פתרון השלב השני של BSidesTLV25 CTF

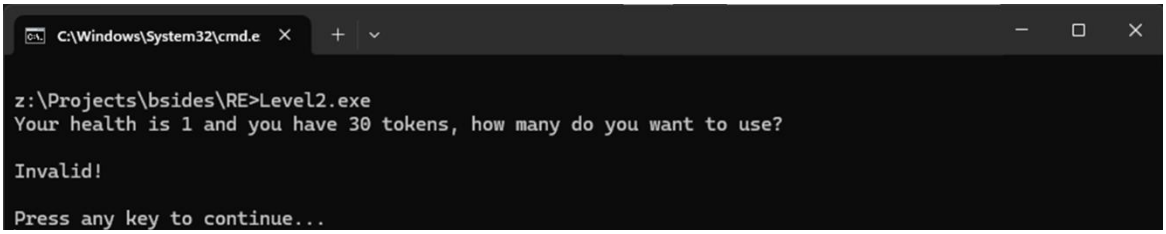
מאת רומן קויפמן

הקדמה

במהלך שבוע הסייבר בדצמבר 2025, BSidesTLV הוציאו סדרת אתגרים במספר נושאים שונים. אמנם לא לקחתי חלק בתחרות, אך רציתי להתנסות באתגר Reversing כלשהו בזמני הפנוי, והאתגר השני (המכונה "Level 2") היה נראה לי נחמד. מדובר בקובץ PE-32bit, נקרא לו: Level2.exe, ובעת הפעלתו מתקבלת המחרוזת הבאה:

Your health is 1 and you have 30 tokens,how many do you want to use?

לחיצה על Enter גורמת להדפסת ההודעה: "Invalid" ויציאה:



[איור 1 - הרצה ראשונית של האתגר]

נפתח את הקובץ ב-Binary Ninja ונחפש את המחרוזות בקוד ואת השימוש בהם. כך נגלה את הפונקציה הראשית:

```
00401a00    int32_t sub_401a00()

00401a0e        int32_t __saved_ebp
00401a0e        int32_t eax_1 = __security_cookie ^ &__saved_ebp
00401a15        int32_t var_28
00401a15        __builtin_memset(dest: &var_28, ch: 0, count: 0x1e)
00401a2e        int32_t var_13c = 0
00401a38        int32_t var_130 = 1
00401a42        int32_t var_138 = 0x1e
00401a42
00401a53        while (true)
00401a53            if (var_138 s<= 0)
00401be6                if (var_138 != 0 || var_130 != 0x64)
00401c08                    sub_401050("Nope...\n")
00401be6                else
00401bed                    sub_401050("Good job!!!!\n")
00401bf9                    sub_401890(&var_28)
00401bf9
00401c15                sub_401050("\nPress any key to continue...\n")
```

```
00401c1d            j___fgetwchar()
00401c22            break
00401c22
00401a5f        int32_t var_140_1 = var_138
00401a66        int32_t var_144_1 = var_130
00401a6c        sub_401050("
00401a6c            Your health is %d and you have %d tokens, how many do you want to use?\n")
00401a7e        int32_t var_140_2 = ___acrt_iob_func(0)
00401a7f        int32_t var_144_2 = 0x100
00401a8a        char var_128[0x100]
00401a8a        char (* var_148_1)[0x100] = &var_128
00401a8a
00401a95        if (sub_406c94() != 0)
00401a9d            int32_t var_12c
00401a9d            int32_t* var_140_3 = &var_12c
00401a95        if (sub_406c94() != 0)
00401a9d            int32_t var_12c
00401a9d            int32_t* var_140_3 = &var_12c
00401a9d
00401ab5            if (_fwprintf(&var_128, 0x4219b8) == 1)
00401b0c                if (var_12c s<= 0 || var_12c s> 0xa || var_12c s> var_138
00401b0c                        || var_130 + var_12c s> 0x64)
00401b13                    sub_401050("Invalid!\n")
00401b20                    sub_401050("\nPress any key to continue...\n")
00401b28                    j___fgetwchar()
00401b2f                    break
00401b2f
00401b40                var_130 += var_12c
00401b52                var_138 -= var_12c
00401b5e                int32_t edx_3
00401b5e                edx_3.b = var_12c.b
00401b64                *(&var_28 + var_13c) = edx_3.b
00401b71                var_13c += 1
00401b71
00401b99                for (int32_t i = 0; i u< 0x11; i += 1)
00401baf                    if (sx.d(*((i << 1) + &data_421900)) == var_130)
00401bbf                        var_130 = sx.d(*((i << 1) + &data_421901))
00401bc5                        i = 0
00401bc5
00401b99                continue
00401b99
00401abc            sub_401050("Invalid!\n")
00401ac9            sub_401050("\nPress any key to continue...\n")
00401ad1            j___fgetwchar()
00401ad8            break
00401ad8
00401c29        sub_401c32(eax_1 ^ &__saved_ebp)
00401c31        return 0
```

[איור 2 - הפונקציה הראשית לפני Reversing]

נקרא לפונקציה main, לאחר Reversing היא תיראה כך:

```
000a1a00    int32_t main(int32_t argc, char**& argv)

000a1a0e        int32_t __saved_ebp
000a1a0e        int32_t cookie = __security_cookie ^ &__saved_ebp
000a1a15        uint8_t path[0x1e]
000a1a15        __builtin_memset(dest: &path, ch: 0, count: 30)
000a1a2e        int32_t index = 0
000a1a38        int32_t health = 1
000a1a42        int32_t tokens = 30
000a1a42
000a1a53        while (true)
000a1a53            int32_t i
000a1a53
000a1a53            if (tokens s<= 0)
000a1be6                if (tokens != 0 || health != 100)
000a1c08                    printf(format: "Nope...\n", index, tokens, i, health)
000a1be6                else
000a1bed                    printf(format: "Good job!!!!\n", index, tokens, i, health)
000a1bf9                    decrypt(&path)
000a1bf9
000a1c15                printf(format: "\nPress any key to continue...\n", index, tokens, i, health)
000a1c1d                fgetwchar()
000a1c22                break
000a1c22
000a1a6c            printf(
000a1a6c                format: "Your health is %d and you have %d tokens, how many do you want to use?\n",
000a1a6c                health, tokens, index, tokens, i, health)
000a1a95            int32_t input
000a1a95            char buffer[0x100]
000a1a95            if (fgets(str: &buffer, count: 0x100, stream: ___acrt_iob_func(_std: stdin))
000a1a95                    != 0 && sscanf(&buffer, format: "%d", &input) == 1)
000a1b0c                if (input s<= 0 || input s> 10 || input s> tokens || health + input s> 100)
000a1b13                    printf(format: "Invalid!\n")
000a1b20                    printf(format: "\nPress any key to continue...\n")
000a1b28                    fgetwchar()
000a1b2f                    break
000a1b2f
000a1b40                health = health.b + input.b
000a1b52                tokens -= input
000a1b64                path[index] = input.b
000a1b71                index += 1
000a1b71
000a1b99                for (i = 0; i u< 17; i += 1)
000a1baf                    if (sx.d(ARRAY[i << 1]) == health)
000a1bbf                        health = sx.d(ARRAY[1 + (i << 1)])
000a1bc5                        i = 0
000a1bc5
000a1b99                continue
000a1b99
000a1abc            printf(format: "Invalid!\n")
000a1ac9            printf(format: "\nPress any key to continue...\n")
000a1ad1            fgetwchar()
000a1ad8            break
000a1ad8
000a1c29        cookie_check(cookie ^ &__saved_ebp)
000a1c31        return 0
```

[איור 3 - הפונקציה הראשית לאחר Reversing]

ניתן להגיע למספר תובנות:

- מספר ה-health ההתחלתי הינו 1
- מספר ה-tokens ההתחלתי הינו 30
- בכל הזנת קלט (מספר הטוקנים לשימוש), המספר חייב להיות בין 1 ל 10 (כולל)
  - אחרת יודפס Invalid והמשחק יסתיים
- בנוסף, יודפס Invalid והמשחק יסתיים אם:
  - הקלט גדול ממספר הטוקנים שנשאר
  - מתקיים health + tokens > 100
  - מתקיים tokens = 0 **and** health ≠ 100

- **בכל הזנת מספר כקלט מתבצע:**
  - שמירת הקלט המספרי במערך קלטים לפי אינדקס רץ (נקרא לו path)
  - החסרת טוקנים tokens-=input
  - הוספת חיים health+=input
  - מעבר על מערך בתים Array באורך 34. אם הערך **באינדקס הזוגי** שווה לכמות ה-health הנוכחי, תבוצע השמה של הערך בתא **האי-זוגי** הבא אחריו אל health וכן יאופס האינדקס. כך עד שלא יהיו התאמות יותר באינדקסים הזוגיים ונגיע לסוף המערך.

```
000c1900  uint8_t ARRAY[34] =
000c1900  {
000c1900      [ 0] = 3
000c1901      [ 1] = 65
000c1902      [ 2] = 4
000c1903      [ 3] = 93
000c1904      [ 4] = 11
000c1905      [ 5] = 21
000c1906      [ 6] = 12
000c1907      [ 7] = 2
000c1908      [ 8] = 14
000c1909      [ 9] = 52
000c190a      [10] = 27
000c190b      [11] = 9
000c190c      [12] = 31
000c190d      [13] = 21
000c190e      [14] = 39
000c190f      [15] = 28
000c1910      [16] = 40
000c1911      [17] = 97
000c1912      [18] = 49
000c1913      [19] = 33
000c1914      [20] = 53
000c1915      [21] = 43
000c1916      [22] = 55
000c1917      [23] = 36
000c1918      [24] = 56
000c1919      [25] = 38
000c191a      [26] = 67
000c191b      [27] = 76
000c191c      [28] = 78
000c191d      [29] = 69
000c191e      [30] = 98
000c191f      [31] = 27
000c1920      [32] = 99
000c1921      [33] = 74
000c1922  }
```

[איור 4 - מערך ARRAY]

לכל הזנת מספר כקלט נקרא **צעד**. צעד יחיד יתניע את כל התהליך שמפורט לעיל. רק כאשר נגיע למספר tokens=0 וכן health=100, נעבור לשלב הבא בתהליך decrypt(&path). כאשר נגיע ל-tokens=0 מערך הקלטים path יכיל את הצעדים שלנו.

כעת המשימה ברורה. עלינו למצוא **מסלול צעדים** שיוביל אותנו לשלב הבא באתגר. לפני שנתעמק בדרך הפתרון למשימה, נציץ בשלב הבא. להלן פונקציית decrypt לפני Reversing:

```
00401890    int32_t sub_401890(char* arg1)

0040189b        int32_t __saved_ebp
0040189b        int32_t eax_1 = __security_cookie ^ &__saved_ebp
004018a0        char var_68
004018a0        __builtin_strncpy(dest: &var_68, src: "j/&L", count: 4)
004018b0        char var_64
004018b0        __builtin_memcpy(dest: &var_64,
004018b0            src: "
004018b0                \xc4\xc0\x76\x70\x90\xd1\x87\x9b\x2c\x4b\x91\x20\x8a\x2a\x7f\x2e\xcb\x1d",
004018b0            count: 0x12)
004018f8        char var_52 = 0x43
004018fc        char var_51 = 0x39
00401915        uint8_t var_4c[0x19]
00401915        sub_401840(arg1, _strlen(arg1), &var_4c)
00401954        char var_33
00401954        char var_32
00401954        char var_31
00401954        int32_t result
00401954
00401954        if (sx.d(var_33) == 0x7c || sx.d(var_32) == 3 || sx.d(var_31) == 0x92)
00401988            for (int32_t i = 0; i s< 0x18; i += 1)
0040199f                var_4c[i] ^= (&var_68)[i]
0040199f
004019c1            var_4c[0x18] = 0
004019cb            sub_401050("Here's your flag:\n")
004019d6            uint8_t (* var_70_1)[0x19] = &var_4c
004019dc            result = sub_401050("BSidesTLV2025{%s}\n")
00401954        else
0040195b            sub_401050("You did it but there is a better solution!\n")
00401968            result = sub_401050("Try combining steps and find a shorter input.\n")
00401968
004019e9        sub_401c32(eax_1 ^ &__saved_ebp)
004019f1        return result
```

[איור 5 - פונקציית decrypt לפני Reversing]

ולאחר Reversing:

```
000a1890    void decrypt(uint8_t* path)

000a189b        int32_t __saved_ebp
000a189b        int32_t cookie = __security_cookie ^ &__saved_ebp
000a18a0        uint8_t key[0x18]
000a18a0        // "\x6a\x2f\x26\x4c\xc4\xc0\x76\x70\x90\xd1\x87\x9b\x2c\x4b\x91\x20\x8a\x2a\x7f\x2e\xcb\x1d\x43\x39"
000a18a0        __builtin_memcpy(dest: &key, src: "j/&L…", count: 24)
000a1915        uint8_t flag[0x20]
000a1915        // The flag is SHA256 of the correct path, hence sized 32 bytes
000a1915        do_sha256(input: path, len: _strlen(path), output_hash: &flag)
000a1915
000a1954        if (sx.d(flag[25]) == 0x7c || sx.d(flag[26]) == 3 || sx.d(flag[27]) == 0x92)
000a1988            for (int32_t i = 0; i s< 0x18; i += 1)
000a199f                flag[i] ^= key[i]
000a199f
000a19c1            flag[24] = 0
000a19cb            printf(format: "Here's your flag:\n", 0x18)
000a19dc            printf(format: "BSidesTLV2025{%s}\n", &flag)
000a1954        else
000a195b            printf(format: "You did it but there is a better solution!\n")
000a1968            printf(format: "Try combining steps and find a shorter input.\n")
000a1968
000a19e9        cookie_check(cookie ^ &__saved_ebp)
```

[איור 6 - פונקציית decrypt לאחר Reversing]

מספר מסקנות על פונקציית decrypt:

- מבוצעת חתימת SHA256 על מסלול הקלטים (path) אשר פותר את הבעיה בשלב הראשון
- אם יש התאמה אחת מ-3, באינדקסים 25,26,27 בחתימה, יבוצע XOR על 24 הבתים הראשונים בחתימה עם מפתח בגודל 24 בתים. התוצאה היא הדפסת הדגל בפורמט {%s}BSidesTLV2025.
- אם אין אף התאמה, יודפס כי ניתן למצוא פתרון **קצר יותר**

לפונקציית חיתום ה-SHA256, קראתי do_sha256. פונקציה זו משתמשת באבני בניין מ-openssl. אך זה לא אקוטי לפתירת האתגר.

```
000a1840    void do_sha256(uint8_t* input, int32_t len, uint8_t* output_hash)

000a184b        int32_t __saved_ebp
000a184b        int32_t cookie = __security_cookie ^ &__saved_ebp
000a1854        struct SHA256_CTX ctx
000a1854        SHA256_Init(c: &ctx)
000a1868        SHA256_Update(c: &ctx, data: input, len)
000a1878        // // output_hash gets h[0-7]
000a1878        SHA256_Final(c: &ctx, md: output_hash)
000a1885        cookie_check(cookie ^ &__saved_ebp)
```

[איור 7 - פונקציית do_sha256 לאחר Reversing]

זיהיתי את SHA256_Init על ידי חיפוש הערכים הקבועים שבתוך הפונקציה (הצ'אט עשה עבודה נהדרת בזיהוי שלהם, ולמעשה מדובר ב-IV של SHA256):

```
000a14e0    void SHA256_Init(struct SHA256_CTX* c)

000a14e6        c->num = 0
000a14f4        c->Nh = 0
000a14f7        c->md_len = 0
000a1505        c->h[0] = 0x6a09e667
000a1518        c->h[1] = 0xbb67ae85
000a152a        c->h[2] = 0x3c6ef372
000a153d        c->h[3] = 0xa54ff53a
000a1550        c->h[4] = 0x510e527f
000a1563        c->h[5] = 0x9b05688c
000a1576        c->h[6] = 0x1f83d9ab
000a1589        c->h[7] = 0x5be0cd19
```

[איור 8 - פונקציית SHA256_Init לאחר Reversing]

לעניינו, אמנם אנחנו לא מבינים את המשמעות של שלושת הבתים שמושווים לחתימה באינדקסים 25,26,27 אך מהמחרוזות המודפסות אנו כן מקבלים רמז כי עלינו לשאוף למצוא את המסלול הקצר ביותר. ניסיונות פיצ'פוץ' וכן מעקף תנאים בעזרת Debugging לא יעזרו לנו מפני שפשוט יודפסו ערכי זבל עבור הדגל.

אני מניח שכולנו בשלב כזה או אחר שמענו על אלגוריתם BFS (Breadth First Search) אשר מוצא את המסלול הקצר ביותר מנקודת התחלה לנקודת סיום. במקרה שלנו זה קצת יותר מורכב. לא יהיה מספיק להגדיר את צמתי הגרף כערכי ניצול הטוקנים בלבד. נגדיר שכל צומת בגרף הוא שילוב של שני הפרמטרים (health,tokens left). כל צומת כזה מהווה **מצב** ולמעשה אנו בונים מכונת מצבים סופית (FSM). כל מצב מאגד בתוכו את כל המידע הדרוש לתאר את הקונפיגורציה הנוכחית של המערכת. הקשתות בגרף הינם המעברים בין המצבים. כל קשת מהווה צעד בודד וכל משקל קשת הוא "1" למען האחידות.

אז אחרי שהגדרנו את הגרף שלנו, אנחנו רוצים למצוא מסלול פותר המקשר בין צומת ההתחלה (1,30) לצומת הסופית (100,0). קיימת משפחת אלגוריתמים בשם State Space Search שיודעת לפתור בעיות שכאלו - כלומר מציאת מסלולים מנקודה A לנקודה B תוך התחשבות במצבי המערכת השונים. המשפחה משלבת בתוכה אלגוריתמים פופולריים כמו BFS, DFS ועוד. במקרה שלנו, אנו מעוניינים במסלול הקצר ביותר, ולכן נבחר להשתמש באלגוריתם **State Space BFS**. למעוניינים בהעמקה, מאמר נחמד שנתקלתי בו בנושא זה:

https://codeforces.com/blog/entry/144451

אם כך, נממש את האלגוריתם בפייתון למציאת המסלול הקצר ביותר, תוך התחשבות במצבים השונים:

```python
""" bfs.py - State Space Search (BFS) """
from collections import deque

ARRAY = [3, 65, 4, 93, 11, 21, 12, 2, 14, 52, 27, 9, 31, 21, 39, 28, 40, 97,
49, 33, 53, 43, 55, 36, 56, 38, 67, 76, 78, 69, 98, 27, 99, 74]

def update_health(health: int) -> int:
    i = 0
    while i < 17:
        index = i << 1
        if health == ARRAY[index]:
            health = ARRAY[index+1]
            i = 0
        i += 1
    return health

def print_path(path: list[int]):
    print(" --> ".join(f"{i}" for i in path), ';', sep="")



def state_space_bfs(max_path_length):
    queue = deque([(1, 30, [])])
    visited = set()
```

```
    while queue:
        health, tokens_left, path = queue.popleft()
        state = (health, tokens_left)
        if state in visited or len(path) > max_path_length:
            continue
        visited.add(state)

        if tokens_left == 0:
            if health == 100:
                print_path(path)
                return
            continue

        max_tokens = tokens_left if tokens_left < 10 else 10
        for tokens in range(1, max_tokens+1, 1):
            new_health = update_health(health + tokens)
            new_path = path + [tokens]
            queue.append((new_health, tokens_left - tokens, new_path))

if __name__ == "__main__":
    state_space_bfs(30)
```

[איור 9 - State Space BFS על מצבי (health,tokens,path)]

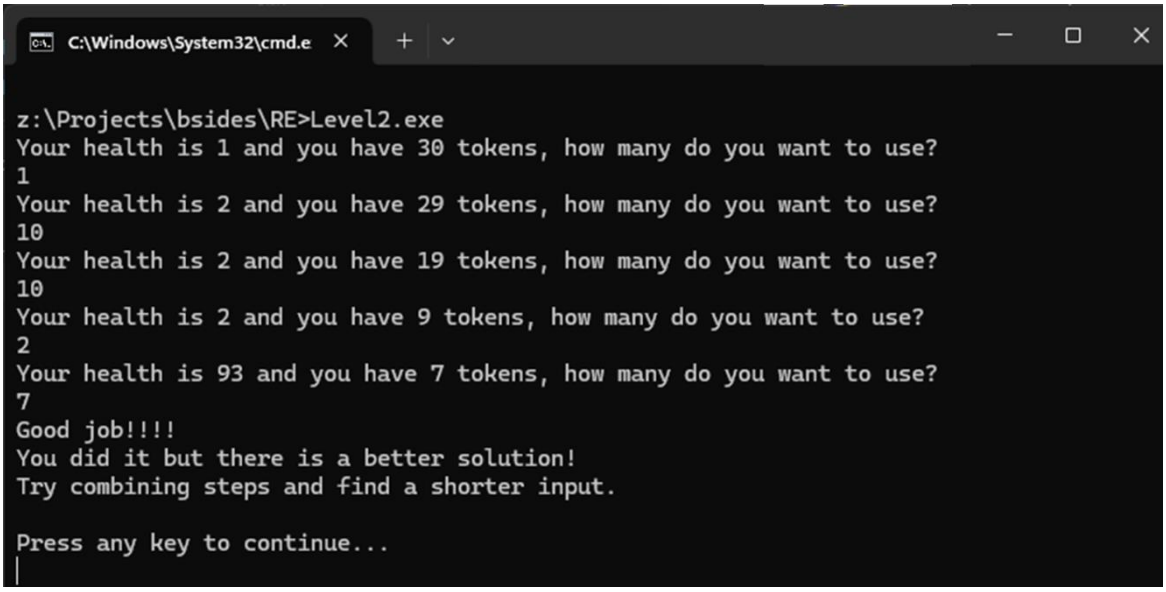
אז מה כתבנו בקוד?

- הגדרנו את המערך כפי שהופיע בקוד
- הגדרנו פונקציה update_health שתסמלץ את לוגיקת עדכון ה-health
- הגדרנו פונקציה print_path להדפסת הנתיב בסינטקס mermaid כדי שיהיה אפשר לרנדר כגרף במקרה הצורך (למשל עם https://mermaid.live)
- וכמובן הגדרנו את הפונקציה state_space_bfs אשר מקבלת כארגומנט את אורך המסלול המקסימלי - לא נרצה לחפש מסלולים ארוכים ממנו. בתור התחלה אפשר להגדיר אותו להיות 30 (מקסימום הצעדים - כמספר הטוקנים).

בפונקציה state_space_bfs נשתמש במבנה הנתונים deque. זה מעין מחסנית / תור שניתן להוציא איברים משני הצדדים בסיבוכיות O(1) אנו נשתמש במבנה זה להכנסת איברים בצד אחד והוצאת איברים בצד השני. כל איבר ב-deque הוא למעשה **מצב** ומוגדר כ-(health,tokens_left,current_path). כך, תוך כדי התקדמות במצבים השונים, אנו נגלה עוד מצבים חדשים שנוסיף אותם באופן דינאמי ל-deque. כאשר לא יישארו מצבים חדשים לטפל בהם, המבנה יתרוקן והפונקציה תסתיים. במהלך הפונקציה אם נתקלים במצב שכבר ביקרנו בו, לא נטפל בו אלא נמשיך לאיבר הבא במבנה הנתונים.

כאשר נגיד לאיבר שבו tokens_left=0 וגם health=0, אז נדפיס את המסלול המורכב מהטוקנים שהשתמשנו בכל צעד ונצא מהפונקציה. הרצה של הסקריפט תניב את הפלט 7→2→10→10→1.

נזין את הסדרה שקיבלנו לתכנית Level2.exe ונקבל:



[איור 10 - הזנה של המסלול הקצר ביותר לתכנית]

מוזר. התכנית מדפיסה כי קיים נתיב קצר יותר. **אבל זה לא ייתכן מתמטית כי BFS בהגדרה מוצא את המסלול הקצר ביותר**. אם כך, אולי קיימים עוד נתיבים באורך 5?

נשנה את הקוד קצת. נמחק את return בעת מציאת מסלול, וכן נמסך את הבדיקה של ביקור בצמתים שכבר ביקרנו בהם. בנוסף, נעדכן את האורך המקסימלי להיות 5 (אחרת נקבל הרבה מאוד סוגי מסלולים ארוכים). נריץ את הקוד, ונקבל מספר מסלולים מתאימים באורך 5:

$$1 \to 10 \to 10 \to 2 \to 7;$$
$$4 \to 7 \to 10 \to 2 \to 7;$$
$$5 \to 6 \to 10 \to 2 \to 7;$$
$$6 \to 5 \to 10 \to 2 \to 7;$$
$$7 \to 4 \to 10 \to 2 \to 7;$$
$$8 \to 3 \to 10 \to 2 \to 7;$$
$$9 \to 2 \to 10 \to 2 \to 7;$$

אם נרצה לצייר את המסלולים ב-mermaid, כדאי להוסיף לכל ערך בנתיב את health וגם את מספר הצעדים שביצענו עד כה. כך נבדיל בציור בין המצבים השונים, ולא נקבל לולאות מבלבלות. ניתן לעדכן את הקוד על ידי הגדרת path בצורה הבאה:

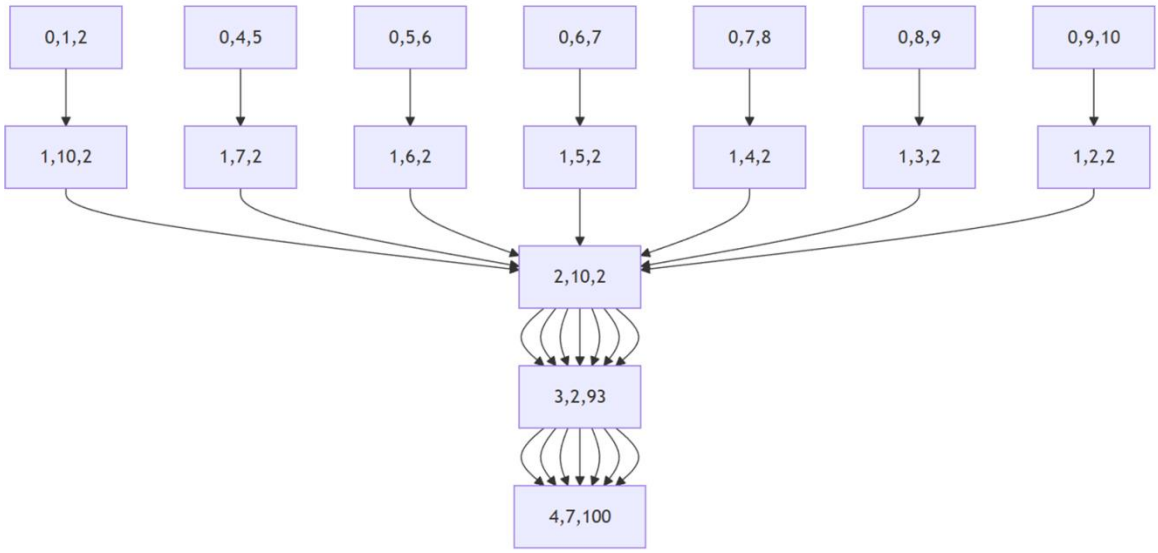
```
# in state_space_bfs
new_path = path + [(len(path), tokens, new_health)]

def print_path(path: list[tuple[int,int,int]]):
    print(" --> ".join(f"{i},{j},{k}" for i,j,k in path), ';', sep="")
```

[איור 11 - עדכון הנתיב עבור ציור צמתים]

למעשה הגדרנו כי כל צומת בגרף יהיה מהסוג (step,tokens,new health) כאשר:

- step זהו מספר הצעדים שעשינו לפני, או אינדקס הצעד הנוכחי אם תרצו.
- tokens זהו מספר הטוקנים לשימוש.
- new health זהו ערך החיים לאחר שימוש הטוקנים שהזנו.

הצמתים החדשים שהגדרנו משמשות רק למטרות הציור. אלגוריתם ה-BFS ימשיך לעבוד על המצבים מהסוג (health,tokens left):



[איור 12 - רינדור המסלולים הקצרים ביותר (אורך 5) על ידי mermaid.live]

נחזור להדפסה הרגילה (רק מספר הטוקנים לשימוש). כעת נרצה להזין את המסלולים לתכנית, אז נשתמש ב-subprocess. נגדיר פונקציית activate אשר בעת מציאת מסלול מתאים, תפעיל את Level2.exe ותשלח את המסלול, עם Enter ('\n') לאחר כל צעד:

```python
import subprocess

def activate(path: list[int]) -> str:
    process = subprocess.Popen("Level2.exe", stdin=subprocess.PIPE,
stdout=subprocess.PIPE, text=True)
    output, _ = process.communicate("\n".join(map(str, path)))
    print(output)


# in state_space_bfs
# ...
if tokens_left == 0:
    if health == 100:
        print_path(path)
        activate(path)  # <--- new
    continue
# ...
```

[איור 13 - פונקציית activate]

ונריץ!

בסוף כל מסלול, עבור כל אחד מהמסלולים הנ"ל, יודפס:

```
Good job!!!!
You did it but there is a better solution!
Try combining steps and find a shorter input.

Press any key to continue...
```

[איור 14 - הדפסה לאחר כל סיום מסלול שפותר את השלב הראשון באורך 5]

אף אחד מהמסלולים עד כה לא גרם להדפסת הדגל. כאן כבר הייתי מבולבל. אז יצרתי קשר עם מחבר האתגר שהודה כי קיימת טעות באתגר וכי קיימים מסלולים נוספים שפותרים את החידה, ושהמסלול הנכון הינו **באורך 7**. בנוסף המחבר ציין כי המטרה היא לא לרוורס את הקריפטו (חתימת ה-SHA256).

אם נדפיס את המסלולים שפותרים את השלב הראשון עד אורך 7 (כולל) נקבל רשימה ארוכה מאוד. **קיימים 792 מסלולים מתאימים!** (נבדק על ידי הוספת מונה).

אז יש תקלה באתגר, ויחסתי חשיבות גבוהה מדי להדפסה המבלבלת כי קיים מסלול קצר יותר, אבל עדיין רציתי למצוא את המסלול הנכון.

נחזור לסקריפט ונוסיף פונקציה שתקבל כקלט את המסלול ותחזיר חתימת SHA256 (באורך 32 בתים):

```python
import hashlib
def sha256(path: list[int]) -> bytes:
    m = hashlib.sha256()
    m.update(bytes(path))
    return m.digest()
```

[איור 15 - חיתום SHA256 ב-Python]

בנוסף, נגדיר פונקציה שתבדוק את שלושת ההתאמות שצוינו לעיל:

```python
def check_hash(digest: bytes) -> bool:
    return (digest[25] == 0x7C) or (digest[26] == 3) or (digest[27] == 0x92)
```

[איור 16 - בדיקת ה-Hash]

נעדכן את פונקציית state_space_bfs לחתום את המסלולים המתאימים, ואם קיימת לפחות התאמה אחת מבין ה-3, רק אז נקרא ל-activate:

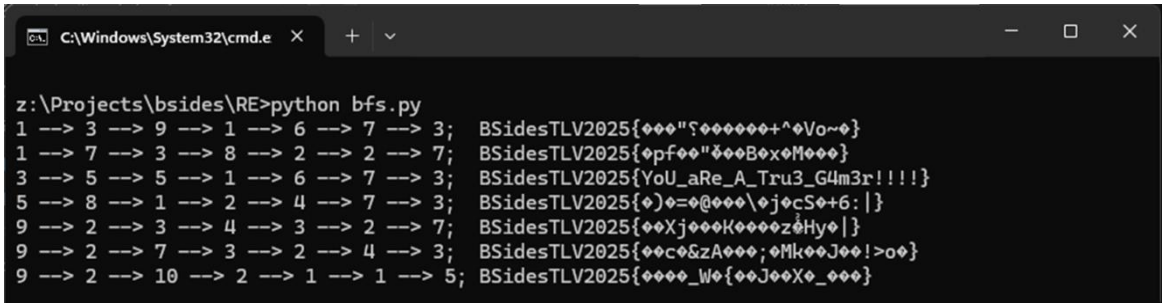
```python
# in state_space_bfs

if tokens_left == 0:
    if health == 100:
        digest = sha256(path)
        if check_hash(digest):
            activate(path)
    continue
```

בנוסף, נעדכן את פונקציית activate להדפיס רק את הדגל:

```python
import subprocess, re
def activate(path: list[int]) -> str:
    process = subprocess.Popen("Level2.exe", stdin=subprocess.PIPE,
stdout=subprocess.PIPE)
    output, _ = process.communicate("\n".join(map(str, path)).encode("utf-8"))
    output = output.decode("utf-8", errors="replace")
    if output:
        match  = re.search(r'BSidesTLV2025\{.+\}', output)
        if match:
            print_path(path)
            # remove whitespaces for bad flags
            print("".join(match.group(0).split()))
```

[איור 17 - פונקציית activate הכוללת הדפסת הדגל]

נריץ ולאחר המתנה של מספר שניות, נקבל את הפלט:



[איור 18 - הדפסת המסלולים שפותרים את האתגר ביחד עם הדגל שנגזר מהם]

אז קיבלנו 7 מסלולים שפותרים את האתגר כפי שהוא, ורק אחד מהם מניב דגל עם תווים הגיוניים. עבור המסלול 3 → 7 → 6 → 1 → 5 → 5 → 3, נקבל את הדגל:

BSidesTLV2025{YoU_aRe_A_Tru3_G4m3r!!!!}

ניתן גם להעתיק את מפתח ה-xor אל הסקריפט הפייתון שלנו ולבצע המרה בעצמנו במקום לקרוא ל-Level2.exe:

```
def xor_hash(digest: bytes):
    key =
b"\x6a\x2f\x26\x4c\xc4\xc0\x76\x70\x90\xd1\x87\x9b\x2c\x4b\x91\x20\x8a\x2a\x7f\
x2e\xcb\x1d\x43\x39"
    flag = ""
    for i in range(len(key)):
        flag += chr(digest[i] ^ key[i])
    print("BSidesTLV2025{" + flag + "}", sep="")
```

[איור 19 - פונקציית xor_hash המבצעת xor של המפתח עם ה-Hash]

סיכום

אהבתי את השימוש ב-State Space Search לפתרון האתגר. ולמרות שהוטרלתי קצת על ידי הטקסט שטען כי קיים מסלול קצר יותר מהפתרון של BFS, עדיין נהניתי מהתהליך. כדאי תמיד לקחת בעירבון מוגבל את מה שרשום :)

בנוסף, אמליץ על קריאת הבלוגפוסט הבא. בפוסט זה המחבר מתאר כיצד פתר אתגר Reversing של Flare On 12 בעזרת State Space BFS ובעזרת יכולות הסקריפט של Binary Ninja.

נספח - סקריפט Python מלא

```python
""" bfs.py - State Space Search (BFS) """
from collections import deque

ARRAY = [3, 65, 4, 93, 11, 21, 12, 2, 14, 52, 27, 9, 31, 21, 39, 28, 40, 97,
49, 33, 53, 43, 55, 36, 56, 38, 67, 76, 78, 69, 98, 27, 99, 74]

import hashlib
def sha256(path: list[int]) -> bytes:
    m = hashlib.sha256()
    m.update(bytes(path))
    return m.digest()

def check_hash(digest: bytes) -> bool:
    return (digest[25] == 0x7C) or (digest[26] == 3) or (digest[27] == 0x92)

def update_health(health: int) -> int:
    i = 0
    while i < 17:
        index = i << 1
        if health == ARRAY[index]:
            health = ARRAY[index+1]
            i = 0
        i += 1
    return health

def print_path(path: list[int]):
    print(" --> ".join(f"{i}" for i in path), ';', sep="", end="\t")

def state_space_bfs(max_path_length):
    """BFS on state space: (health, tokens_left, path)"""
    queue = deque([(1, 30, [])])
    visited = set()

    while queue:
        health, tokens_left, path = queue.popleft()
        state = (health, tokens_left)
        #if state in visited or len(path) > max_path_length:
        if len(path) > max_path_length:
            continue
        visited.add(state)

        if tokens_left == 0:
            if health == 100:
                digest = sha256(path)
                if check_hash(digest):
                    activate(path) # or xor_hash(path, digest)
            continue

        max_tokens = tokens_left if tokens_left < 10 else 10
        for tokens in range(1, max_tokens+1, 1):
            new_health = update_health(health + tokens)
            new_path = path + [tokens]
            queue.append((new_health, tokens_left - tokens, new_path))
```

```python
import subprocess, re
def activate(path: list[int]) -> str:
    process = subprocess.Popen("Level2.exe", stdin=subprocess.PIPE,
stdout=subprocess.PIPE)
    output, _ = process.communicate("\n".join(map(str, path)).encode("utf-8"))
    output = output.decode("utf-8", errors="replace")
    if output:
        match  = re.search(r'BSidesTLV2025\{.+\}', output)
        if match:
            print_path(path)
            print("".join(match.group(0).split()))  # remove whitespaces for
bad flags

def xor_hash(path: int, digest: bytes):
    key =
b"\x6a\x2f\x26\x4c\xc4\xc0\x76\x70\x90\xd1\x87\x9b\x2c\x4b\x91\x20\x8a\x2a\x7f\
x2e\xcb\x1d\x43\x39"
    flag = ""
    for i in range(len(key)):
        flag += chr(digest[i] ^ key[i])
    print_path(path)
    flag = "".join(flag.split()) # remove whitespaces for bad flags
    print("BSidesTLV2025{" + flag + "}", sep="")

if __name__ == "__main__":
    state_space_bfs(7)
```

[איור 20 - סקריפט Python מלא]

רפרנסים

- https://jhalon.github.io/flare-on-12-ntfsm
- https://codeforces.com/blog/entry/144451
- https://en.wikipedia.org/wiki/Breadth-first_search
- https://en.wikipedia.org/wiki/Finite-state_machine
- https://en.wikipedia.org/wiki/State-space_search