digital whisper
Technologic Papers

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".

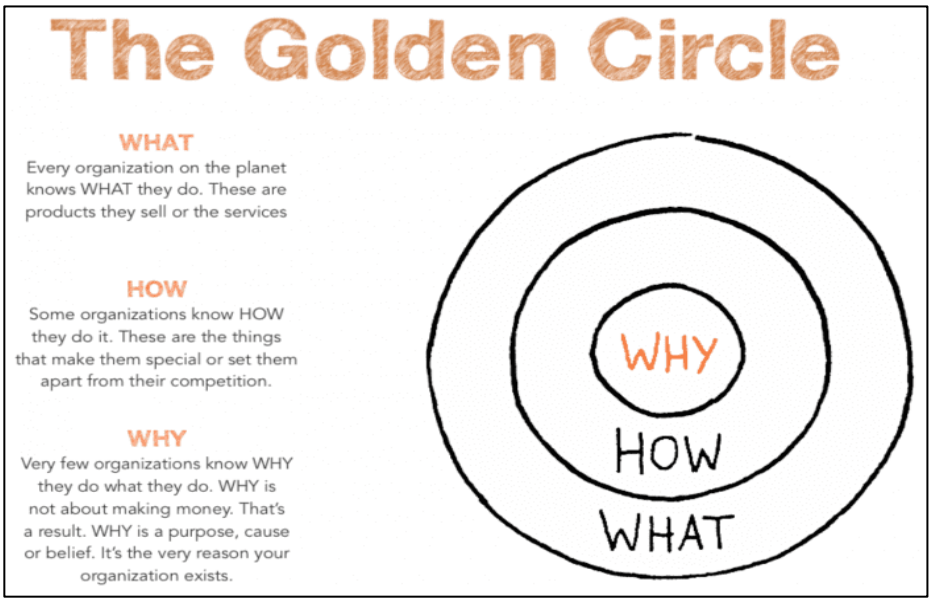
מצד שני, המטרה הסופית שלנו היא לא לעשות קשר דייגים כפול - אלא למצוא דרך שהקירות של הבקתה לא יקרסו לתוך עצמם בלילה גשום.

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?

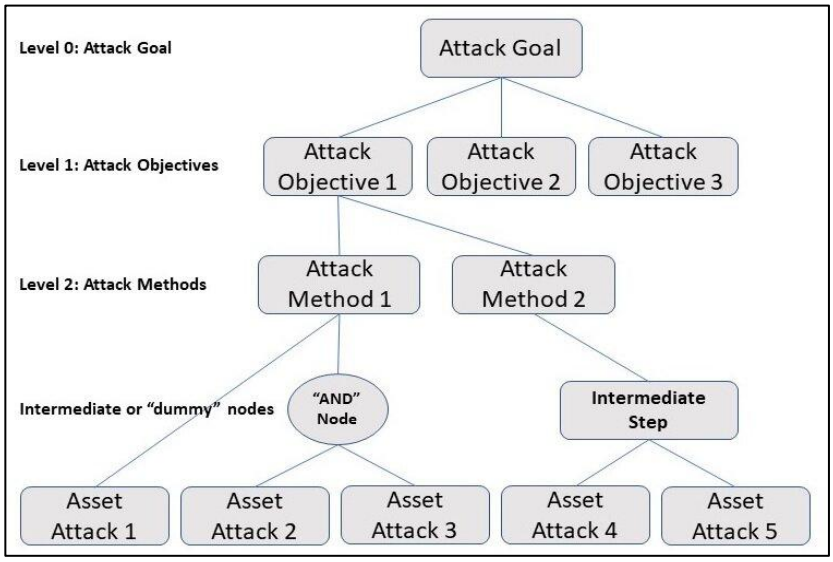
לאחר שמבינים את העובדה שלעבוד freestyle לא יגיע לתוצאה הכי טובה ברוב הבעיות המורכבות שניגש אליהן - צריך למצוא פתרון אחר שיתן לנו פלטפורמה לבצע את התכנון הזה, לשקלל את כל הפרמטרים השונים של הבעיה, לגלות מידע חדש ולפעול בצורה מיושבת וממוקדת לפתרון הבעיה.

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"

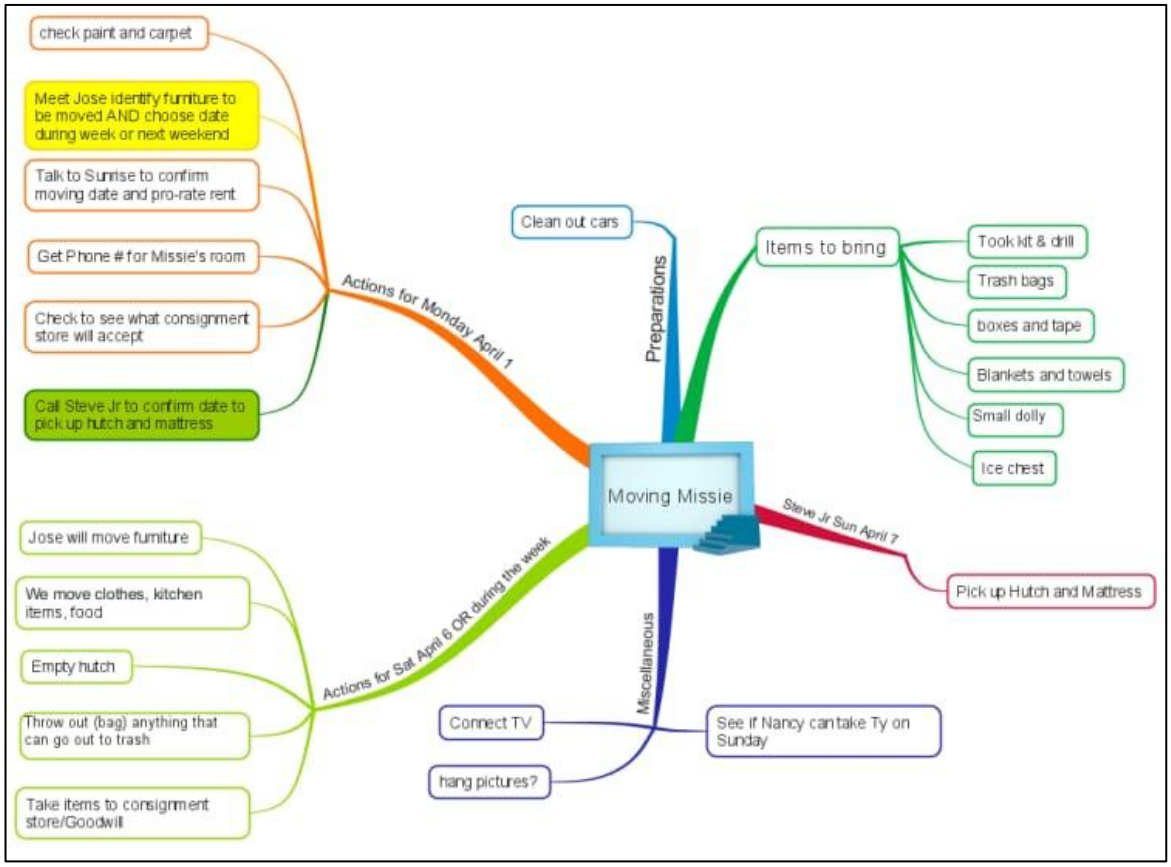
עצי תקיפה (attack trees) / עצי מחשבה (mind trees) הם מודלים גרפיים המשמשים לתיאור צעדים אפשריים לתקיפה של מערכת או מטרה מסוימת. הם עוזרים לזהות נקודות תורפה פוטנציאליות ולתכנן מנגנוני הגנה יעילים יותר, לפרוס בעיה שצריכה פתרון כדי לפתור אותה בצורה יותר ממוקדת או פשוט לחשוב נכון לוגית.



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.

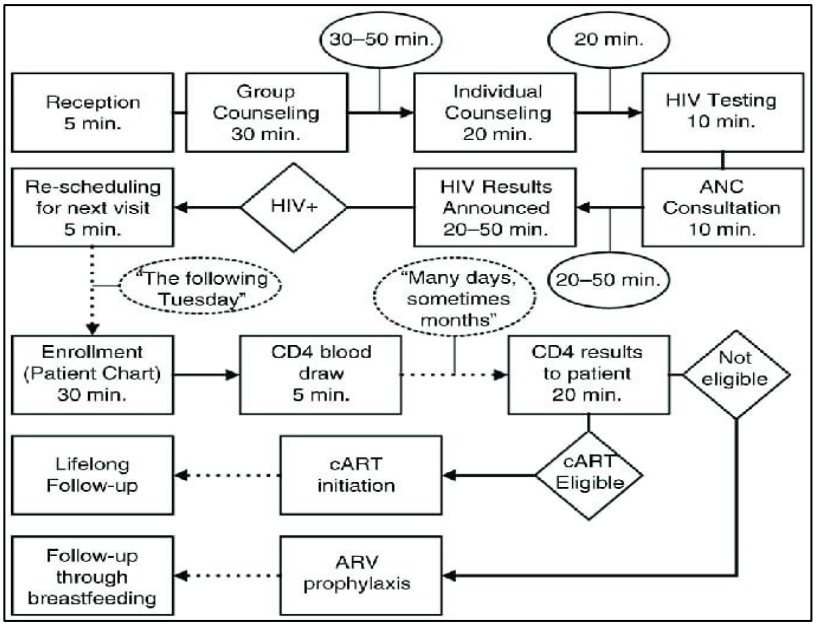
לרוב השלב הראשון לאחר השורש של העץ יצריך הכי הרבה מחקר כי הוא דורש הבנה של המרכיבים הכי גדולים לשלב הבא בפרויקט, וככל שנכנסים לכל רכיב בדרך כלל השאלה "מה השלב הבא" יהיה יותר ויותר ספציפי, עם יותר מידע מתאים באינטרנט.

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.

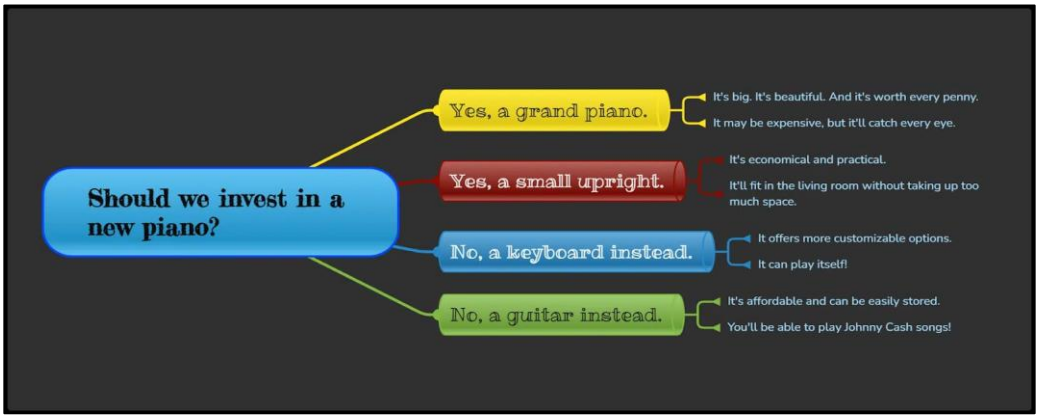
בדברים כמו מחקר, זה נחמד לחשוב על מפת עץ כשהמטרה הסופית שלנו היא משהו כמו "למצוא RCE באנדרואיד". לרוב גם אם נפרק את הרכיב שאנחנו חוקרים ונחפש את כל האפשרויות להגיע לתוצאה הזאת:

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

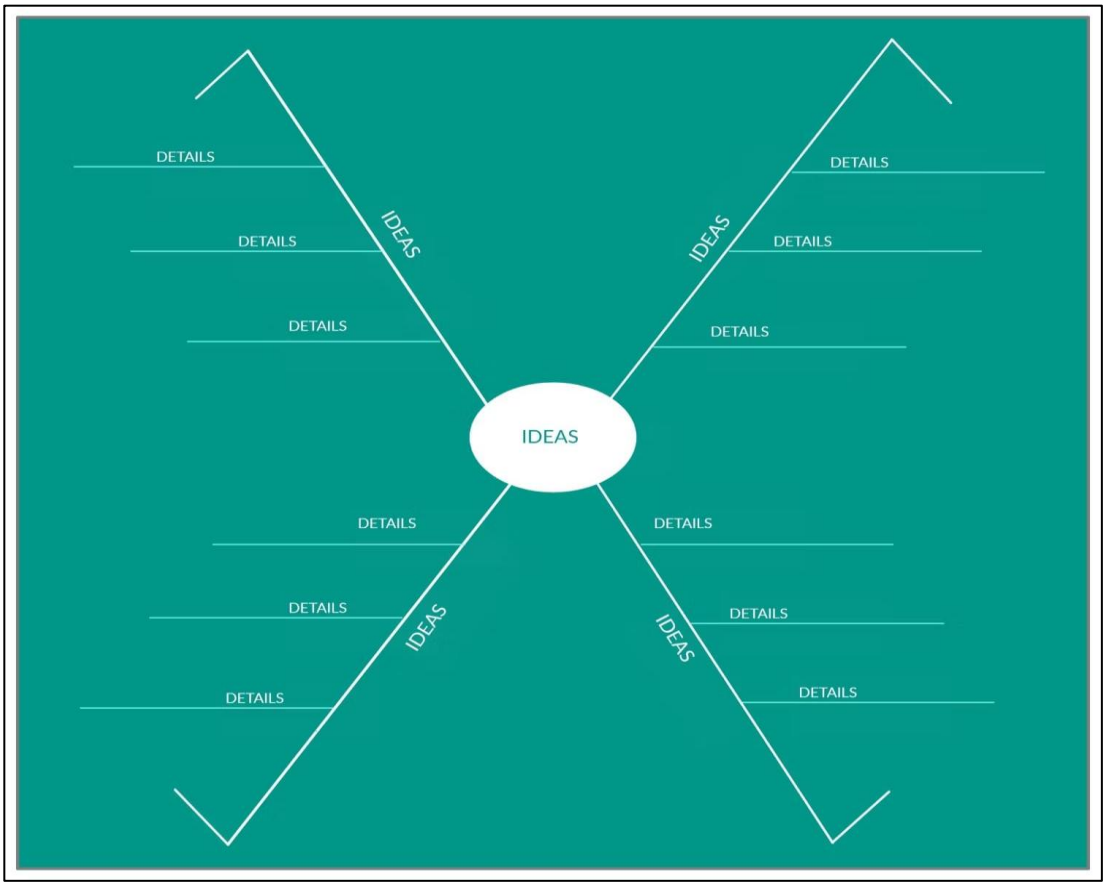
2. כדי שתהיה בן אדם שעובד לבד על חלק מפרויקט משמעותי, ראש הצוות ושאר ההנהלה שלך ירצו שיהיה אפשר לסמוך עליך. אם התשובה לכל בעיה היא ללכת לפי מה שהמנהל אומר בדיוק בלי יוצא דופן, כשתבוא בעיה טכנולוגית קצת שונה המתלמד יהפוך לגולם, כי הוא אף פעם לא ניסה לחשוב על צורת המחשבה שאותו מנהל חשב בה.

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.

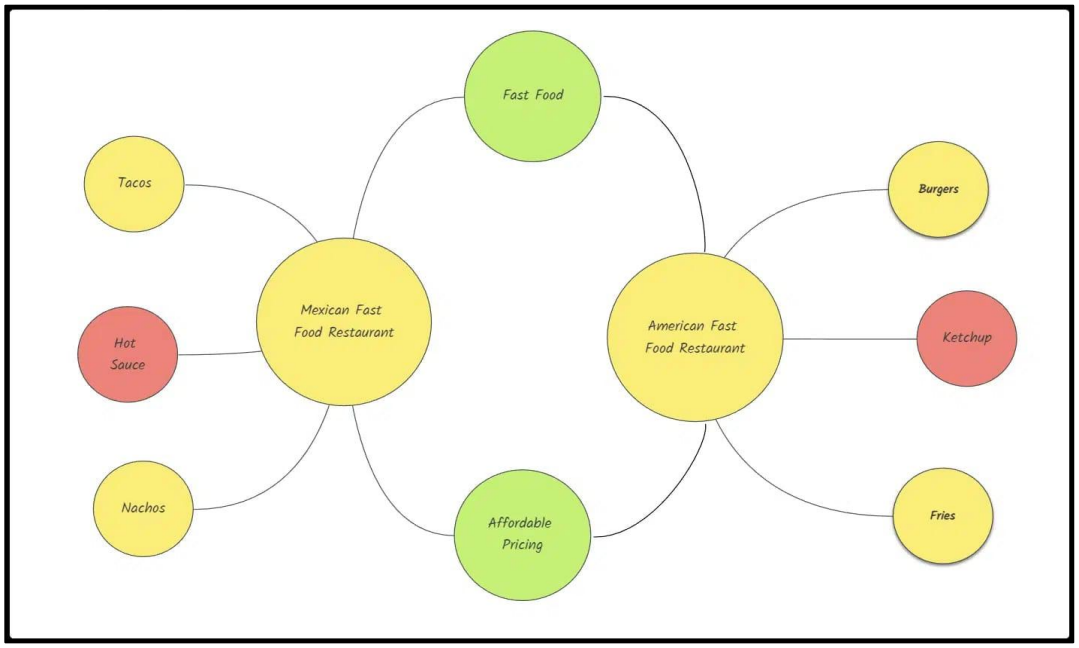
4. מפת עכביש:



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.

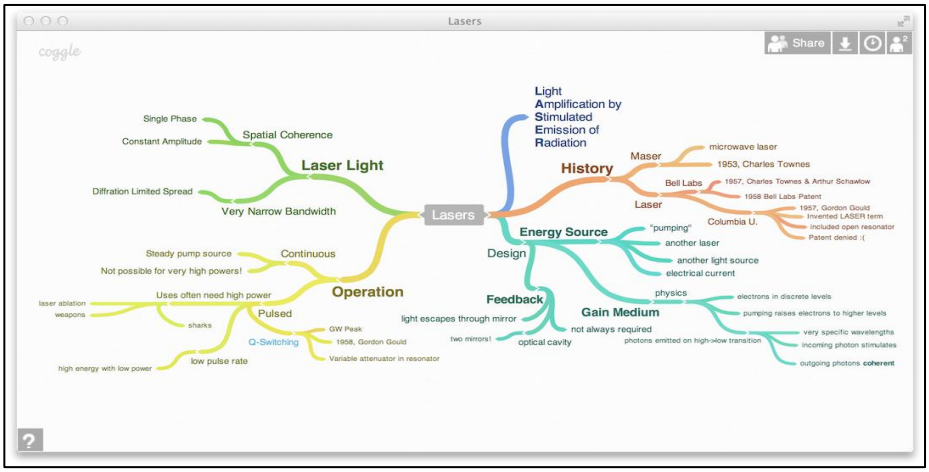
כלי עזר לתיעוד מחשבות וכיוונים

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.

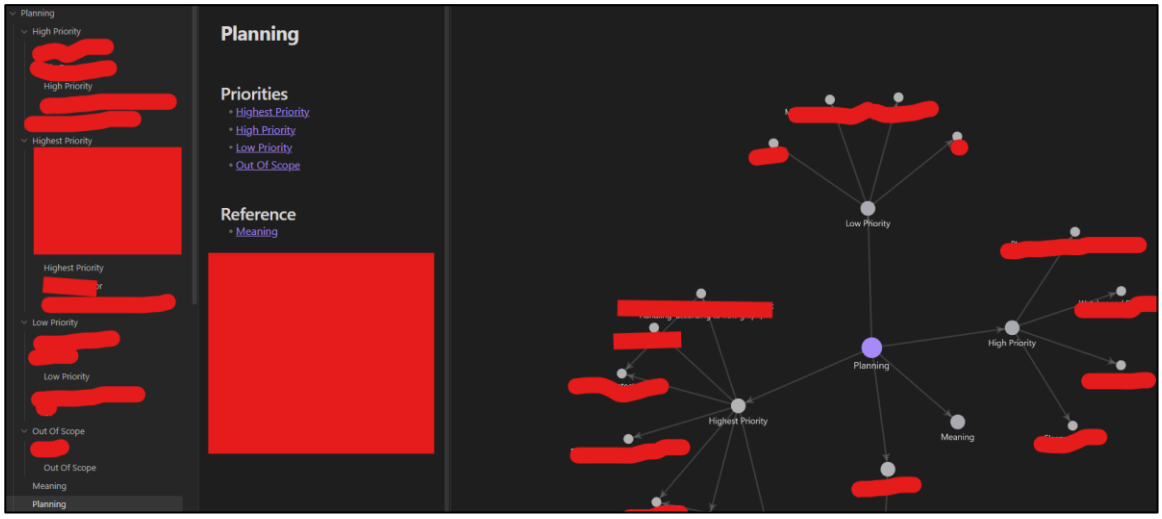


Obsidian - כלי תיעוד אישי חזק וגמיש, שנבנה סביב קבצי טקסט פשוטים בפורמט Markdown, בכלי הקבצים נשמרים מקומית על המחשב שלך (אלא אם תבחר לסנכרן), מה שהופך את הכלי למהיר, אמין ובשליטת המשתמש. כל "פתק" או מסמך שאתה יוצר הוא קובץ עצמאי, וניתן לקשר בין קבצים באמצעות סימון פשוט של [[שם הקובץ]], מה שהופך את אובסידיאן 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.

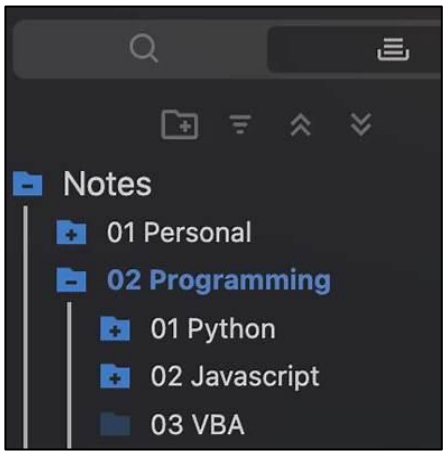
תוספים ב-obsidian

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 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.

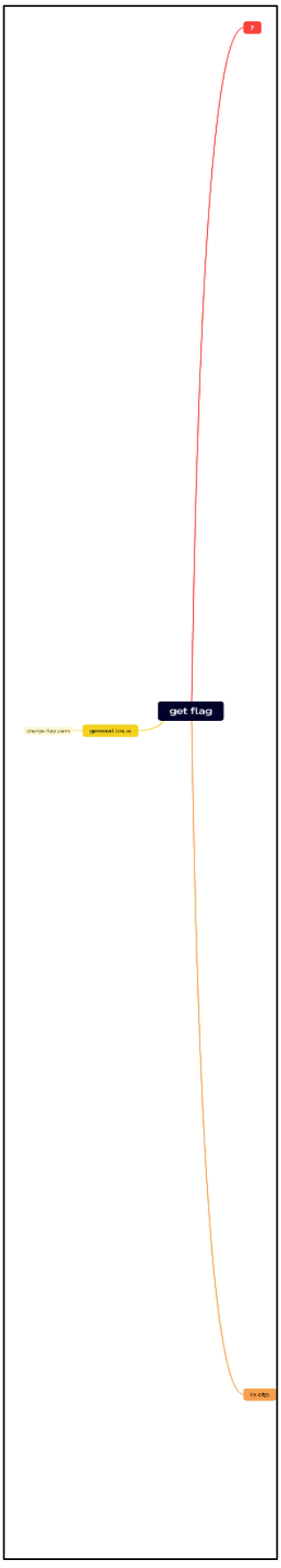
מאוד קל להתחיל להכנס ישר לכיוונים ספציפיים שעולים לראש ואז להתחיל לעבוד על ענפים ספציפיים, אבל חשוב באמת להתחיל מכמה שיותר גבוה וכללי, לרדת לאט-לאט בעץ ככל שאפשר ובכל שלב לחשוב טוב על כל הפיצולים החדשים שאפשר ליצור לעץ.

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.

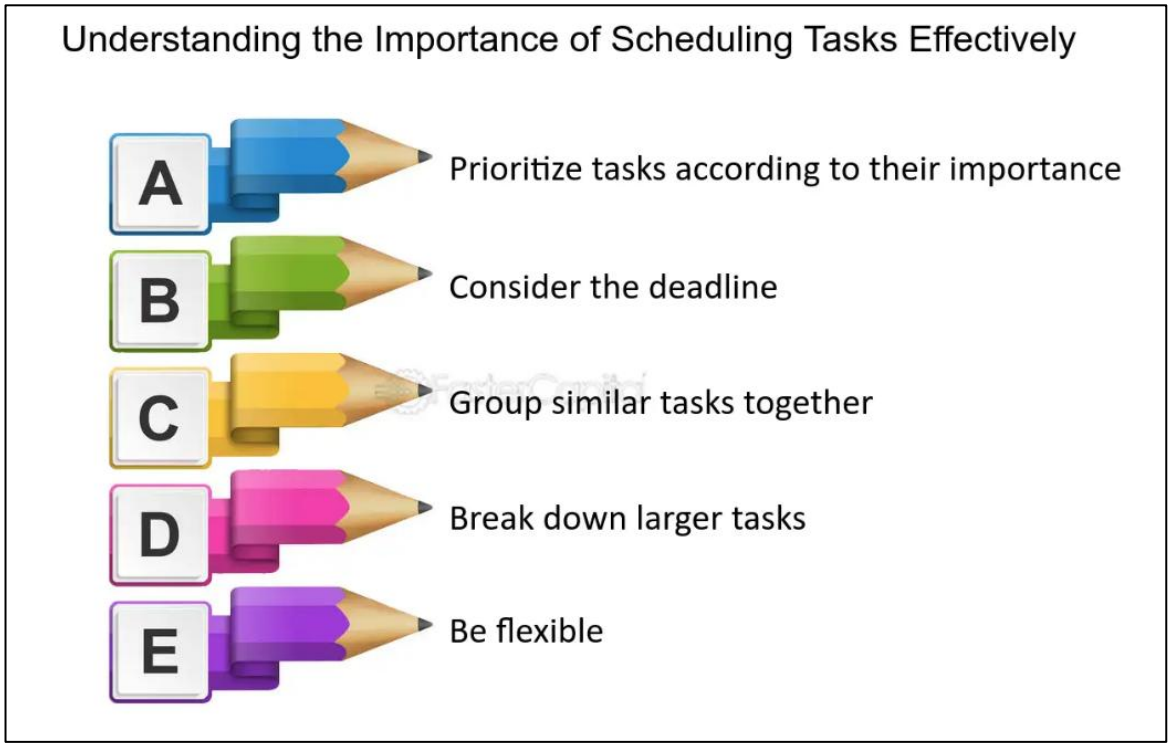
במקרה הזה הרכיבים שבחרתי לעבוד עליהם הם המערכת הכללית שבה האתגר נמצא (מערכת linux), קובץ האתגר בפנים (OTP) וקבצים אחרים שרלוונטיים לאתגר (במקרה הזה אין משהו כזה). זה כבר מצטמצם ל-2 רכיבים מרכזיים שנוכל להמשיך למפות בהמשך. חשוב מאוד לבוא בגישה שהמיפוי ההתחלתי לא בהכרח הרמטי, זה אומר שנשמור ראש פתוח לעוד 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.

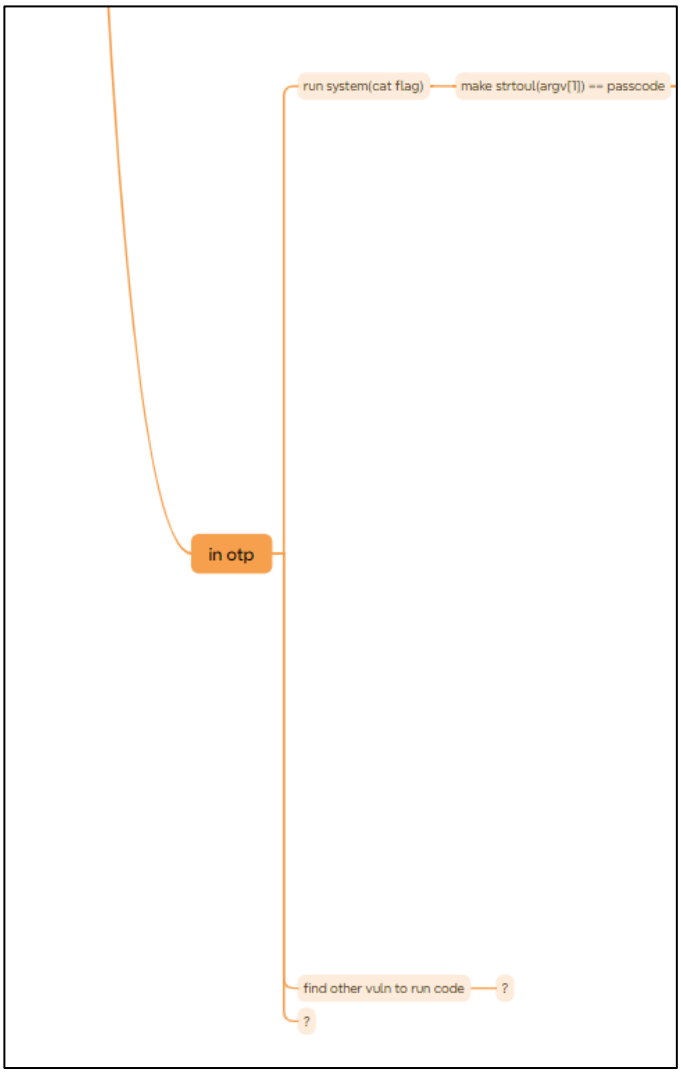


בגלל שהכיוון המרכזי כרגע הוא הקוד עצמו, אפשר להתחיל לחשוב מה הפעולות הכי ספציפיות וישירות שיכולות להוביל אותנו למטרה. גם כאן קל לצלול ישר למסלול שבו ה-passcode שאנחנו מביאים שווה למספר הרנדומלי, אבל זה כמה צעדים קדימה ולכן יש סיכוי שבדילוג כזה נדלג גם על כיוונים שיובילו לפתרון.

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:

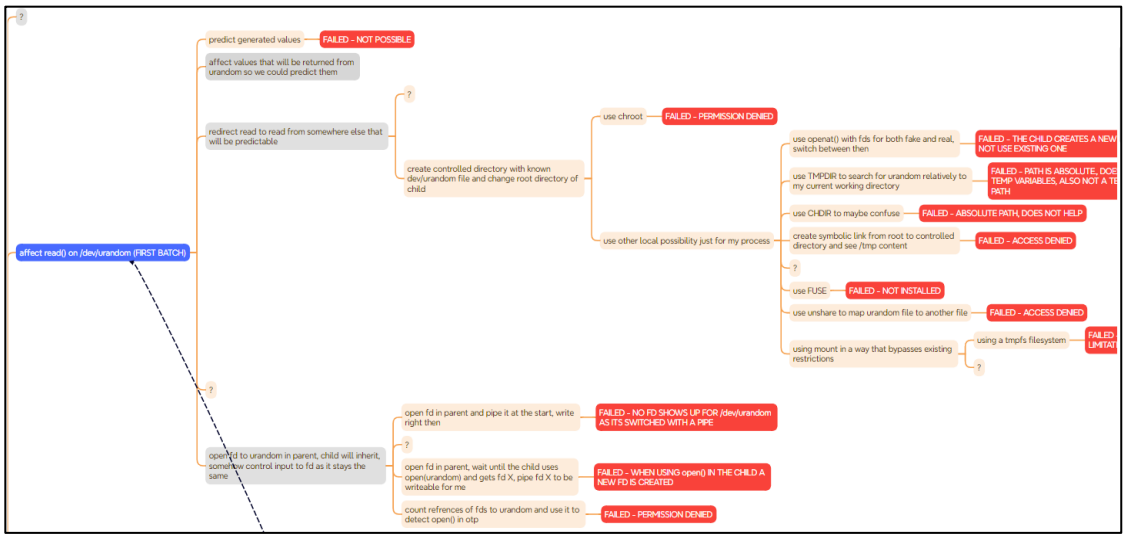
- האופצייה הראשונה מסובכת מצד המחקר והחיפוש ובגלל שזה אתגר נקודתי שלרוב עושים בו עבודת תשתיות טובה זה לא סביר שפה תהיה הבעיה
- האופצייה השנייה לא רלוונטית לאתגר בגלל שפה הערך נלקח מ-"/dev/urandom" והוא Int ככה שיש יותר מדי אופציות בשביל ניחוש אפקטיבי, בלי קשר שזה מוציא את הכיף מהאתגר

כאן החלק המעניין מגיע, וכדי להתמודד איתו הלכתי בגישה של פיצול הקוד לפי פונקציונליות שמקדימה את תנאי הנצחון:

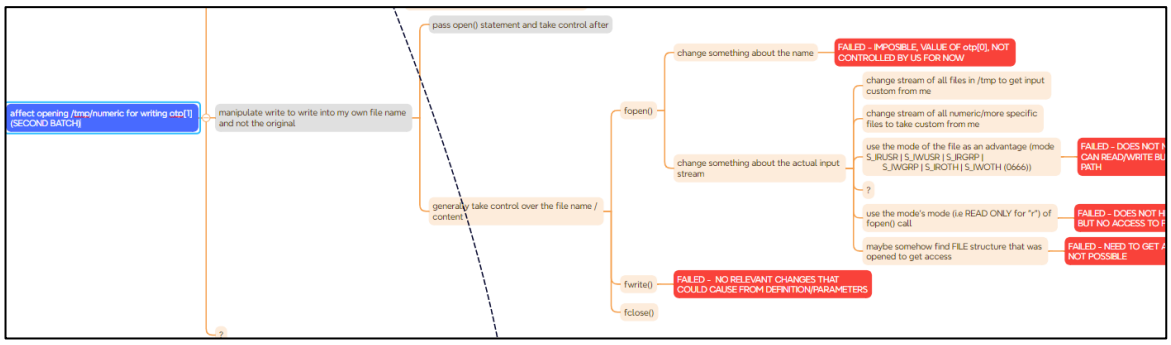
1. קבוצה ראשונה – קריאה של ערך רנדומלי בגודל Int מ-"/dev/urandom"
2. קבוצה שנייה – כתיבה של otp[1] ל-"/tmp/{otp[0]}"
3. קבוצה שלישית – פתיחת "/tmp/{otp[0]}" וקריאת הערך לתוך משתנה נוסף

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.



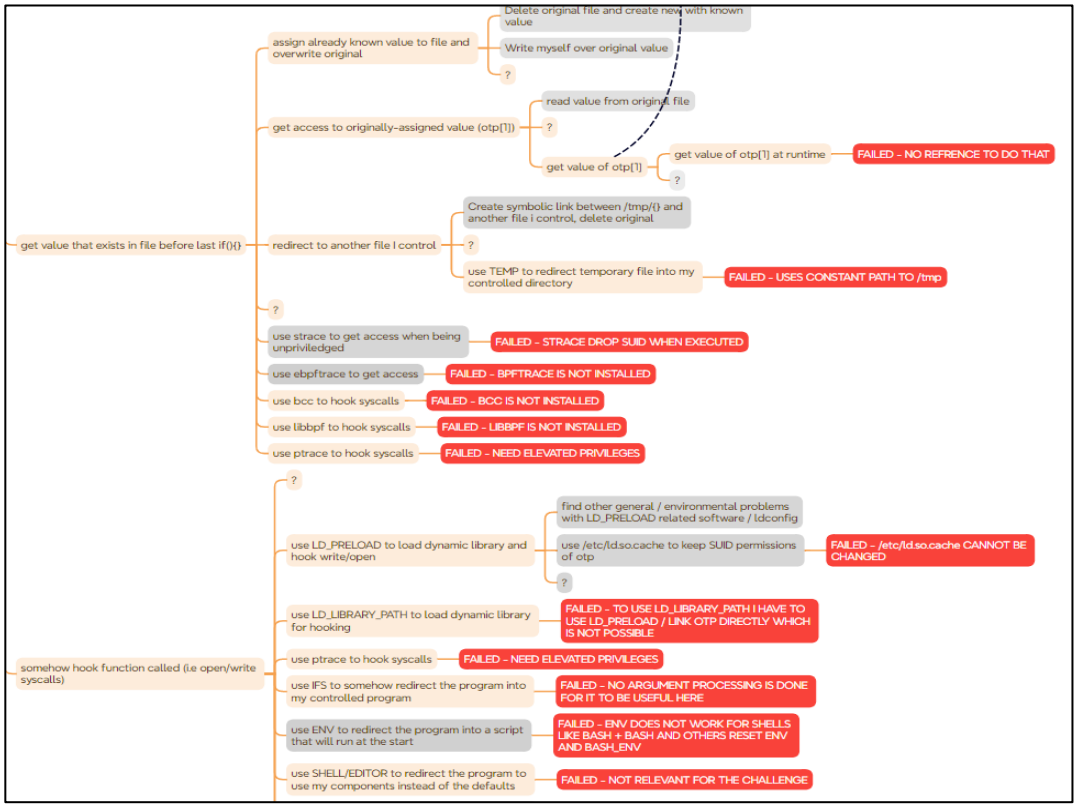
לאחר מכן המשכתי לחלק השני שבו פותחים את הקובץ שיכתב אליו הערך שצריך לנחש, ושם הכיוון היחיד שהגעתי אליו הוא דרך שינוי הפתיחה כך שהכתיבה תהיה בשליטתי, כלומר:

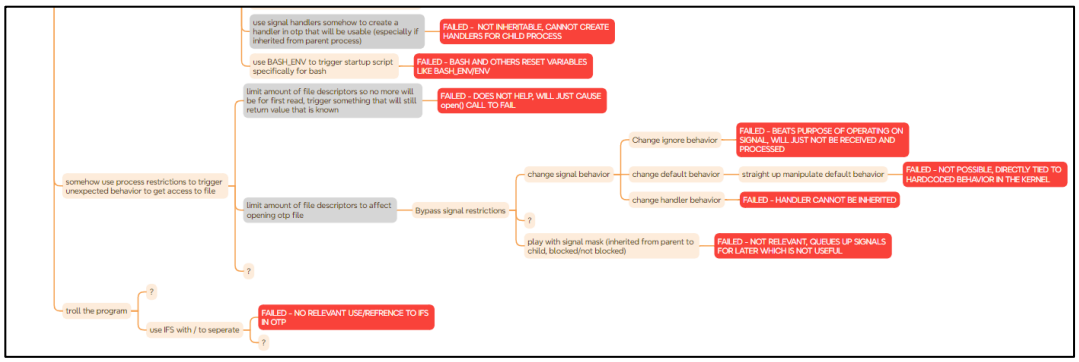
- יצירת קובץ בשם שאמורים ליצור בו פה (מתברר ככיוון לא טוב בגלל המחקר על קריאה מ-urandom)
- הצבעת המסלול שאמורים ליצור בו לקובץ אחר בשליטתי (גם כיוון לא טוב מאותה סיבה)

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?





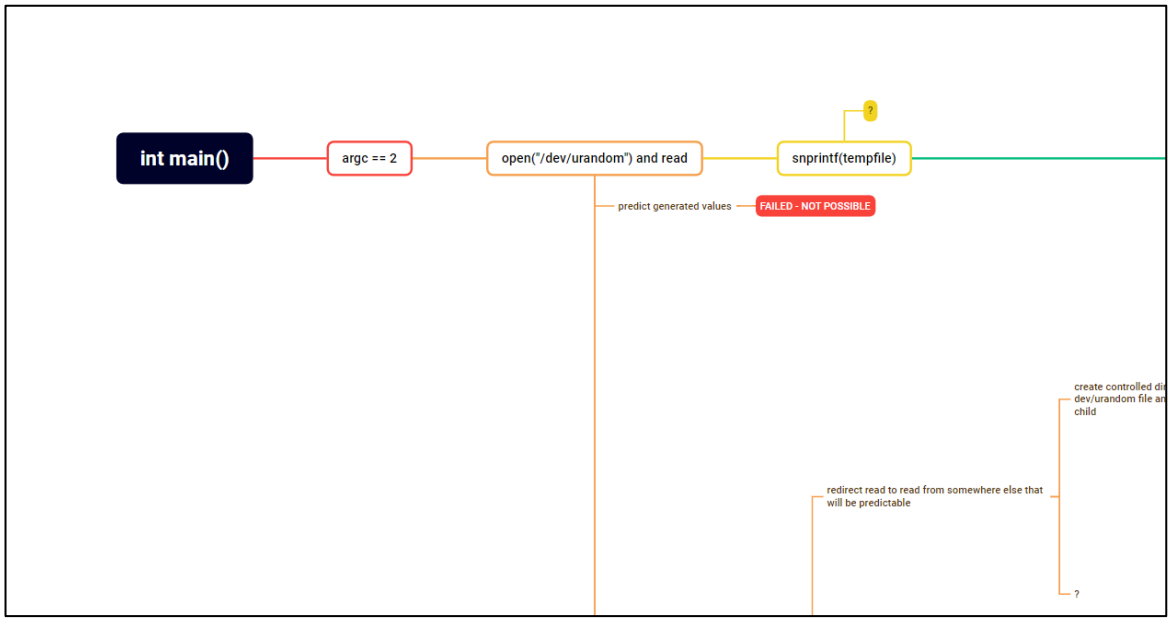
בדיעבד לא הצלחתי לחשוב על משהו שיכול לעזור לי לפתור את התרגיל, ובמובן מסוים אני חושב שאחת מהטעויות שעשיתי כאן שאפשר ללמוד מהם היא להפריד בין החלקים השונים במערכת – בהתחלה הסתכלתי על חלק ספציפי בקוד מבלי להצליב את ההשפעה של הסביבה ושל קטעי הקוד הבאים\קודמים, וכאן התרכזתי הרבה יותר בסביבה אבל בניתוק מהקוד.

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

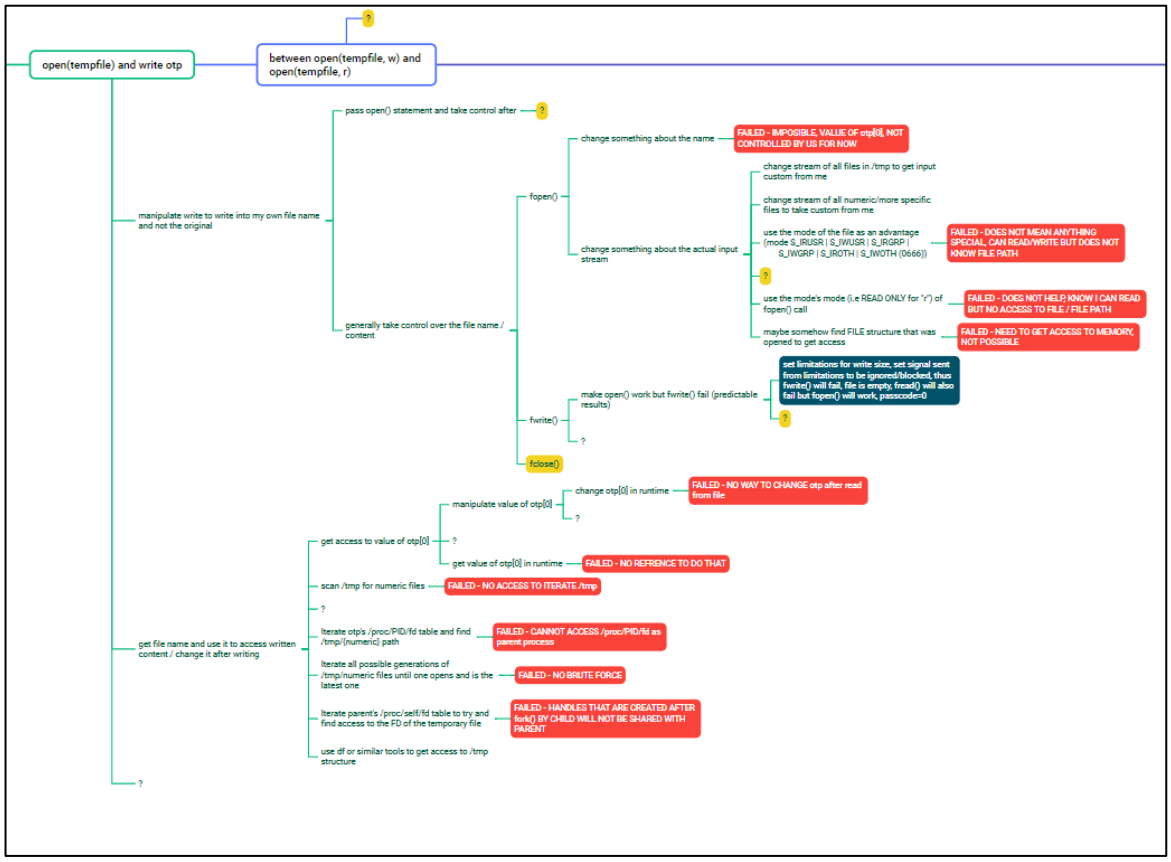
מפת הזרם מאוד פשוטה במובן הזה בתרגיל בגלל שכמות הקוד בו מאוד קטנה, עד כדי כך שאפשר לקטלג כל שורה \ שורה מרכזית בתור node:



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.

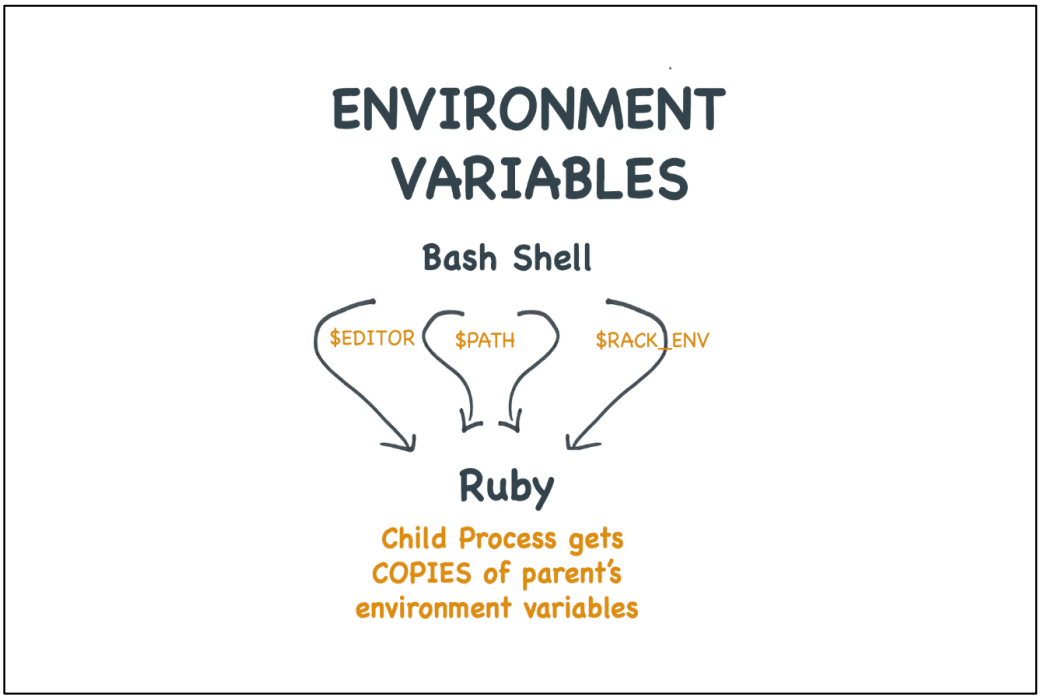
לכן ניסיתי לחשוב איך אני מגיע למצב שאני שולט על אותה פעולה כי היא כרגע מונעת ממני מלנצח את האתגר:

- 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 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/oztel/file-tree-alternative
- /https://blacksmithgu.github.io/obsidian-dataview
- https://www.reddit.com/r/ObsidianMD/comments/1b8sdnh/best_kanban_plugin_for_obsid/ian
- 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