ארגון ותכנות המחשב

תרגיל בית 2 (יבש)

מיכאל בייגל	213764251
אביב אביטל	207260258

חלק ראשון – פונקציות ומחסנית (60 נקודות)

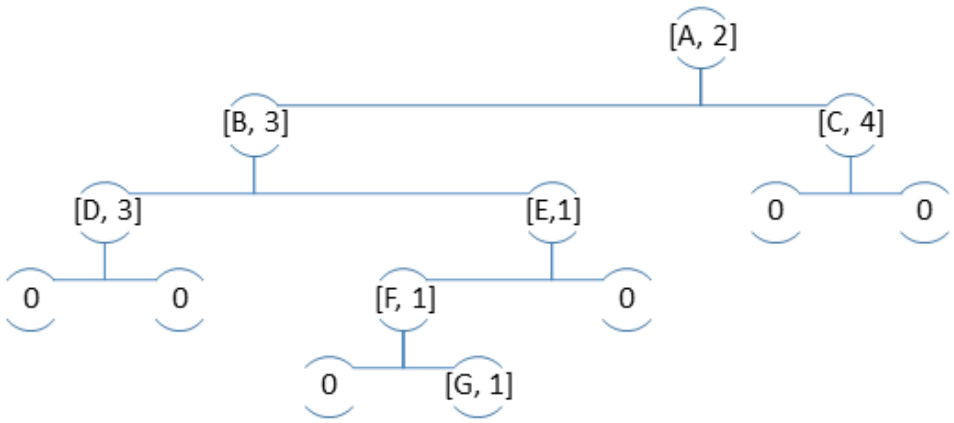
ניק רוצה לקחת ספינה מויקטוריה לאורביס. אך אבוי, קרתה תקלה במערכות הכרטיסים של הספינה. ניק מפחד שיפספס את הספינה, ולכן משתמש בכישורי האסמבלי שרכש בקורס את"מ כדי למצוא את התקלה!

ראשית, נתבונן במבנה הנתונים הבא. כמות הזיכרון שהוקצתה ל-buf אינה ידועה לניק, אך היא לפחות בית אחד.
buf הינה תווית המכילה מספר (integer number)

1	.section .data		13		.quad 0
2	buf:	._____ 0	14		.quad 0
3	A:	.int 2	15	E:	.int 1
4		.quad B	16		.quad F
5		.quad C	17		.quad 0
6	B:	.int 3	18	F:	.int 1
7		.quad D	19		.quad 0
8		.quad E	20		.quad G
9	C:	.int 4	21	G:	.int 1
10		.quad 0	22		.quad 0
11		.quad 0	23		.quad 0
12	D:	.int 3			

סעיף 1 (5 נקודות)

עזרו לניק לצייר את הגרף המתקבל מפירוש מקטע הנתונים (מומלץ להסתכל בתרגול 3, תרגיל 1, ולהיזכר כיצד מפרשים את הזכרון לגרף). בכל צומת בגרף ציינו את התווית המתאימה לה ליד הערך המספרי המתאים לו. לדוגמה: [A , 5] מציין ש-A זה שם התווית ו-5 זהו הערך אשר נמצא בתווית זו.



המשך השאלה בעמוד הבא

כחלק מהמאבק במערכת הכרטיסים, ניק הצליח לקבל את קוד האסמבלי של המכונה, מהקובץ `tickets.asm`, והוא מצורף לכם כאן. קוד זה משתמש במבנה הנתונים המתואר בתחילת השאלה.

tickets.asm

```
1  .global _start, sod
2  .section .text
3
4  _start:
5    leaq A, %rdi
6    movl $1, %esi
7    call sod
8    movq %rax, buf
9    leaq buf, %rsi
10   movl $1, %eax
11   movl $1, %edi
12   movl $1, %edx
13   syscall
14   movq $60, %rax
15   movq $0, %rdi
16   syscall
17 sod:
18   pushq %rbp
19   movq %rsp, %rbp
20   subq $32, %rsp
21   movq %rdi, -24(%rbp)
22   movq %esi, -28(%rbp)
23   cmpq $0, -24(%rbp)
24   jne .L2
25   movl $1, %eax
26   jmp .L3
27 .L2:
28   addl $1, -28(%rbp)
29   movq -24(%rbp), %rax
30   movq 4(%rax), %rax
31   movl -28(%rbp), %edx
32   movl %edx, %esi
33   movq %rax, %rdi
34   call sod
35   movl %eax, -4(%rbp)
36   movq -24(%rbp), %rax
37   movq 12(%rax), %rax
38   movl -28(%rbp), %edx
39   movl %edx, %esi
40   movq %rax, %rdi
41   call sod
42   movl %eax, -8(%rbp)
43   movl -4(%rbp), %eax
44   imull -8(%rbp), %eax
45   movl -28(%rbp), %edx
46    imull %edx, %eax
47   movl %eax, -12(%rbp)
48   movq -24(%rbp), %rax
49   movl (%rax), %eax
50   imull -12(%rbp), %eax
51   movl %eax, %edx
52   movq -24(%rbp), %rax
53   movl %edx, (%rax)
54   movq -24(%rbp), %rax
55   movl (%rax), %eax
56 .L3:
57   leave
58    ret
```

סעיף 2 (15 נקודות)

עזרו לניק לתרגם את הקוד של tickets.asm משפת אסמבלי לשפת C, על ידי כך שתשלימו את המקומות החסרים בקטע הקוד הבא. מותר להשלים יותר ממילה אחת בכל קו, אך לא יותר מפקודה אחת בכל קו!

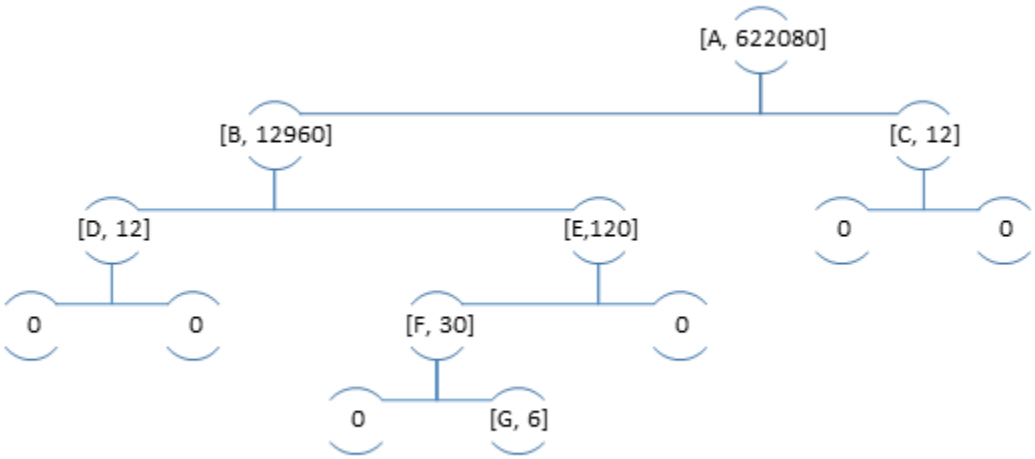
```
struct TreeNode {

       int value;

       TreeNode* left;

       TreeNode* right;

__attribute__((packed)); //הסבר בתחתית העמוד


int sod ( TreeNode* root, int mul) {


       if (root == NULL) {


                    return 1; {


      mul = mul + 1;


       int leftProduct = sod(root->left, mul) ;


       int rightProduct sod(root->right, mul);


       int tmp = leftProduct * rightProduct * mul;


       root->value *= tmp;


       return root->value;

}
```

__attribute__((packed)) משמש כדי לומר לקומפיילר לסדר בזכרון את איברי ה-struct ללא הוספת ביטים שישמשו כריפודים בזכרון, מאחר ובאופן דיפולטיבי הקומפיילר מכניס ריפודים בין איברי struct משיקולי alignment על מנת לייעל גישה לזכרון וביצועים. (כיף לדעת לידע כללי😊)

המשך השאלה בעמוד הבא

סעיף 3 (5 נקודות)

עזרו לניק לצייר את הגרף מחדש לאחר שהרצנו עליו את הקוד של `tickets.asm`:



סעיף 4 (5 נקודות)

נתון שבתחילת התוכנית ערך rsp הינו X, כאשר X הוא מספר **הקסדצימלי**. רשמו את טווח הערכים של rsp בעזרת X לאורך ריצת התוכנית על הקלט שניתן בתחילת השאלה (את טווח הערכים יש גם לרשום בבסיס הקסדצימלי).

$$X - 0x120 \leq rsp \leq X$$

סעיף 5 (5 נקודות)

הציעו פקודה שתחליף את 2 הפקודות בשורות 54-55, ואשר חוסכת בפניות לזיכרון מבלי לשנות את התוכנית.

movl %edx, %eax

סעיף 6 (5 נקודות)

מה מחזירה הפונקציה sod? יש להסביר באופן כללי ולציין את ערך החזרה הספציפי בריצה עבור מבנה הנתונים הנתון בתחילת השאלה.

ערך החזרה החזרה הספציפי במבנה הנתונים הנתון הוא 622080. באופן כללי, הפונקציה sod עוברת איבר איבר בעץ, עבור עלה - מחזירה את כפל (העומק שלו + 1) בערך שלו. עבור כל צומת שאינה עלה,

היא מחזירה את כפל (העומק שלו + 1), הערך שלו, בתוצאת הפעולה על הילד השמאלי ותוצאת הפעולה על הילד הימני. עבור ילד שמאלי / ימני ריק, מחזירה 1.

המשך השאלה בעמוד הבא

סעיף 7 (5 נקודות)

מה יהיה הפלט (לערוץ הפלט הסטנדרטי) של התוכנית בקוד `tickets.asm`? יש לכתוב תשובה **בהקסדצימלי**. (ערוץ הפלט הסטנדרטי אינו מה שנראה בטרמינל מאחר ואינו תו 7bit-ASCII סטנדרטי)

0x00

סעיף 8 (5 נקודות)

ניק שם לב שהפלט בסעיף 7 אינו תואם את התשובה שקיבל בסעיף 6. הסבירו מה הבאג בתוכנית שגורם לכך.

הפלט בסעיף 7 אינו תואם את התשובה בסעיף בגלל הפרמטר ברגיסטר rdx (או בקוד הנתון - edx).
פרמטר זה קובע את מספר הבתים שיש לכתוב לערוץ ההוצאה בפקודת write. בגלל שהמספר הנתון שקיבלנו גדול מבית אחד, המספר יכתב בחלקו בלבד לערוץ ההוצאה ולא יכתב בשלמותו, ולכן ההבדל בין סעיף 7 לסעיף 6.

סעיף 9 (5 נקודות)

הציעו לניק תיקון לקוד, כך שהפלט לערוץ הסטנדרטי יהיה תואם לערך החזרה של הפונקציה sod. יש לציין במפורש אילו שורות שיניתם ומה השינוי.

ניתן לשנות את שורה 12 בקוד הנתון - במקום לקבוע את הפרמטר כבית אחד (movl %1, %edx), ניתן לכתוב את הפרמטר כ-4 בתים (movl $4, %edx).

סעיף 10 (5 נקודות)

לאחר שניק תיקן את קוד המכונה, הגיע בלרוג רשע ושינה בקובץ `tickets.asm`,(ללא שורות 11 ו-15) כל הופעה של הרגיסטר rdi ברגיסטר rdx. כך, לדוגמה, בשורה 33 הקוד יהיה כעת `movq %rax, %rdx`. מה תהיה הבעיה בשינוי כזה? הניחו כי הפונקציה sod צריכה בעתיד לשמש עוד מכונות של חברות אחרות.

הבעיה בשינוי כזה היא שהשינוי ישבור את כללי הקונבנציה - אמנם בקוד הנתון הפונקציה תעבוד, אך באופן מכליל קבלת פרמטרים עבור פונקציה משתמש בrdi כפרמטר ראשון. כאשר שינינו רגיסטר זה להיות rdx, על כל קוד שישתמש בפונקציה הזאת בעתיד להכיר את השינוי הספציפי הזה, ואין ביכולתו להשתמש בה באופן נאיבי כפי שיכול היה להשתמש לפי הכללים הידועים.

חלק שני – קריאות מערכת (40 נקודות)

ניק הגיעה לאורביס בהצלחה, במשימה ללמוד את הסודות של קריאות מערכת על מנת שיוכל לבנות מערכת יעילה שתוכל להביס את זאקום. עזרו לניק להבין לעומק על ידי מענה השאלות הבאות:

סעיף 1 (10 נקודות)

עבור כל אחת מהטענות הבאות, סמנו נכון/לא נכון. אין צורך בהסברים.

1. בעת קריאת מערכת הפעלה (syscall), החלפת מחסנית המשתמש למחסנית גרעין מתבצעת על ידי קוד ה-handler[1]. נכון / לא נכון
2. לאחר ביצוע קריאת מערכת הפעלה (syscall), ערך החזרה מועבר על המחסנית. נכון / לא נכון
3. הפקודה sysret מעדכנת רק את רגיסטר הדגלים. נכון / לא נכון
4. בביצוע קריאת מערכת הפעלה (syscall) בארכיטקטורת 64bit, כתובת החזרה מה-handler לקוד המשתמש עוברת דרך rcx . נכון / לא נכון
5. רמת ההרשאה הנוכחית של תהליך נשמרת ב-2 ביטים ברגיסטר CS. נכון / לא נכון

המשך השאלה בעמוד הבא

[1] כפי שנלמד בהרצאות, ה-handler בלינוקס (ובקורס) הוא entry_SYSCALL()

ניק רוצה להבין לעומק מהי חלוקת האחריות בין המשתמש ומערכת ההפעלה כאשר מתבצעת קריאת מערכת.

סעיף 2 (5 נקודות)

מלאו את הטבלה הבאה, כך שתהיה ברורה חלוקת האחריות בין המשתמש ומערכת ההפעלה והמעבד בהקשרים של גיבוי ערכים רלוונטיים לתוכנית. יש להתייחס רק למה שנלמד בקורס.

סוג הקוד	ערכים בתוכנית שבאחריותו לגבות
משתמש	rcx , r11
מערכת ההפעלה	syscall handler saves all the registers on the kernel stack.
המעבד	rip, rflags, cs

על מנת להילחם בזאקום, ניק החליט להיעזר בבישוף החכם עדן שיעזור לו לכתוב קריאת מערכת להבסתו, מכיוון שזאקום היא מפלצת חזקה ודורשת שימוש בכל משאבי המערכת יחד, מה שמשתמש אינו יכול לקבל ללא קריאת מערכת.

סעיף 3 (5 נקודות)

שניהם החליטו על השם sys_attack כשם לקריאת המערכת החדשה, והם רוצים להוסיפה למערכת ההפעלה. כיצד יוכלו לעשות זאת? (רק לפי מה שנלמד בקורס)

הדרך בה יוכלו להוסיף את התוכנית למערכת ההפעלה היא הוספת התוכנית sys attack לטבלה

sys_call_table והגדלת הקבוע NR_syscalls. כלומר, מציאת הערך NR_syscalls, הגדלתו ב-1, הוספה

לטבלה באינדקס של הערך שנוסף כך שיצביע למקום של התוכנית החדשה.

המשך השאלה בעמוד הבא

סעיף 4 (5 נקודות)

לזאקום 8 ידיים, שנדרש להפילן על מנת להביסו. לכן ניק רוצה להיות מסוגל לתקוף את 8 הידיים של זאקום בבת אחת, ולכן נדרש להעביר 8 פרמטרים כאשר כל פרמטר הוא מיקום (ערך מספרי שלם) של יד ספציפית לקריאת המערכת שלו. האם ניק מסוגל לעשות זאת ביצירת קריאת המערכת? **נמקו.**

בצורה הנאיבית - ניק לא יכול להעביר 8 פרמטרים לקריאת מערכת ברגיסטרים שונים, לכן עבור העברת כל פרמטר בנפרד - אין יכולת לעשות זאת. אך, כיוון שלמערכת יש גישה למרחב הזיכרון של המשתמש, ניתן להעביר כפרמטר את הכתובת לרצף או מבנה בזיכרון המכיל את הפרמטרים הנדרשים (למשל - להעביר באחד מהפרמטרים הרגילים כתובת בזיכרון ממנה כל 4 בתים הם הint אותו אנחנו צריך לקרוא, והם כתובים בזיכרון באופן רציף). למדנו בהרצאה כי משתדלים להימנע מכך, אך במידת הצורך פתרון זה יספק את הנדרש, וכך תוכל קריאת המערכת לקרוא את הפרמטרים. לסיכום - ניק מסוגל לעשות זאת

סעיף 5 (5 נקודות)

לאחר יצירת קריאת המערכת בסעיף 3, ניק החליט שאין צורך בחזרה מקריאת המערכת עם sysret, אלא שישתמש רק ב-ret מאחר שאינו משנה את רגיסטר הדגלים. עדן טוען שניק טועה. מי מהם צודק, ומדוע?

עדן צודק, כיוון שהsysret לא רק משחזר את רגיסטר הדגלים, אלא בנוסף משחזר את מצב המערכת - מחזיר את רגיסטר ההרשאות לרמה הנכונה ועובר מגישה לאיזור הזיכרון של הkernel לגישה לאיזור הזיכרון של הuser. כיוון שret לא משנה את איזור הזיכרון בו נמצאים ולא משנה את רמת ההרשאות, הפעולות אינן שקולות ויכולות לגרום לבעיות.

המשך השאלה בעמוד הבא

סעיף 6 (10 נקודות)

עדן שיתף מחוכמתו ואמר לניק שיצטרך קריאת מערכת נוספת, sys_defense אשר תגן עליו מהמתקפות אש של זאקום, אך שתקרא לאחר כל שימוש של קריאת המערכת הראשונה שיצר sys_attack. כלומר sys_defense תבוא אחרי כל קריאה של sys_attack ואף פעם לא בנפרד. בנוסף, שתיהן מגיעות תמיד באותו הסדר.
כיצד עליהם לבצע זאת בצורה היעילה ביותר? נמקו בעזרת המידע הנלמד בקורס, התייחסו לאופן שבו קריאת מערכת וכיצד ניתן לבצע זאת, ומדוע הדר שבחרתם היעילה ביותר.

נבצע את הפעולה הנ"ל בצורה היעילה ביותר באופן הבא - הקוד שלנו יהיה בנוי כך שנקבע syscall אחד עבור sys_attack, כפי שעשינו בסעיפים קודמים ובתוך sys_attack נרצה לבצע קריאה לsys_defence. ניצור את sys_defence כקריאת מערכת רגילה המצפה לקבל את הפרמטרים לפי הקונבנציה של syscall, אך לא נוסיף אותה לטבלה של קריאות המערכת ולא נפנה אליה ישירות. עבור sys_attack, נוסיף לקוד שלנו לאחר סיום הפעולות הרגילות, נעביר את הפרמטרים לsys_defence ע"פ הקונבנציה של קריאות מערכת - כלומר, נכניס את הפרמטרים לרגיסטרים לפי קונבנציה זו. בשונה מקריאת מערכת רגילה, .נעבור לקוד של sys_defence על ידי קריאת jmp - זאת כיוון שבsys_attack אנחנו כבר במרחב של מערכת ההפעלה וכן כל הרגיסטרים גובו על ידי הsyscall שאיתו קראו לsys_attack. לאחר סיום ביצוע sys_defence, הקוד יקרא לsysret כאילו היה קריאת מערכת רגילה. בגלל שבקוד המשתמש המקורי ביצענו syscall, קריאה לsysret יחזיר אותנו אל הקוד המקורי.

לסיכום, רצף הפעולה יראה באופן הבא:

$$user\ code \rightarrow_{syscall} sys\_attack \rightarrow_{jump} sys\_defence \rightarrow_{sysret} user\ code$$

בהצלחה!