF loc_1360AF: ; CODE XREF: .text:001360A5fj
.text:001360AF mov byte ptr [ebp-10h], 0
.text:001360B3 ;
.text:001360B3 loc_1360B3: ; CODE XREF: .text:0013609Ffj
.text:001360B3 ; .text:001360ABfj ...
.text:001360B3 movzx ecx, byte ptr [ebp-10h]
.text:001360B7 cmp ecx, 1
.text:001360BA jnz short loc_1360BD
```

נראה מוכר? הרי זאת הפונקציה שגרמה לנו ל-Exception בקוד שאחריו לא יכולנו להמשיך בריצה הראשונה. הפונקצייה מוצאת את כתובת הפונקציה [NtQueryInformationProcess](#) ולאחר מכן קוראת לה בצורה הבאה:

```
NtQueryInformationProcess(GetCurrentProcess(), 0x1F, &out_buffer, 4, 0);
```

ה-Enum המועבר לפונקציה אינו מתועד ע"י Microsoft ולא נמצא ב-[ntnative.h](#), אבל אפשר למצוא אותו בכל מני מדריכים על איך לבצע Anti-debugging⁷. ה-Enum נקרא **ProcessDebugFlags** וכאשר הוא מועבר לפונקציה היא מתשאלת את [EPROCESS!NoDebugInherit](#) ומחזירה את ערכו. נריץ את הקוד עד

⁷ <http://www.codeproject.com/Articles/30815/An-Anti-Reverse-Engineering-Guide#NtQueryObject>



אחרי הקריאה לפונקציה ונשנה את ערך ההחזר (eax) מ-0 ל-1. נמשיך בהרצה, נראה שהקוד לא יצור Exception ונגיע לקטע קוד אשר משתמש ב-[CreateThread](#).

מנגנון ההגנה:

הדרך בה ממומש מנגנון הגנה במידה ונמצא Debugger היא כזאת:
במידה ונמצא Debugger תתבצע קפיצה לכתובת 0x001360C2 שממנה לא תתבצע ריצה תקינה מפני ש-assembly של intel x86 עובד בצורה כזאת שכל Instruction הוא בעל אורך משתנה בהתאם ל-Instruction. ריצה תקינה תתבצע מהכתובת 0x001360BD ובריצה זאת הכתובת 0x001360C2 לא תתפרש כתחילתה של פקודה אלא חלק מהפקודה שמתחילה ב-0x11360C0 (mov [ebp-0x10], edx) ובאורך 3-Byte לכן כאשר תתבצע קפיצה ל-0x001360C2 המעבד יפרש את המידע הממוקם בכתובת בצורת Instruction שאי אפשר לבצע ולכן יבצע Exception.

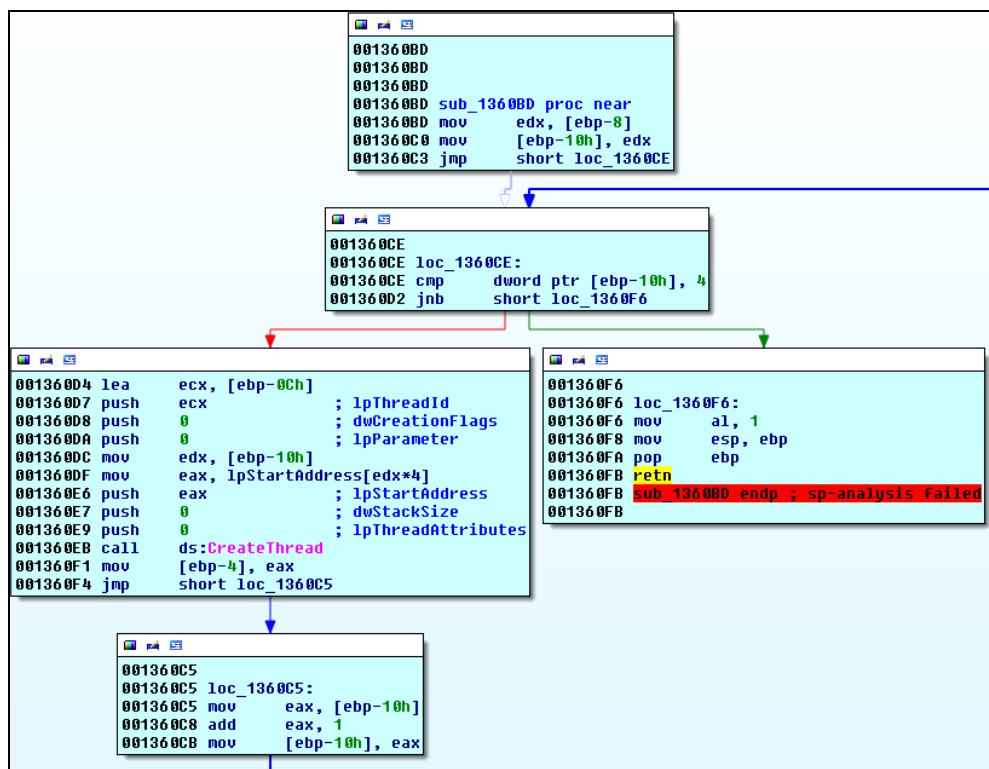
מצד ימין Disassembly תקין, מצד שמאל לא תקין.

```
.text:001360C2 lock jmp short sub_1360CE
```

```
.text:001360BD loc_1360BD:  
.text:001360BD mov     edx, [ebp-8]  
.text:001360C0 mov     [ebp-10h], edx  
.text:001360C3 jmp     short sub_1360CE
```

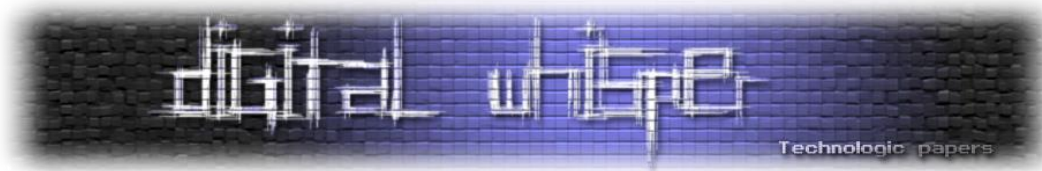
תקין

הפונקציה אליה הגענו בשלב זה נראית ככה:



אתגריה CrackMe של רפאל 2015

www.DigitalWhisper.co.il

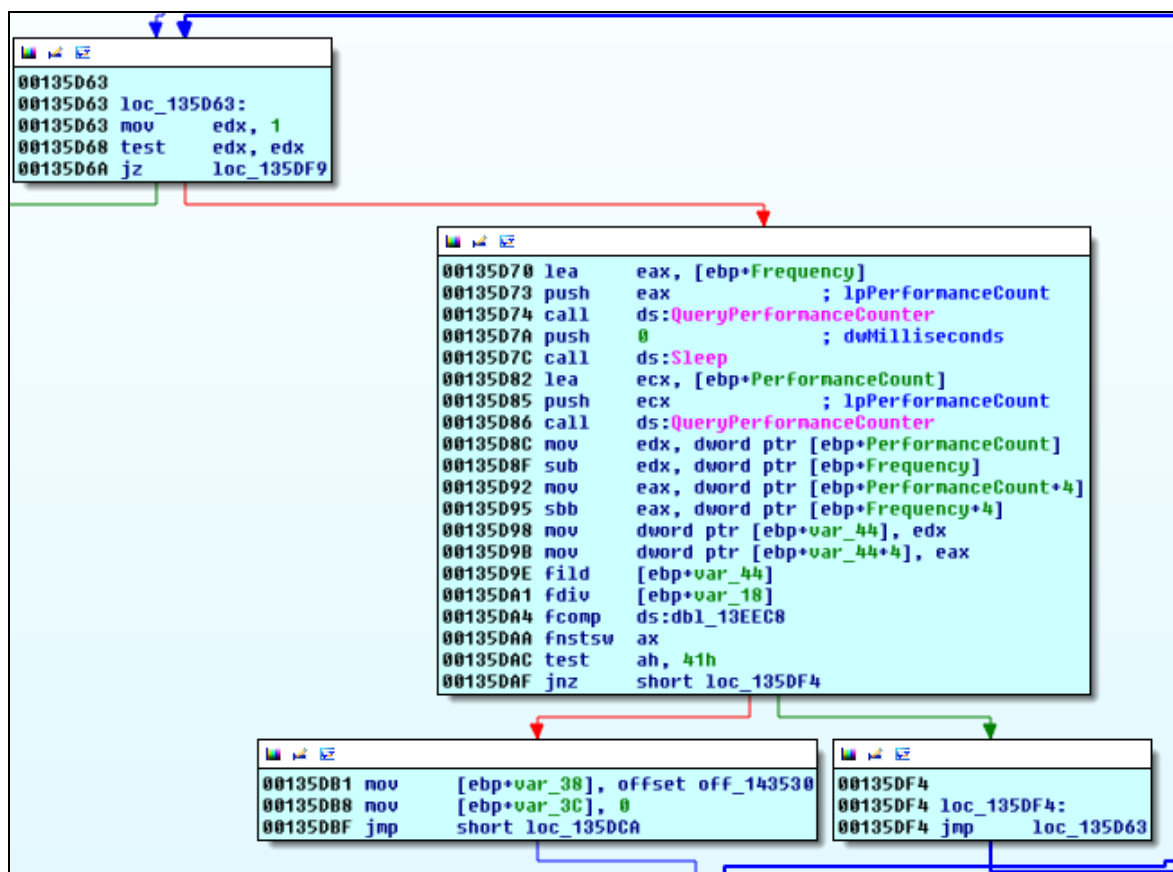


פעם שלישית [CreateThread](#) - גלידה? פחות. הפונקציה תיצור 4-Thread-ים, כל Thread בתבנית קבועה התואמת את ה-Pseudocode הבא:

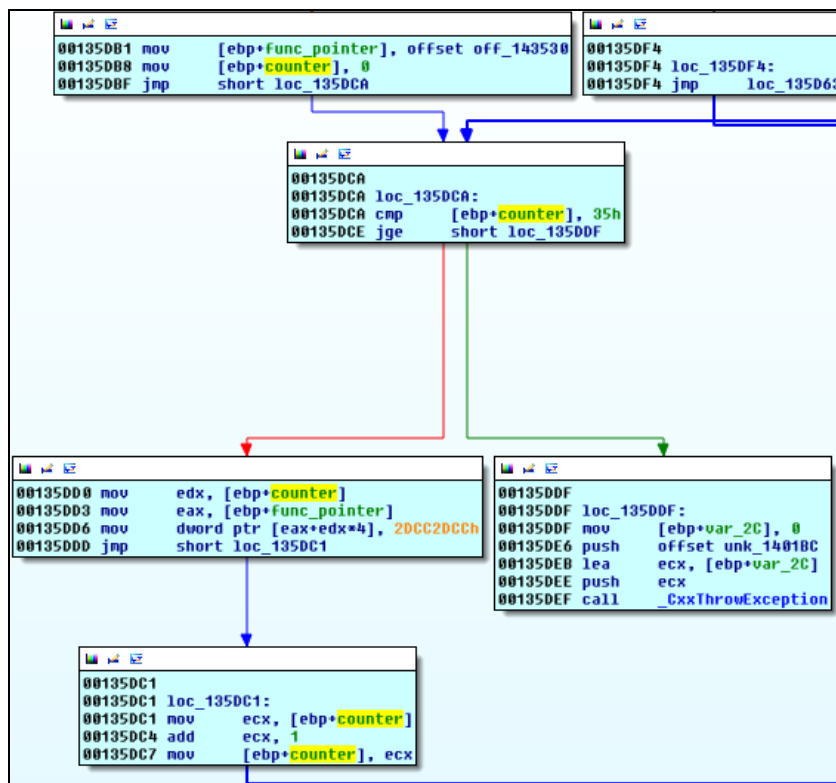
```
NtSetInformationThread(GetCurrentThread(), 0x11, 0, 0);  
while(!isDebugged())  
{  
}  
destroyFunction();
```

כאשר isDebugged הוא אחד מארבעת טכניקות למציאת Debugging עליהן יורחב בהמשך ו-destroyFunc הוא מנגנון ההגנה השני של הקובץ וכאשר אחת מטכניקות מציאת ה-Debugging מדווחת כי נמצא Debugger ה-Thread יעבור למצב בו הוא מפעיל את destroyFunc והורס (ע"י שכתוב) פונקציה הדרושה בשביל לפתור את האתגר.

דוגמא לאיך isDebugged נראה:



destroyFunc(0x143530) הוא תא במערך שאותחל קודם המכיל מצביע לפונקציה - במידה וימצא Debugger הפונקציה הזאת היא שתיהרס):



מה שקורה בפועל אם התוכנית אינה רצה תחת Debugger כל ה-Thread-ים ירוצו בלולאה עד שימצא Debugger, בשביל לעקוף את המנגנון ניתן לבצע Suspend לכל Thread לאחר יצירתו ולהמשיך הלאה.

4 טכניקות ל-`isDebugged`:

- [QueryPerformanceCounter](#) - הפונקציה מחזירה זמן מעבד ברזולוציה גבוהה. בגדול זאת טכניקת השוואת זמנים, הטכניקה תמצא את ה-Debugger רק אם נעצרים בין קריאה אחת לשניה ונותנים לזמן לעבור. לצורך התחמקות מהטכניקה הזאת נרוץ מהקריאה הראשונה עד אחרי השניה ללא עצירה.
- [GetTickCount](#) - הפונקציה מחזירה את הזמן במילישניות מאז הפעלת מערכת ההפעלה. שוב, זאת היא טכניקת השוואת זמנים, והטכניקה תמצא את ה-Debugger רק אם נעצרים בין קריאה אחת לשניה. לצורך התחמקות מהטכניקה הזאת נרוץ מהקריאה הראשונה עד אחרי השניה ללא עצירה.



- [IsDebuggerPresent](#) - הפונקציה מחזירה את ערך ה-[PEB!BeingDebugged](#). לצורך התחמקות מהטכניקה הזאת אפשר לבחור בכל אחת מהדרכים הבאות:

- ✓ לשנות את ערך ה-[PEB!BeingDebugged](#)
- ✓ לשנות את ערך ההחזר (eax) מ-1 ל-0.
- ✓ לשנות את ה-Zero Flag בהסתעפות הקוד הרלוונטית

- [NtQueryInformationProcess](#) - הפונקציה הוזכרה קודם לכן ולא יורחב עליה פה. לצורך התחמקות מהטכניקה הזאת אפשר לבחור בכל אחת מהדרכים הבאות:

- ✓ לשנות את ערך ההחזר (eax) מ-0 ל-1.
- ✓ לשנות את ה-Zero Flag בהסתעפות הקוד הרלוונטית

פונקציית אתחול 4 (sub_13D940):

הפונקציה קוראת ל-sub_132020 אשר נראית ככה (הפונקציה קצת יותר גדולה, רק החלק הרלוונטי נגזר):

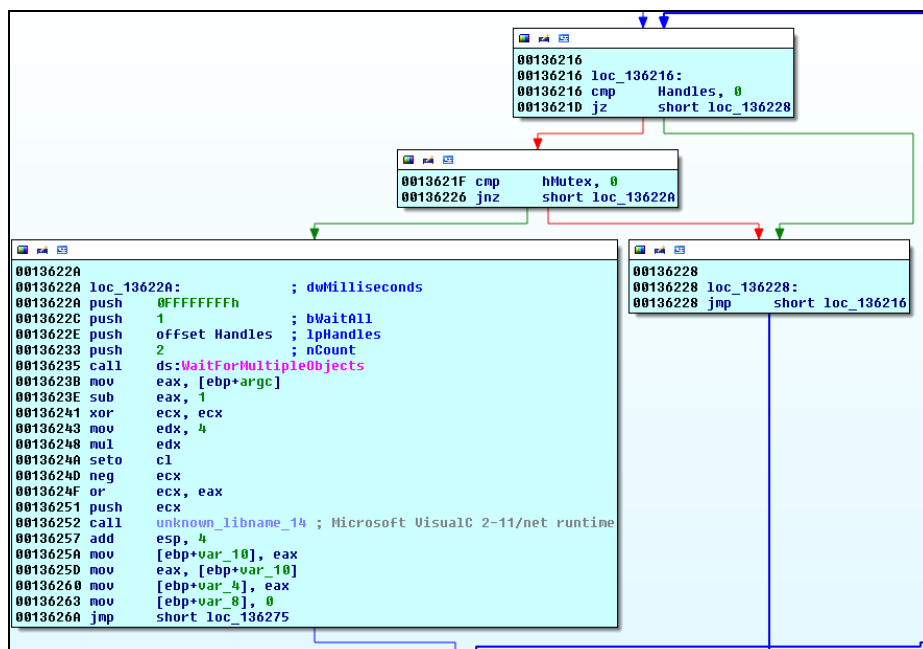
```
0013205C call    ds:GetModuleHandleA
00132062 mov     [ebp+lpAddress], eax
00132065 lea     eax, [ebp+f101dProtect]
00132068 push    eax           ; lpf101dProtect
00132069 push    PAGE_READWRITE ; flNewProtect
0013206B push    1000h         ; dwSize
00132070 mov     ecx, [ebp+lpAddress]
00132073 push    ecx           ; lpAddress
00132074 call    ds:VirtualProtect
0013207A push    1000h         ; Size
0013207F push    0             ; Val
00132081 mov     edx, [ebp+lpAddress]
00132084 push    edx           ; Dst
00132085 call    memset
0013208A add     esp, 0Ch
0013208D mov     al, 1
0013208F jmp     short loc_1320B4
```

מקטע הקוד מקבל את ה-BaseAddress שלו ע"י קריאה ל-[GetModuleHandle](#), מבצע שינוי הרשאות מ-BaseAddress עד BaseAddress+0x1000 להרשאות RW ולאחר מכן רושם למקטע הזיכרון אפסים, בגדול פשוט מוחק את ה-Header [PE](#) וכל מה שאחריו מהזכרון. לא מצאתי סיבה מיוחדת לבצע את הפעולה הזאת.

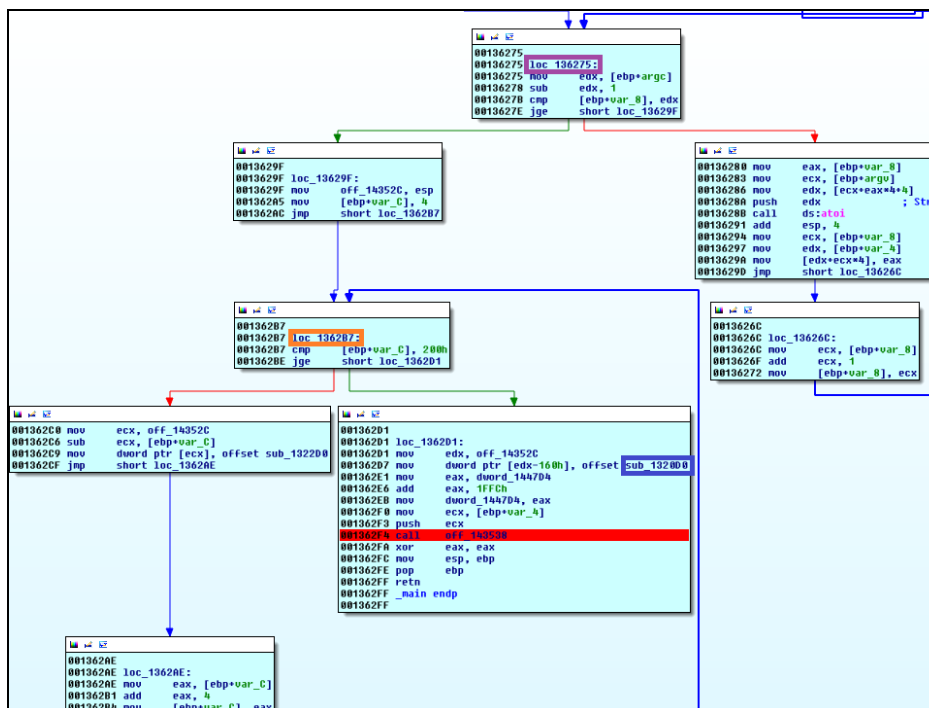
פונקציות האתחול הנותרות אינן רלוונטיות אלינו ולכן מפה אפשר לקפוץ ל-main ולהתחיל לנתח אותה.

Main

תחילת הפונקציה נראית ככה:

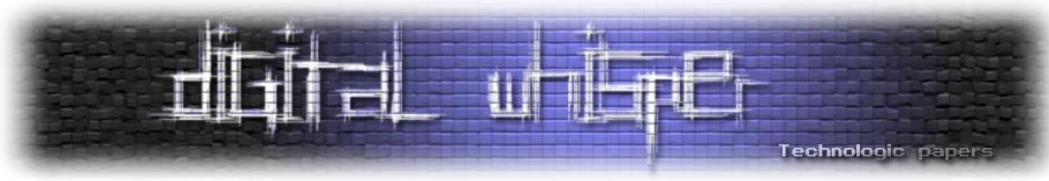


[WaitForMultipleObjects](#) מקבלת מערך של ה-Handle-ים ל-Mutex-ים שנוצרו קודם לכן ומחכה עד שיסגרו, בשלב הזה שניהם אמורים להיות כבר סגורים, תכלית מקטע הקוד הזה הוא לחכות עד שמערך הפונקציות שראינו קודם לכן יהיה מאותחל (ובמילה אחת - סינכרוניזציה). החלק השני של הפונקציה נראה ככה:



אתגרי ה CrackMe של רפאל 2015

www.DigitalWhisper.co.il



נחלק את הפונקציה לשתי לולאות:

loc_136275 - הלולאה בונה מערך של מספרים (בגודל של DWORD) המתקבלים בהעברה דרך [argv](#) בעזרת הפונקציה [atoi](#).

loc_1362B7 - הלולאה בונה מערך (שתמיד יבנה באותה הצורה) בגודל של 0x200, כל תא בגודל של DWORD (128 תאים). את כלל התאים במערך הלולאה מאכלסת באותה הפונקציה (sub_1322D0) בהצצה מהירה - הפונקציה דומה בתכלית שלה ל-destroyFunc (הפונקציה הלוגית שהגדרנו קודם לכן).

בסיום ריצת הלולאה מוכנסת הפונקציה **sub_1320D0** לכתובת (endOfArray - 0x160) (המערך מאותחל מכתובת גבוהה וגודל לכתובות נמוכות), לאחר מכן נדחף מערך המספרים שנוצר ע"י **loc_136275** ויש מעבר למקטע הלוגי האחרון המתחיל עם הפונקציה ב-**off_143538**.

במעבר זריז על **sub_1320D0** ניתן להניח כי בשביל לסיים את האתגר יש להגיע לפונקציה הזאת שכן היא כוללת בתוכה את הפונקציה **puts**, הפונקציה בנוסף קוראת לפונקציה [RemoveVectoredExceptionHandler](#) ניתן להניח שתיהיה קריאה ל-[AddVectoredExceptionHandler](#) לפני הכניסה לפונקציה.

פונקציות ניתוב:

בשלב זה מועברת הריצה לטבלה בת 53 פונקציות, המצביע לאחת מהן הוא ב-**off_143538**:

```
.data:00143530 off_143530 dd offset sub_133E10,offset sub_134C70,offset rf_argv_check_1,offset sub_135120
.data:00143530 ; DATA XREF: sub_1322D0+670
.data:00143530 ; NtQueryInformationProcess_anti_dbg+E770 ...
.data:00143530 dd offset sub_134AD0,offset sub_1356F0,offset sub_133010,offset sub_133A50
.data:00143530 dd offset rf_vectored_exception_handler,offset sub_133900,offset sub_132CA0
.data:00143530 dd offset sub_132DC0,offset sub_134000,offset sub_134600,offset sub_133680
.data:00143530 dd offset sub_135530,offset sub_1341A0,offset sub_1333C0,offset sub_133530
.data:00143530 dd offset sub_134540,offset sub_135000,offset sub_134300,offset sub_132A70
.data:00143530 dd offset sub_132EC0,offset sub_132880,offset sub_132930,offset sub_1332A0
.data:00143530 dd offset sub_133F50,offset sub_133C20,offset sub_1348D0,offset sub_132590
.data:00143530 dd offset sub_134650,offset sub_135A60,offset sub_133710,offset sub_1354A0
.data:00143530 dd offset sub_1358A0,offset sub_1326E0,offset sub_1342F0,offset sub_1344E0
.data:00143530 dd offset sub_133870,offset sub_134F70,offset sub_134780,offset rf_argv_check_3
.data:00143530 dd offset sub_133130,offset rf_argv_check_2,offset sub_1337C0,offset sub_1353E0
.data:00143530 dd offset sub_134320,offset rf_argv_check_min_1,offset sub_133D60,offset sub_132B20
.data:00143530 dd offset sub_134960,offset sub_135640
```

ע"י מעבר על הפונקציות בצורה ויזואלית ניתן לראות כי הפונקציה הכי שונה היא הפונקציה שנקראת בתמונה rf_vectored_exception_handler (sub_133AE0), ניתן להניח שהיא הפונקציה שתוביל אותנו להגעה לפונקציה המיוחלת **sub_1320D0**.

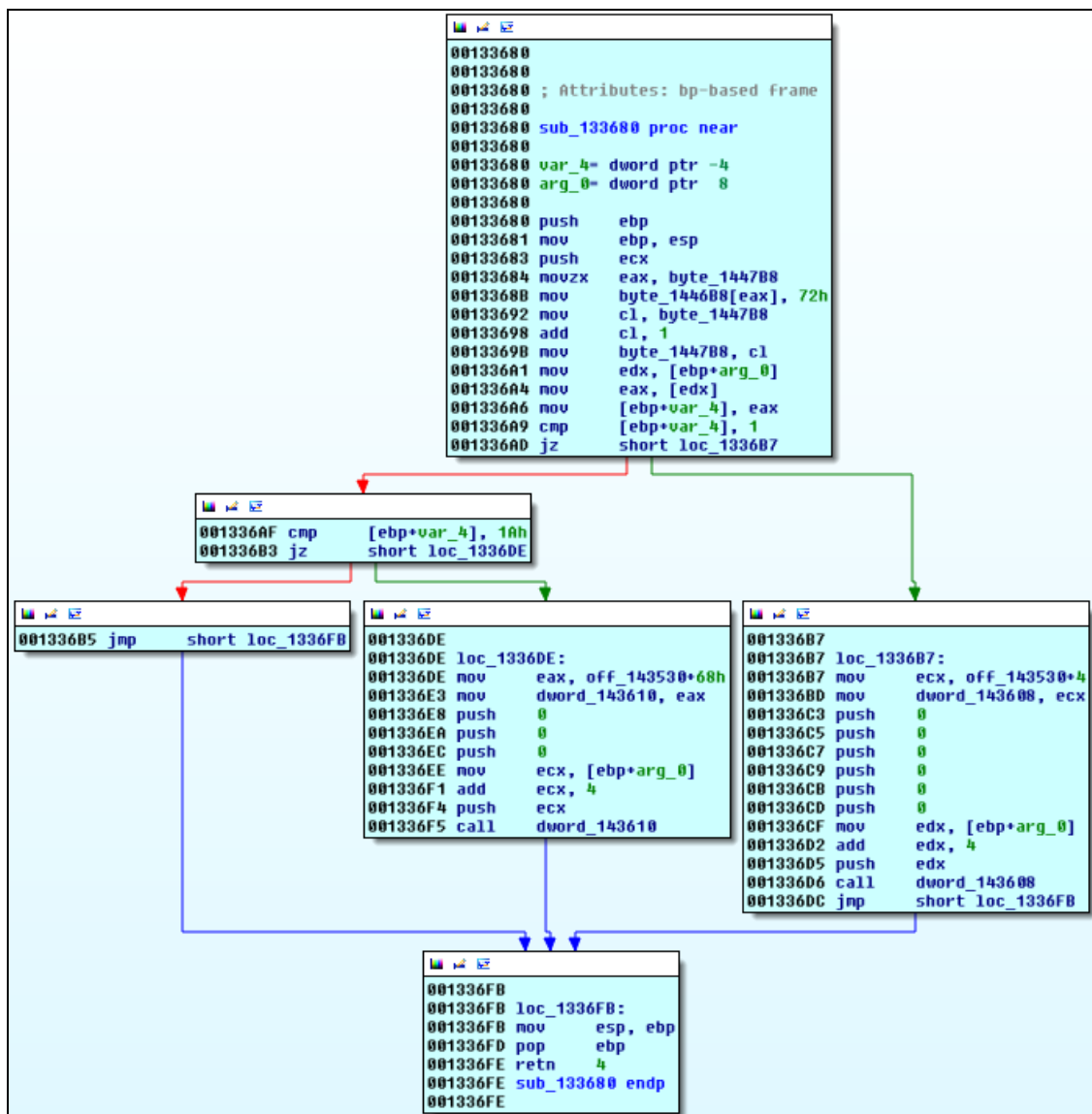
ע"י ניתוח כמה מהפונקציות במערך ניתן להסיק כי תבנית הפונקציות היא קבועה ונראית ככה (בנוסף להסבר האלגוריתמי צירפתי את הפונקציה sub_133680 בה ניתן לראות את האלגוריתם המתואר)

1. קבלת מספר (אותו מספר ממערך המספרים שהוכנס בעזרת [argv](#)).
2. הוספת אות ל-String גלובלי (נמצא ב-0x001446B8) (בפונקציה המדגמית: 0x13368B)

אתגרי ה-CrackMe של רפאל 2015

www.DigitalWhisper.co.il

3. ביצוע מניפולציה על המספר (אופציונלי - מהפונקציות שראיתי המניפולציה מסתכמת בהחסרת מספר קבוע מהמספר שהוכנס) (בפונקציה המדגמית: לא קיים)
4. השוואת המספר מול מספר קבוע או מול Jump table (בפונקציה המדגמית: 0x1336A9)
5. דחיפת המספר הבא במערך המספרים כפרמטר (בפונקציה המדגמית: 0x1336EE/0x1336CF)
6. לפי התוצאה ניתוב לפונקציה הבאה במערך הפונקציות. (בפונקציה המדגמית: 0x1336D6/0x1336F5)



המשימה הבאה שלנו היא לבנות ניתוב בין הפונקציות כך שנגיע מהפונקציה הראשונה עד הפונקציה האחרונה (rf_vectored_exception_handler).

offset	function	byte	xrefs
off_143534	dd offset sub_134C70	0x64	sub_check_-5 sub_1358A0 sub_1341A0
off_14353C	dd offset sub_135120		sub_133D60
off_143544	dd offset sub_1356F0		
off_143548	dd offset sub_133010	0x30	sub_check_-5
off_14354C	dd offset sub_133A50		
off_143554	dd offset sub_133900		sub_133010 sub_1341A0
off_143558	dd offset sub_132CA0	0x36	sub_133010 sub_133D60 sub_132590
off_14355C	dd offset sub_132DC0		sub_1341A0
off_143560	dd offset sub_134000		sub_135A60
off_143564	dd offset sub_1346D0		
off_143568	dd offset sub_133680	0x72	sub_134C70 check_3rd_argv
off_14356C	dd offset sub_135530		sub_133010 sub_133D60
off_143570	dd offset sub_1341A0	0x32	sub_135A60
off_143574	dd offset sub_1333C0		sub_133D60 sub_132590
off_143578	dd offset sub_133530		sub_133010 sub_1358A0
off_14357C	dd offset sub_134540		sub_133D60
off_143580	dd offset sub_135000		sub_133010
off_143584	dd offset sub_1343D0		sub_check_-5 sub_132930 sub_132590
off_143588	dd offset sub_132A70		
off_14358C	dd offset sub_132EC0		sub_132930
off_143590	dd offset sub_1328B0		sub_1341A0
off_143594	dd offset sub_132930	0x58	sub_132CA0
off_143598	dd offset sub_1332A0		
off_1435A0	dd offset sub_133C20		sub_1341A0
off_1435A4	dd offset sub_1348D0		
off_1435A8	dd offset sub_132590	0x73	sub_133D60
off_1435AC	dd offset sub_134650		
off_1435B0	dd offset sub_135A60	0x39	sub_1358A0
off_1435B4	dd offset sub_133710		
off_1435B8	dd offset sub_1354A0		sub_132590
off_1435BC	dd offset sub_1358A0	0x76	sub_132590
off_1435C4	dd offset sub_1342F0		
off_1435C8	dd offset sub_1344E0		
off_1435CC	dd offset sub_133870		sub_132930 sub_135A60
off_1435D0	dd offset sub_134F70		sub_133010
off_1435D4	dd offset check_min_5_argv		
off_1435DC	dd offset sub_133130		sub_1358A0
off_1435E4	dd offset sub_1337C0		sub_132590
off_1435E8	dd offset sub_1353E0		
off_1435EC	dd offset sub_134320		sub_133010 sub_1358A0
off_1435F4	dd offset sub_133D60	0x64	sub_check_-5 sub_132930
off_1435F8	dd offset sub_132B20		
off_1435FC	dd offset sub_134960		sub_1358A0
off_143600	dd offset sub_135640		

נתחיל במיפוי הנתיבים שאי אפשר לשנות אותם - זאת אומרת פונקציות בשתי קצוות הניתוב, מהסוף או מההתחלה שאנו יודעים בוודאות לאיזה פונקציה לנתב אותן - או מאיזה פונקציה לקרוא להן (ע"י ביצוע XREF-ים ומיפוי הקריאות בפונקציות ניתן להבין - פונקציות ניתוב 1 עד 3 ופונקציות ניתוב 5- עד rf_vectored_exception_handler שהיא האחרונה הן פונקציות הנופלות תחת הקטגוריה).

בשלב זה אנחנו מכירים 8 פונקציות שניהיה מחויבים לנתב דרכן וחסרות לנו כמה חוליות ניתוב בדרך, עכשיו הגיע הזמן לבצע קצת עבודה שחורה. יש למפות מאיפה פונקציה 5- נקראת, בנוסף למפות את כל הפונקציות החסרות, ולהבין מאיפה כל אחת מהן נקראת עד שנוכל להשלים את החוליות החסרות בין פונקציה 3 ל-5-.

כל המיפוי נעשה ידנית ע"י XREF-ים, התוצר הוא הטבלה משמאל.

בשביל להשלים את החוליות החסרות נדרשתי למפות 10 פונקציות אחריון הצלחתי לחבר בין ההתחלה לסוף ולבנות ניתוב.

עכשיו נותר רק לחשב עבור כל פונקציה שאנו עוברים דרכה את הערך עלינו להכניס אליה בשביל לעבור בהסתעפות המתאימה לנו. מתקבלת הטבלה הבאה:

פונקציה	ערך מעבר
0x1355E0	44
0x132530	42
0x1352A0	14
0x133680	1
0x134C70	41
0x134780	36
0x1326E0	0
0x133E10	27
0x133F50	48
0x135750	8
0x133AE0	-

אתגרי ה CrackMe של רפאל 2015

www.DigitalWhisper.co.il

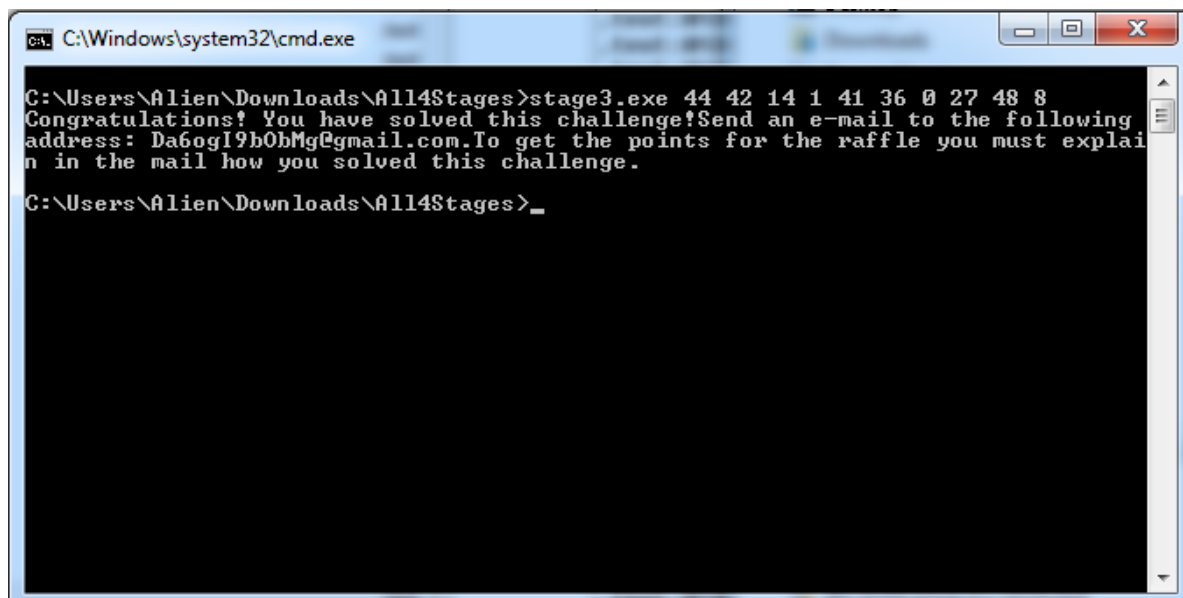
ניתוב אחרון:

```

00133AE0
00133AE0
00133AE0 ; Attributes: bp-based frame
00133AE0
00133AE0 rf_vectorored_exception_handler proc near
00133AE0
00133AE0 var_4= dword ptr -4
00133AE0 arg_0= dword ptr 8
00133AE0
00133AE0 push ebp
00133AE1 mov ebp, esp
00133AE3 push ecx
00133AE4 movzx eax, byte_1447B8
00133AE8 mov byte_1446B8[eax], 36h
00133AF2 mov cl, byte_1447B8
00133AF8 add cl, 1
00133AFB mov byte_1447B8, cl
00133B01 mov off_14360C, esp
00133B07 mov esp, dword_1447D4
00133B0D push offset Handler ; Handler
00133B12 push 0 ; First
00133B14 call ds:AddVectoredExceptionHandler
00133B1A call sub_132480
00133B1F push offset Handler ; Handle
00133B24 call ds:RemoveVectoredExceptionHandler
00133B2A mov esp, off_14360C
00133B30 mov edx, [ebp+arg_0]
00133B33 mov eax, [edx]
00133B35 mov [ebp+var_4], eax
00133B38 mov ecx, [ebp+var_4]
00133B3B sub ecx, 9
00133B3E mov [ebp+var_4], ecx
00133B41 cmp [ebp+var_4], 26h ; switch 39 cases
00133B45 ja loc_133BDE ; jumtable 00403B55 default case

```

ניתן לשים BP על הפונקציה [AddVectoredExceptionHandler](#) ולהריץ, נראה כי אנו באמת מגיעים אל ה-BP שלנו, ה-String שמתקבל הוא "WMardD87Qy6" - כנראה מפתח הפיענוח להודעת ההצלחה, הפונקציה מגדירה את sub_132480 כ-Exception Handler, יודע לקרוא לו ככה בעצמו (Handler), לאחר מכן יש קריאה לפונקציה sub_13B24B0 אשר מבצעת Exception והריצה מועברת ל-Handler משם מתבצע מעבר (כמובן רק אם נעביר את ה-Exception לאפליקציה ולא נטפל בו בעצמנו) לפונקציה המיוחלת שלנו (sub_1320D0) עכשיו רק נותר לבדוק אם באמת הצלחנו.



נראית כמו התוצאה הרצויה.



תודות

תודה לחברה שלי, שרון שתרמה מזמנה לקרוא את המאמר ותמכה בי לאורך כתיבתו.
תודה לשחק שלו שנתן מזמנו בשביל לקרוא טיוטה ארוכה מאוד.
תודה ל-Digital Whisper שפרסמו את המאמר וממשיכים להזרים מאמרים באיכות גבוהה לאורך השנים.

ליצירת קשר ניתן לפנות לכתובת: d.pshoul@gmail.com



נספח קישורים ופונקציות

- `RegOpenKeyEx` - הפונקציה מקבלת מפתח ל-Registry ופותחת אותו, מחזירה `Handle` למפתח:
[https://msdn.microsoft.com/en-us/library/windows/desktop/ms724897\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/desktop/ms724897(v=vs.85).aspx)
- `RegQueryValueEx` - הפונקציה מקבלת `Handle` למפתח פתוח, נתיב + ערך לתשאל ומחזירה את המידע בתוך הערך:
[https://msdn.microsoft.com/en-us/library/windows/desktop/ms724911\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/desktop/ms724911(v=vs.85).aspx)
- `CryptBinaryToString` - הפונקציה מקבלת מידע וצורת המרה, מחזירה מידע מומר:
[https://msdn.microsoft.com/en-us/library/windows/desktop/aa379887\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/desktop/aa379887(v=vs.85).aspx)
- `Base64 encoding` - שיטת קידוד נפוצה, תומכת בקידוד מידע בינארי ל-ASCII:
<https://en.wikipedia.org/wiki/Base64>
- `Strncmp` - הפונקציה מקבלת 2 String-ים ומשווה ביניהם:
<https://msdn.microsoft.com/en-us/library/eywx8zcx.aspx>
- `GetTickCount` - הפונקציה מחזירה את הזמן במילישניות מאז הפעלת מערכת ההפעלה:
[https://msdn.microsoft.com/en-us/library/windows/desktop/ms724408\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/desktop/ms724408(v=vs.85).aspx)
- `GetProcAddress` - הפונקציה מקבלת `Handle` ל-DLL ושם של פונקציה ומחזירה את כתובת של הפונקציה:
[https://msdn.microsoft.com/en-us/library/windows/desktop/ms683212\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/desktop/ms683212(v=vs.85).aspx)
- `Fputc` - הפונקציה מקבלת `Byte` ו-`Handle` לקובץ וכותב את ה-`Byte`- לקובץ:
<https://msdn.microsoft.com/en-us/library/yah67377.aspx>
- `NtSetInformationThread` - הפונקציה מקבלת `Handle` ל-`Thread` ופרמטרים נוספים המורים לה על שינויים שעליה לבצע על ה-`Thread`, הפונקציה מתועדת בצורה חלקית ע"י Microsoft:
[https://msdn.microsoft.com/en-us/library/windows/hardware/ff567101\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/hardware/ff567101(v=vs.85).aspx)
- `VirtualProtect` - הפונקציה מקבלת כתובת וקבוע הרשאות, הפונקציה מכילה את ההרשאות המבוקשות על ה-`Page` של הכתובת:
[https://msdn.microsoft.com/en-us/library/windows/desktop/aa366898\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/desktop/aa366898(v=vs.85).aspx)



- NtQueryInformationProcess - הפונקציה מקבלת Handle ל-Process הנוכחי ופרמטרים נוספים המורים לה על מידע עליו נרצה לעשות שאילתה, הפונקציה מתועדת בצורה חלקית ע"י Microsoft:
[https://msdn.microsoft.com/en-us/library/windows/desktop/ms684280\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/desktop/ms684280(v=vs.85).aspx)
- IsDebuggerPresent - הפונקציה בודקת האם יש Debugger המסתכל על Process, מחזירה True או False):
[https://msdn.microsoft.com/en-us/library/windows/desktop/ms680345\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/desktop/ms680345(v=vs.85).aspx)
- QueryPerformanceCounter - הפונקציה מחזירה את ערך ה-Performance Counter הנוכחי:
[https://msdn.microsoft.com/en-us/library/windows/desktop/ms644904\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/desktop/ms644904(v=vs.85).aspx)
- GetModuleHandle - הפונקציה מקבל NULL או שם של מודול ומחזירה את כתובת הטעינה שלו - (BaseAddress):
[https://msdn.microsoft.com/en-us/library/windows/desktop/ms683199\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/desktop/ms683199(v=vs.85).aspx)
- WaitForMultipleObjects - הפונקציה מחכה זמן מסוים או זמן אינסופי לאחד או יותר מהאובייקטים שהועברו לה במערך):
[https://msdn.microsoft.com/en-us/library/windows/desktop/ms687025\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/desktop/ms687025(v=vs.85).aspx)
- Atoi - הפונקציה מקבלת מידע מספרי ב-String וממירה אותו לערך מספרי, ר"ת של (Ascii to int):
<https://msdn.microsoft.com/en-us/library/yd5xkb5c.aspx>
- SetUnhandledExceptionFilter - הפונקציה מקבלת פונקציה ומתקינה אותה כ- UnhandledExceptionFilter, במידה ויש Debugger הפונקציה לא תעביר את השליטה לפונקציה המותקנת):
[https://msdn.microsoft.com/en-us/library/windows/desktop/ms680634\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/desktop/ms680634(v=vs.85).aspx)
- AddVectoredExceptionHandler - הפונקציה מקבלת פונקציה ורושמת אותה VectoredExceptionHandler):
[https://msdn.microsoft.com/en-us/library/windows/desktop/ms679274\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/desktop/ms679274(v=vs.85).aspx)
- RemoveVectoredExceptionHandler - הפונקציה מקבלת פונקציה ומורידה אותה מה- VectoredExceptionHandler):
[https://msdn.microsoft.com/en-us/library/windows/desktop/ms680571\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/desktop/ms680571(v=vs.85).aspx)



LoadLibraryA - הפונקציה מקבלת נתיב של קובץ DLL וטוענת אותו:

[https://msdn.microsoft.com/en-us/library/windows/desktop/ms684175\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/desktop/ms684175(v=vs.85).aspx)

- OutputDebugStringA - הפונקציה מדפיסה String ל-Debugger:

[https://msdn.microsoft.com/en-us/library/windows/desktop/aa363362\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/desktop/aa363362(v=vs.85).aspx)

- CreateMutexA - הפונקציה יוצרת אובייקט של סינכרוניזציה - Mutex, ReleaseMutex - הפונקציה

משחררת Mutex:

[https://msdn.microsoft.com/en-us/library/windows/desktop/ms682411\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/desktop/ms682411(v=vs.85).aspx)

- CreateThread - הפונקציה יוצרת Context ריצה נוסף המתבטא ביצירת Thread מקביל:

[https://msdn.microsoft.com/en-us/library/windows/desktop/ms682453\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/desktop/ms682453(v=vs.85).aspx)

- PE - פורמט קבצי ההרצה ב-Windows:

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

- EPROCESS - מבנה נתונים של ה-Kernel המתאר Process:

http://www.nirsoft.net/kernel_struct/vista/EPROCESS.html

- PEB - מבנה נתונים הנמצא בכל Process השומר מידע שימושי על ה-Process:

[https://msdn.microsoft.com/en-us/library/windows/desktop/aa813706\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/desktop/aa813706(v=vs.85).aspx)

- ntnative.h - קובץ Header המתאר מבני נתונים של פונקציות Native:

<http://mytoybox.googlecode.com/svn/trunk/SetEvent/ntnative.h>

- Argv - העברת משתנים לקובץ הרצה ע"י שורת פקודה - למשל בהרצה: 1.exe param_a param_b:

https://en.wikipedia.org/wiki/Entry_point#C_and_C.2B.2B

הסודות החבואים ב-WebSocket

מאת רזיאל בקר

הקדמה

רשת האינטרנט "הקלאסית" כיום עובדת בתבנית המבוססת על "בקשה-תגובה" של פרוטוקול HTTP. הדפדפן טוען את עמוד האינטרנט, ועד שהמשתמש לא לוחץ על קישור בדף לא קורה שום דבר מבחינת הרשת. בשנת 2005, טכנולוגיית ה-Ajax נכנסה לשוק ובעזרתה אתרים הפכו ליותר דינמיים. אך עם זאת, כל התקשורת בין הדפדפן לאתר התבצעה על ידי הפרוטוקול HTTP כך שתמיד נדרשה התערבות מצד הלקוח כדי לקבל מידע מהשרת.

אתרי האינטרנט גדלו ועם גדילתם גדל הצורך להעביר יותר ויותר נתונים, הבעיה ב-HTTP הייתה שבכל פעם שהדפדפן היה צריך לקבל נתונים מהשרת, הוא היה צריך לשלוח בקשה אליו כל פעם מחדש כדי לבדוק האם ישנו נתון חדש שעליו לקבל והאם להשתמש בו או לא, אחד הפתרונות היצירתיים הוא HTTP

BROWSER	MAX PARALLEL CONNECTIONS PER HOST
IE 6 and 7	2
IE 8	6
IE 9	6
IE 10	8
Firefox 2	2
Firefox 3	6
Firefox 4 to 17	6
Opera 9.63	4
Opera 10	8
Opera 11 and 12	6
Chrome 1 and 2	6
Chrome 3	4
Chrome 4 to 23	6
Safari 3 and 4	4

כמות החיבורים המקסימלית למארח לכל דפדפן

⁸Long Polling, הדפדפן שולח בקשת HTTP לשרת ומחכה לתגובה שלו, ברגע שיש מידע חדש שהדפדפן אמור לקבל מהשרת השרת מגיב עם המידע החדש והדפדפן שולח את הבקשה עוד פעם וכך יוצא מצב שבעצם מפתח האתר גורם ל-DOS על השרת שלו, בעיה נוספת היא מגבלת החיבורים למארח⁹ הדפדפן מגביל את החיבורים כדי שהשרת לא יוצף בבקשות על ידי הדפדפן.

אחת הבעיות הגדולות בפתרונות קשורים למודל ה-HTTP היא שבכל פעם שהדפדפן ישלח בקשת HTTP אל השרת, בבקשה יכללו המון נתונים שברב המקרים השרת לא צריך אותם, הנתונים האלו נקראים כותרים (Headers) וזה יכול להגיע למצב

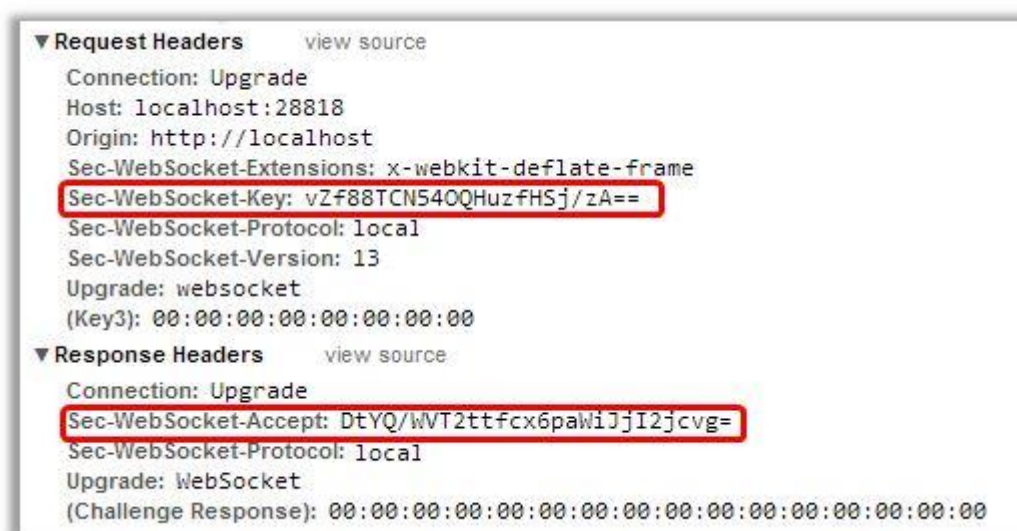
⁸ <http://stackoverflow.com/a/333884>

⁹ <http://sgdev-blog.blogspot.sg/2014/01/maximum-concurrent-connection-to-same.html>

מה שאנחנו באמת צריכים הוא יצירת חיבור קבוע בין הדפדפן לשרת מה שיקצר את זמן התגובה וכאן פרוטוקול ה-Websocket נכנס למשחק.

משה! < קראת לי? < כן! (Handshake)

התקשורת ב-Websocket מחולקת ל-2: לחיצת היד (Handshake) והעברת נתונים. כאשר אובייקט Websocket נוצר בדפדפן מאחורי הקלעים מתבצעת "לחיצת יד" בין הדפדפן לשרת. הדפדפן שולח בקשת HTTP GET Upgrade לשרת.



בתמונה למעלה אנחנו יכולים לראות את הכותרת Upgrade: websocket שמצביעה על בקשת "שדרוג" הפרוטוקול. השרת "משדרג" את הפרוטוקול ל-Websocket שהוא פרוטוקול מבוסס TCP, בקשת ה"שדרוג" היא גשר בין פרוטוקול ה-HTTP ל-Websocket, בלחיצת היד הנתונים המועברים הם "משא ומתן" כדי לוודא שאין בעיות אבטחה.

2 הידירים חשובים הם "Sec-WebSocket-Key" ו-"Sec-WebSocket-Accept": הדפדפן שולח את הכותרת "Sec-WebSocket-Key" המכילה בתים אקראיים מקודדים ב-base64 והשרת מגיב עם המפתח המגובב ב-"Sec-WebSocket-Accept", לחיצת היד נועדה כדי למנוע שהמטמון ישלח "שיחה" קודמת מסוג WebSocket ואינה מספקת שום אימות עם השרת או פרטיות המידע. פונקציית הגיבוב מצרפת את המחרוזת הקבועה "258EAF5E91447DA95CA5AB0DC85B11" (GUID) לערך "Sec-WebSocket-Accept".



"Key" (שאינו מפוענח ב-base64), חלה על המחרוזת SHA1 ומקודדת ב-base64. השרת שלכם יצטרך לעקוב אחרי החיבורים הפעילים של כל המשתמשים, כך כדי שלא לשמור על לחיצת היד עם הלקוחות שכבר השלימו את לחיצת היד. כתובת IP יכולה להתחבר מספר פעמים.

במידה והחיבורים רבים הדפדפן יחסום אותם כדי להמנע מהתקפת מניעת שירות (DoS).

העברת הנתונים

הדפדפן והשרת יכולים לבחור לשלוח הודעה בכל רגע זה אחד מהיתרונות של הפרוטוקול WebSocket. הנתונים העוברים בין הדפדפן לשרת מוצפנים באמצעות XOR¹⁰ (עם מפתח של 32 סיביות). כל frame המועבר בין הדפדפן לשרת נשלח במסגרת הזאת:

1	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1
2	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1
3	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+
4		F		R		R		R		opcode		M		Payload len		Extended payload length						
5		I		S		S		S		(4)		A		(7)		(16/64)						
6		N		V		V		V				S				(if payload len==126/127)						
7				1		2		3				K										
8	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+
9																						
10	+	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
11																						
12	+																					
13		Masking-key (continued)																				
14	+																					
15	:																					
16	+																					
17																						
18	+																					

שדה ה-Mask (סיבית) מייצג האם ה-Payload מקודד או לא, מידע שנשלח על ידי הדפדפן חייב לעבור את תהליך המיסוך ולכן השרת מצפה ל-1 (אם הסיבית תהיה 0 השרת ינתק את החיבור עם הדפדפן), המידע שנשלח על ידי השרת לא מקודד ולכן שדה ה-Mask כבוי, אני אסביר על תהליך המיסוך בהמשך.

ההודעות ממוסכות על ידי הדפדפן גם בעת שימוש בחיבור מאובטח.

¹⁰ https://en.wikipedia.org/wiki/XOR_cipher



שדה ה-opcode (4 סיביות), מגדיר את היחס למטען (Payload) שהתקבל:

- 0x01 - טקסט
- 0x02 - נתונים בינאריים

בגרסה הנוכחית של WebSocket, ל-0x3 והלאה אין משמעות.

שדה ה-Fin (סיבית) מייצג האם המטען שהתקבל הוא המטען האחרון בסדרה, אם השדה כבוי השרת יחכה לעוד חלקים של ההודעה.

פענוח אורך ה-Payload

כדי לקרוא את המטען, אנחנו צריכים לדעת מתי להפסיק לקרוא אותו. זו הסיבה מדוע אורך המטען חשוב לנו, קחו את הסיביות ב-offest 9 עד ה-offest 15 והעבירו אותם לפורמט unsigned int, זה אורך המטען, אם אורך המטען הוא 126, תקראו את הבית (16 סיביות) הבא אם אורך המטען הוא 127 תקראו את 8 הבתים (64 סיביות) הבאים והעבירו אותם לפורמט unsigned int.

קריאה ומיסוך המטען

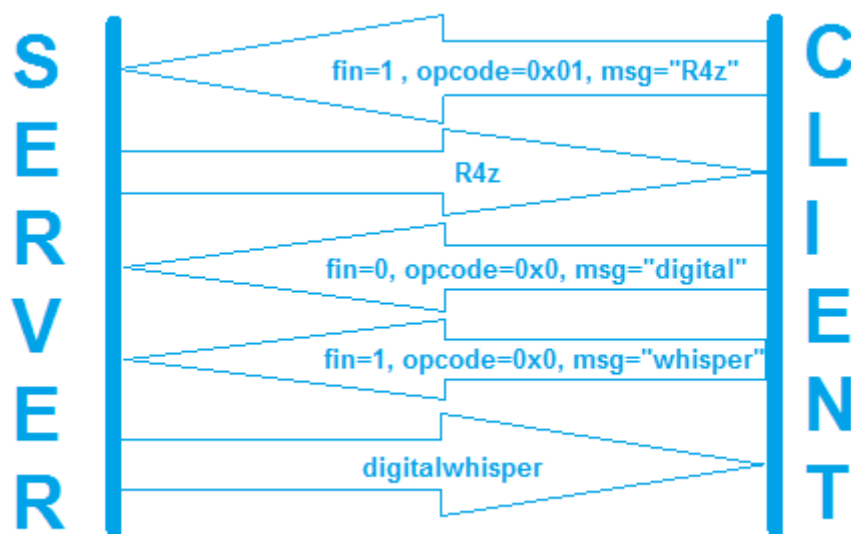
אם שדה ה-Mask דולק (והוא יידלק במידה וההודעה נשלחת על ידי הדפדפן אל השרת), הבתים (32 סיביות) הבאים הם מפתח המיסוך. כעת מפתח המיסוך ואורך המטען בידינו, שאר המידע הוא המטען שלנו. פענוח המטען יתבצע כך:

```
var DECODED = "";
for (var i = 0; i < ENCODED.length; i++) {
    DECODED[i] = ENCODED[i] ^ MASK[i % 4];
}
```

נרוץ בלולאה על המטען שקיבלנו ונבצע XOR עם הבית הנוכחי של המטען והבית ה-4 % i של המפתח. וקיבלנו את המטען המקורי שהדפדפן שלח אל השרת 😊.

חלקת חבילות

כדי להעביר מידע, שדות ה-FIN וה-opcode משתלבות בהעברת הנתונים בין הדפדפן לשרת, תהליך זה נקרא חלקת חבילות (Message Fragmentation) וזמין רק כאשר הערך ב-opcode הוא בין 0x0-0x2. שדה ה-opcode מייצג את סוג המטען, 0x1 מייצג את המטען כטקסטואלי, 0x2 מייצג את המטען כנתונים בינאריים לעומת זאת - אם הערך בשדה ה-opcode הוא 0x0 המסגרת היא מסגרת המשך, זאת אומרת שהשרת צריך לשרשר את ההודעה עם המסגרת האחרונה שקיבל מהדפדפן.



במסגרת הראשונה דגל ה-FIN דולק (fin=1 ו-opcode!=0x0), כך הדפדפן מודיע לשרת שזה סוף ההודעה. במסגרת השנייה הדפדפן שלח הודעת טקסט (opcode=0x1) אבל השרת עדיין מחכה לסוף ההודעה (דגל ה-FIN כבוי) ואת המסגרת האחרונה הדפדפן ישלח כאשר דגל ה-FIN דולק.

פינג-פונג ב-WebSocket

אחרי לחיצת היד בין הדפדפן לשרת, הדפדפן או השרת שולחים חבילת פינג. כשהפינג מתקבל - הנמען חייב לשלוח חבילת פונג, הפרוטוקול משתמש בזה כדי לוודא שהלקוח עדיין מחובר. פינג פונג או הוא רק מסגרת קבועה, אבל זה מסגרת שליטה. יש לי Pings opcode של 0x9, ויש לי pongs opcode של 0xA. כאשר אתה מקבל פינג, לשלוח בחזרה פונג עם אותם נתונים המדויקים כמטען פינג (ל-pings ו-pongs, אורך המטען המרבי הוא 125). ייתכן גם לקבל פונג מבלי לשלוח פינג; להתעלם מכך אם זה קורה.

חבילות הפינג או הפונג הן בעצם מסגרת קבועה, הערך ב-opcode 0x9 מייצג פינג והערך 0xA מייצג פונג, כאשר הדפדפן מקבל פינג הוא חייב לשלוח בחזרה פונג עם אותה הודעה מדויקת, לחבילות הפינג-פונג אורך המטען הוא 125 - השרת יכול גם לקבל פונג מבלי שהוא ישלח פינג.

אם אתה מקבל פינג אחד או יותר לפני שאתה שולח חבילת פונג, אתה רק צריך לשלוח חבילת פונג אחת.



סגירת החיבור

כדי לסגור חיבור בין הלקוח או שרת יכול לשלוח מסגרת שליטה בנתונים המכילים רצף שליטה צוין להתחיל לחיצת יד הסגירה (מפורט בסעיף 5.5.1). עם קבלת מסגרת כזו, העמיתים האחרים שולח מסגרת לסגור בתגובה. העמיתים הראשונים אז סוגרים את החיבור. כל הנתונים נוספים שהתקבלו לאחר סגירת החיבור אז הושלכה.

סגירת החיבור בין הדפדפן לשרת מתבצעת על ידי שליחת חבילת סגירה (`opcode=0x8`), הנמען שולח מסגרת סגירה בתגובה ולאחר מכן סגירת החיבור מתבצעת על ידי שני הצדדים. המטען מכיל את הסיבה לסגירת החיבור (אם החבילה שהתקבלה גדולה משמעותית או לא עמדה בתקן). גם חבילות סגירה הנשלחות מהדפדפן אל הלקוח עוברות תהליך מיסוך.

אז איפה הכיף?

הערת צד: ישנן בעיות חוקיות בכל הנוגע להנדסה לאחור. מדריך זה נועד ללימוד עצמי בלבד!

אחרי שאנחנו מכירים את הפרוטוקול לעומק, מבינים איך הוא עובד - מה קורה ברקע ומה עומד מאחורי כל תכונה, ואף אם ניקח את זה רחוק יותר, יישום לקוח או שרת מבוססים Websocket. עכשיו אפשר לעבור לצד המעשי או במילים אחרות - הגענו לחלק הכיפי ☺.

הפרוטוקול נכנס לשוק לא מזמן ולכן שירותים רבים משתמשים ב-Websocket (משחקי דפדפן, Skype, Whatsapp ועוד) וזה נובע מכך שהפרוטוקול עדיין טרי - מפתחים רבים לא מבינים איך לעבוד איתו ואיך לשלב אותו נכון במערכות שונות. בהמשך, ננתח משחק דפדפן (MMORPG) מבוסס Websocket ובהמשך:

- נבין את המחשבה שעומדת מאחורי כל החלטה שהמערכת מקבלת ואיך אפשר לנצל אותה.
- נלמד "לרמות" במשחק ואני מתכוון לעריכת שם משתמש, חפצים וכל נתון אחר שתבחרו.
- כתיבת סקריפט שמתחבר אוטומטית לכל שחקן.

תצטרכו:

- גוגל כרום כדי להנדס לאחור (Reverse Engineering)
- ידע בסיסי ב-Javascript
- Wireshark למי שלא אוהב להשתמש בכרום כדי לנתח חבילות
- להשתמש בראש שלכם ☺

BrowserQuest

BrowserQuest הוא משחק MMORPG שנכתב על ידי Little Workshop ו-Mozilla Foundation, המשחק נכתב ב-HTML5 ו-WebSocket ותומך בדפדפנים העדכניים. BrowserQuest שוחרר כ-Open Source ב-GitHub. הקוד תחת הרישיון [MPL 2.0](https://www.mozilla.org/en-US/MPL/2.0/) תחת [CC BY-SA](https://creativecommons.org/licenses/by-sa/3.0/) 3.0.

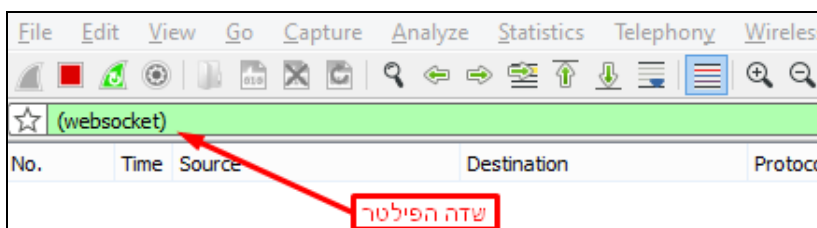
מידע נוסף:

- [HTTP://www.littleworkshop.fr/browserquest.HTML](http://www.littleworkshop.fr/browserquest.HTML)
- <https://hacks.mozilla.org/2012/03/browserquest/>
- [HTTP://www.engadget.com/2012/03/28/mozilla-browserquest-HTML5-game/](http://www.engadget.com/2012/03/28/mozilla-browserquest-HTML5-game/)

הסנפת חבילות Websocket עם Google Chrome ו-Wireshark

הסנפת חבילות Websocket ב-Wireshark:

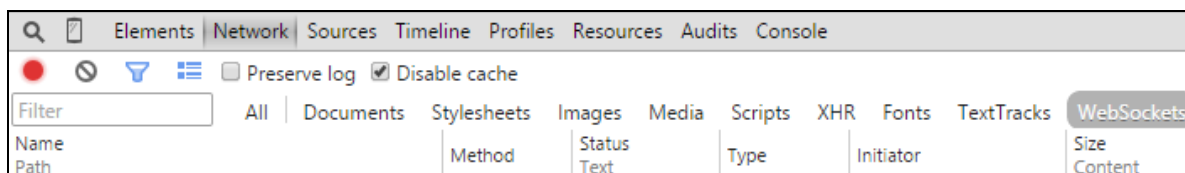
1. Capture Options
2. בחר בכרטיס הרשת
3. מלא את שדה הפילטר ב-(websocket) או tcp port 8001:



באמצעות Chrome Developer tools אפשר לצפות במטענים שעוברים בין הדפדפן לשרת ולצפות בו אם המידע אינו נתונים בינאריים (opcode=0x1). הסנפת חבילות Websocket ב-Google Chrome:

1. אתחלו את Google Developer tools (ctrl+shift+c או F12)
2. רענונו את הדף
3. לחצו על "Network" בטאבים.
4. בחרו בחיבור ה-WebSocket (אם הסטטוס הוא 101 Switching Protocol) או בטאב

:WebSockets



אופציה נוספת היא להשתמש ב-XSS-Track - לקריאה נוספת:

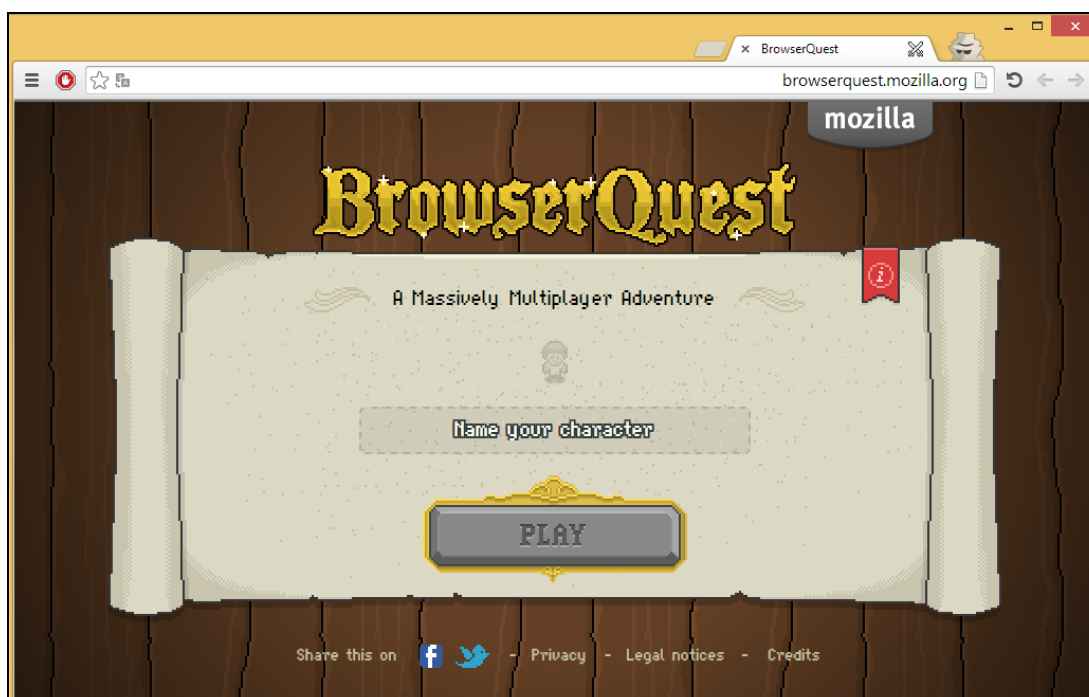
[HTTP://blog.kotowicz.net/2011/01/xss-track-as-HTML5-websockets-traffic.HTML](http://blog.kotowicz.net/2011/01/xss-track-as-HTML5-websockets-traffic.HTML)

הסודות החבואים ב-WebSocket-

www.DigitalWhisper.co.il

וידבג

יאללה בואו נתחיל, פתחתי את המשחק BrowserQuest ב-Google Chrome גרסא 45:



המשחק הוא משחק דפדפן, כדי שהדפדפן יבצע פעולה מסוימת, צריך קוד javascript שיוצר אותה. מה שאומר שהקוד javascript אחראי על מה שהדפדפן מראה לנו והדפדפן מראה לנו את מה שהמפתח רצה שנראה - חושבים ששווה לעבור על הקוד?

Filter	All	Documents	Stylesheets	Images	Media	Scripts
Name	Method	Status	Type			
<input type="checkbox"/> modernizr.js	GET	304	application/			
<input type="checkbox"/> detect.js	GET	304	application/			
<input type="checkbox"/> log.js	GET	304	application/			
<input type="checkbox"/> require-jquery.js	GET	304	application/			
<input type="checkbox"/> ga.js	GET	304	text/javascr			
<input type="checkbox"/> home.js?bust=1	GET	304	application/			
<input type="checkbox"/> game.js?bust=1	GET	304	application/			
<input type="checkbox"/> mapworker.js	GET	304	application/			

משמות הקבצים ניתן להבין שהכי מעניינים הם home.js ו-game.js (בדקתי ואכן הם קבצים ששווה לנו לעבור עליהם) אם תנסו לקרוא את הקוד, תצטערו לגלות שהקוד מסורבל וקשה לקריאה בדרך כלל מפתחים משתמשים בזה כדי שהקובץ יתפוס פחות מקום או Anti-debugging, במקרים כאלה אפשר להשתמש בכלי codebeautify.org. באמצעותו נהפוך את הקוד לקריא, לאחר מכן נעתיק אותו לעורך שלנו (אני משתמש ב-Notepad++).

אבל קודם כל, אני אראה לכם איך אפשר להתחבר לשרת websocket ב-javascript.

הסודות החבואים ב-WebSocket

www.DigitalWhisper.co.il



כתיבת ה-client ב-javascript

נאתחל את Chrome Developer tools כדי להסניף את המטענים שעוברים ב-Websocket:

Name Path	× Headers Frames
178.79.166.11	Data
	[[2,572389342,1,44,205,"Saraphina19",3,23,61]]
	[[2,926,61,60,206],[2,927,61,34,210],[2,928,39,77,232],[2,1120,2,47,21
	[20,926,927,928,1120,1121,1122,1220,1221,1222,1820,1821,11920,1
	[[17,13,null],[19,926,927,928,1120,1121,1122,1220,1221,1222,1820,1
	[1,52389348,"R4z",71,223,80]
	[0,"R4z",21,60]
1 / 241 requests 0 B / 7.6 MB trans...	90°

בלשונית Frames אפשר לראות את המטענים שעוברים בין הדפדפן לשרת. המטענים שהשרת שולח אל הדפדפן מודגשים בלבן והמטענים שהדפדפן שולח אל השרת מודגשים בירוק. נכתוב את ה-client ב-javascript (כמובן שאפשר לכתוב אותו בכל שפה, בחרתי לכתוב אותו ב-javascript מכיוון שנוח להשתמש ב-API של javascript - לעוד מידע: <http://bit.ly/1FziUWg>):

```
var host = "ws://178.79.166.11/"; // הכתובת של השרת שאנחנו רוצים להתחבר אליו
var socket = new WebSocket(host); // פתיחת החיבור בין הדפדפן לשרת(תהליך לחיצת היד מתבצע כאן)
console.log(socket.readyState);
socket.onopen = function(){
    console.log('Socket Status: '+socket.readyState+ ' (open)');
}
socket.onmessage = function(msg){
    console.log('Received: '+msg.data);
    if(msg.data=="go") { // כשאנחנו מקבלים מהשרת הודעה כזו אנחנו מבינים שאנחנו צריכים לשלוח לו מידע על השחקן שלנו
        socket.send('['+0,"Digitalwhisper",23,65]'); // שם משתמש + מיקום
        socket.send('['+20,90,924,925,926,927,1021,1120,1121,1122,1220,1221,1222,1320,1321,1322,1820,1821,84195,87195,815222,818209,837200,841195,847195,867220,870197,873197,8154138,540386503,551386501,564386450]');
    }
}
socket.onclose = function(){
    console.log('Socket Status: '+socket.readyState+ ' (Closed)');
}
```

נריץ את הקוד javascript באמצעות הדפדפן ..



הסודות החבואים ב-WebSocket

www.DigitalWhisper.co.il



home.js

נתחיל ב-home.js, נסתכל עליו קצת ונראה משהו כזה:

```
define("storage", [], function() {  
    ...  
    ...  
});
```

השימוש ב-define משמש להגדרת מודלים חדשים ב-requireJS, requireJS טוען מודלים, הם עוברים תהליך אופטימיזציה בטעינת המודל על ידי הדפדפן. requireJS יכול לשמש גם סביבות נוספות כמו [Rhino](#) או [Node](#).

RequireJS מאפשר לך מהירות ואיכות גבוהה יותר. הקוד javascript יוצר מודל חדש בשם storage, כנראה שזה המודל שאחראי על אחסון הנתונים - לא חושבים ששווה להעביר מבט?

```
var a = Class.extend({  
    init: function() {  
        this.hasLocalStorage() && localStorage.data ? this.data =  
        JSON.parse(localStorage.data) : this.resetData()  
    },
```

מפתח המשחק השתמש ב-Local Storage כדי לאחסן נתונים מסויימים בדפדפן שלנו, אחסון נתונים לוקאלי בדפדפן הוא כלי חזק עבור מפתחי Web. שימוש ב-Local Storage¹¹ בדפדפנים עדכניים:

```
localStorage.setItem('username', 'R4z'); // אחסון נתונים  
var username = localStorage.getItem('username'); // קריאת נתונים  
localStorage.removeItem('username'); // מחיקת נתונים  
כדי לאחסן נתונים באופן זמני ניתן להשתמש ב-Session Storage (עד שהדפדפן יסגר).  
  
resetData: function() {  
    this.data = {  
        hasAlreadyPlayed: !1,  
        player: {  
            name: "",  
            weapon: "",  
            armor: "",  
            image: "",  
            ...  
        },  
    },
```

התכונה resetData (מודגש בצהוב), מהשם resetData אפשר להבין שהוא אחראי על איפוס המידע, המתודה מאפסת נתונים על השחקן (מודגש בירוק), אפשר להבין שנתונים כמו שם השחקן, הנשק, שריון והתמונה מאופסים מה שאומר שהמשחק משתמש בנתונים האלה בשביל ביצוע לפחות פעולה אחת במשחק.

¹¹ <http://www.smashingmagazine.com/2010/10/local-storage-and-how-to-use-it/>

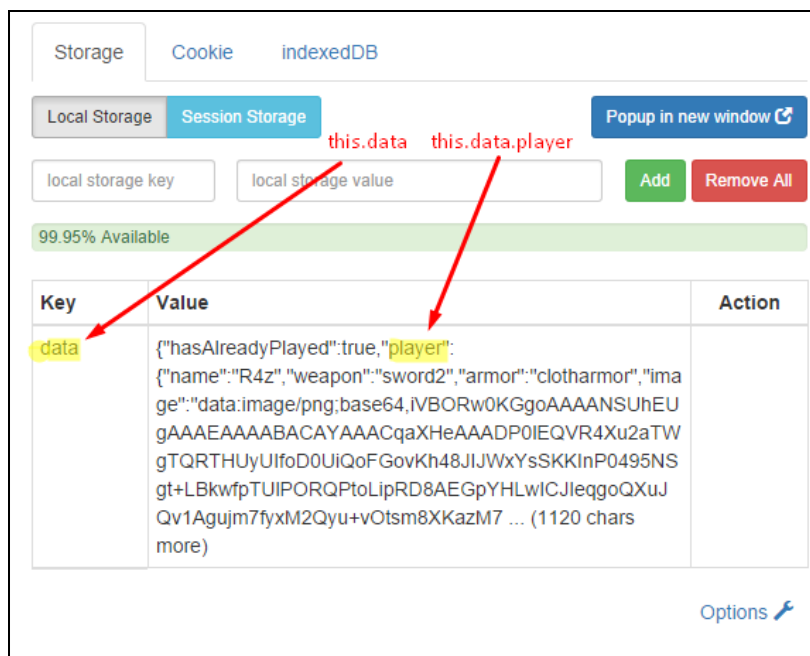
הנתונים שהאתר מאחסן בדפדפן (Local / Session Storage) יושבים בתיקיית ה-AppData בנתיב:

- %LocalAppData%\Google\Chrome\User Data\Default\Local Storage (Windows)
- ~/Library/Application Support/Google/Chrome/Default/Local Storage/ (OSX)

כדי לגשת אל הנתונים האלו בצורה נוחה יותר, אפשר להשתמש בתוספת Storage Manager של misc.im. הורדה:

<https://chrome.google.com/webstore/detail/HTML5-storage-manager-all/giompennnhheakjcnobejbngbbkmdnd?hl=en>

אחרי שהורדנו את התוסף, אפשר להתחיל לשחק 😊. אחרי שהרגתם עכבר או שניים, רעננו את העמוד ואתחלו את ה-Storage Manager:



המידע של השחקן שלנו נשמר ב-Local Storage בדפדפן, אם מעניין אותכם מה ה-image מייצג נשתמש שוב ב-codebeautify שוב: <http://codebeautify.org/base64-to-image-converter> הכניסו את הערך של המפתח image אל שדה ה-base64 ושלחו את הטופס:



ה-image מכיל את תמונת השחקן בדף ההתחברות.

הסודות החבואים ב-WebSocket-

www.DigitalWhisper.co.il



כדי לצפות ב-json בצורה יותר קריאה, אפשר להשתמש באתר [HTTP://json.parser.online.fr](http://json.parser.online.fr):

```
{
  "hasAlreadyPlayed":true,
  "player":{
    "name":"R4z",
    "weapon":"sword2",
    "armor":"clotharmor",
    "image":"..."
  },
  "achievements":{
    "unlocked":[
      18,
      11,
      1
    ],
    "ratCount":4,
    "skeletonCount":0,
    "totalKills":4,
    "totalDmg":34,
    "totalRevives":0
  }
}
```

מגניב, הבנו שהמאפיינים של השחקן נשמרים אצלנו בדפדפן או ליתר דיוק ב-Local Storage ואנחנו יודעים שאפשר לערוך את מאפיינים אלו מהסיבה שהם נשמרים על המחשב שלנו, איך זה עוזר לנו? איך נערוך את ה-Local Storage כדי לרמות במשחק?

המפתח player אחראי על השחקן, armor ו-weapon מייצגים את הנשק והשריון של השחקן, הנשק והשריון משפיעים על נק' ההתקפה וההגנה של השחקן הרי ככל שהנשק והשריון ברמה גבוהה יותר, כך אנחנו יותר חזקים אם הנתונים האלו נשמרים אצלנו במחשב, ב-Local Storage אז אפשר לערוך את ה-Local Storage כדי שיהיה לשחקן שלנו את הנשקים הכי טובים. מה הנשקים או השריונים הטובים ביותר, מה האפשרויות שהמשחק מציע לנו? נקבל את התשובות האלו על ידי ניתוח קוד ה-javascript האחראי על ממשק המשחק.

game.js

הקובץ game.js אחראי על ממשק המשחק (הגדרת אובייקטים וכו'), כדי למצוא את החלק הזה של הקוד אפשר לעבור על כולו, אבל אפשר לנסות לחפש את האובייקט שמגדיר את הנשק sword2 לצורך העניין, נסו לחפש את הביטוי: "sword2:". (זה בדור"כ עובד במערכות, פשוט תכנסו לראש של המפתח).

```
define("items", ["item"], function(a) {
    var b = {
        Sword2: a.extend({
            init: function(a) {
                this._super(a, Types.Entities.SWORD2, "weapon"),
            },
            this.lootMessage =
                "You pick up a steel sword"
        }),
        Axe: a.extend({
            init: function(a) {
                this._super(a, Types.Entities.AXE, "weapon"),
            },
            this.lootMessage =
                "You pick up an axe"
        }),
        ...
    });
});
```

אחרי שקיבלנו קצת עזרה מה-js viewer ב-1689 אפשר למצוא את המודול שאחראי על החפצים (items - מסומן בירוק), לפי הסדר שבו המפתח מגדיר את החפצים בקוד אפשר להבין איזה חפץ הכי משתלם לנו לקחת ועל פי השם, כמובן שיותר נכון אפשר לרדת עמוק יותר ולהבין לפי התכונות של החפצים את החפץ החזק ביותר אבל לא נעשה את זה עכשיו.

אם תעברו על כל החפצים תוכלו לראות את ה-GoldenSword ואת ה-GoldenArmor:

```
GoldenSword: a.extend({
    init: function(a) {
        this._super(a, Types.Entities.GOLDENSWORD, "weapon"),
    },
    this.lootMessage = "You pick up the ultimate sword"
}),
...
...
GoldenArmor: a.extend({
    init: function(a) {
        this._super(a, Types.Entities.GOLDENARMOR, "armor"),
    },
    this.lootMessage = "You equip a golden armor"
}),
```

הסודות החבואים ב-WebSocket

www.DigitalWhisper.co.il


```
}
},
```

נחשבו איתי - ממשק המשחק מגדיר את הנשקים שלנו לפי השם שלהם ב-Local Storage, אם נשנה את שם הנשק או השריון ב-Local Storage - נוכל לשנות את החפצים של השחקן שלנו.

נאתחל את ה-Storage Manager שלנו ונערוך את ה-Local Storage בדפדפן כך שהנשק שלנו הוא goldensword והשריון הוא goldenarmor. (שמות החפצים נשמרים באותיות קטנות - אפשר להסיק את זה מכך ששם הנשק והשריון הם שמרים כבר אצלנו).

Key	Value	Action
data	<pre>{ "hasAlreadyPlayed": true, "player": { "name": "R4z", "weapon": "sword2", "armor": "clotharmor", "image": "data:image/png;base64,iVBORw0KGgoAAAANSUhEUgAAAEAAAABACAYAAACQaXHeAAADP0IEQVR4Xu2aTWgTQRTHUyUIfoD0UiQoFGovKh48JIJWxYsSKInP0495NSgt+LBkwfpTUIPQRQPtoLipRD8AEGpYHLwICJleqgoQXUJQv1Agujm7fyxM2Qyu+vOtsm8XKazM7 ... (1126 chars more)" } }</pre>	<div> <div>Dump JSON</div> <div>Edit</div> <div>Remove</div> <div>Show All</div> </div>

עריכת מידע במפתח הנתונים

שנו את הערכים weapon ו-sword, לנשקים שתוצאו (בחרתי בטובים במשחק). שמרו את ה-Local Storage ותרעננו את הדף, לאחר מכן תתחברו לשחקן שלכם.



כעת, השחקן שלנו מנצח כל עכבר במכה אחת ☺



כמובן אפשר לקחת את האפשרות לשינוי ה-Local Storage קדימה (שינוי שם השחקן ואני בטוח שעוד, פשוט תתעמקו בקוד ותחפשו פירצות נפוצות).

אז... איך לא ישחקו עם המידע שלי?

אני מאמין שבמשחק BrowserQuest הכוונה הייתה להשתמש ב-Local Storage כדי להציג את היכולות של javascript ולערב כמה שפחות צד שרת, **בדרך כלל עדיף לשמור נתונים משמעותיים בשרת ולא בלקוח.**

אם עדיין אתם **חייבים** לשמור מידע משמעותי אצל הלקוח **תצפינו אותו** (אפשר להשתמש ב-[CryptoJS](#)). בכדי ליצור מפתח [FingerprintJS](#) או שתוכלו להצפין את המידע בצד הלקוח ולפצח אותו כשהוא נשלח לשרת. במילים אחרות תנסו שהמימוש והבנת התהליך של שמירת / קריאת המידע בדפדפן יהיה כמה שיותר מסובך ואם אתם יכולים להשתמש בשרת כדי לאחסן את המידע הזה - תשתמשו בו. תמיד תשתמשו ב-Obfuscation Javascript אך על תבנו על זה. כמובן שתמיד אפשר לראות איך ההצפנה מתבצעת בצד לקוח ו"לרמות" במשחק אתם צריכים להקשות על תהליך ההבנה ולשבור את רוחו של הרוורסר.

וכמובן - לעולם, אבל לעולם אל תסמכו על קלט שהגיע מהמשתמש.



על המחבר

R4z בן 17 עוסק בפיתוח Web בחברת Articoloo, ובזמנו הפנוי מתעסק באבטחת מידע לכל שאלה או יעוץ ניתן לפנות אליו בשרת ה-IRC של NIX בערוץ [#Security](#) או באימייל, בכתובת: raziel.b7@gmail.com

בנוסף, אני מעוניין להודות לאפיק קסטיאל על עזרתו המועילה למאמר זה.

למידע נוסף

- <https://developer.mozilla.org/en-US/docs/Web/API/WebSocket>
- <http://tools.ietf.org/HTML/rfc6455>
- <http://www.w3.org/TR/2011/WD-websockets-20110419/>
- <http://www.codeproject.com/Articles/531698/Introduction-to-HTML-WebSocket>
- <http://rawkes.com/articles/creating-a-real-time-multiplayer-game-with-websockets-and-node.HTML>



דברי סיכום

בזאת אנחנו סוגרים את הגליון ה-65 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 ביום האחרון של חודש אוקטובר 2015.

אפיק קסטיאל,

ניר אדר,

30.09.2015