הנדסה לאחור 236496 – תרגיל בית 1

מגישים: שליו ריסין, אלון פליסקוב

חלק ב – היכרות ראשונית עם קובץ C מהודר

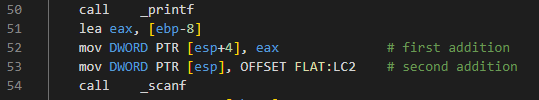
נפרט על הטעויות בקבצים partb.nopt.s ו-partb.opt.s .

בקובץ partb.nopt.s :

1. בלייבל LC2, מופיעה המחרוזת “%c\0” במקום המחרוזת “%d\0”. בגלל טעות זו, בפונקציה scanf, כאשר המשתמש מכניס מספר לקלט, אז המספר נקרא בתור מחרוזת של ספרות ascii, במקום בתור מספר (int). בנוסף, תיקרא רק ספרה אחת מהקלט במקרה הזה, כי מחרוזת הפורמט %c מתאימה לקליטת תו יחיד.
2. לפני הקריאה ל-scanf, לא נדחפים הפרמטרים למחסנית – גם הכתובת של המשתנה בו יישמר קלט המשתמש, וגם המחרוזת המתאימה ל-scanf, שבמקרה זה היא "%d".

המחרוזת שמועברת ל-scanf היא אותה המחרוזת שהועברה ל-printf, שהיא

“enter your guess: ”. מכיוון שלא מופיעים במחרוזת אחוזים כגון %d %c, לא ייקלט קלט מהמשתמש. על מנת לתקן את הטעויות, הוספנו שתי שורות לקוד, המסומנות בהערות.



1. ללייבל L3 אמורים להגיע אחרי ניחוש לא מוצלח, וממנו אמורים לקפוץ חזרה לנסיון הניחוש – L2, אך פקודת ה-jmp בסוף L3 לא קיימת, לכן כל נסיון ניחוש של המשתמש היה מצליח.

כדי לתקן את טעות זו, הוספנו פקודה “jmp L6” תחת הלייבל L3.

כעת, אחרי שהוספנו את השורה הזו, יש להוסיף גם את השורה “jmp L8” לפני הלייבל L3, כדי שבעת ניחוש מוצלח (L4), נקפוץ לסיום התכנית. ולא חזרה ל-L6.

.בקובץ partb.opt.s :

1. נשים לב, כי נקראה הפונקציה time ונקראה הפונקציה rand, אך לא נקראה הפונקציה srand.

הרצה של הפונקציה rand ללא שימוש ב-srand לפניה, מובילה לכך שה-seed קבוע ושווה ל-1, כלומר שבכל הרצה של התכנית, המספר ה"רנדומלי" יהיה זהה. תיקנו זאת על ידי הוספת שתי שורות:

A picture containing text, screenshot, font, number

Description automatically generated

כעת תיקרא srand עם הפלט של time כפרמטר, מה שיוביל לריצה המצופה של התכנית.

1. בשורה 42 (לאחר הוספת שתי השורות לעיל), מתבצעת קריאה ל-printf עם המחרוזת

“Guess a number between 1 and 100\0”, אחרי שכבר הדפסנו את המחרוזת הזו עם puts.

החלפנו את הלייבל של המחרוזת בשורה 41 ללייבל המתאים כך שתודפס המחרוזת המתאימה:

“Enter your guess:\0”.



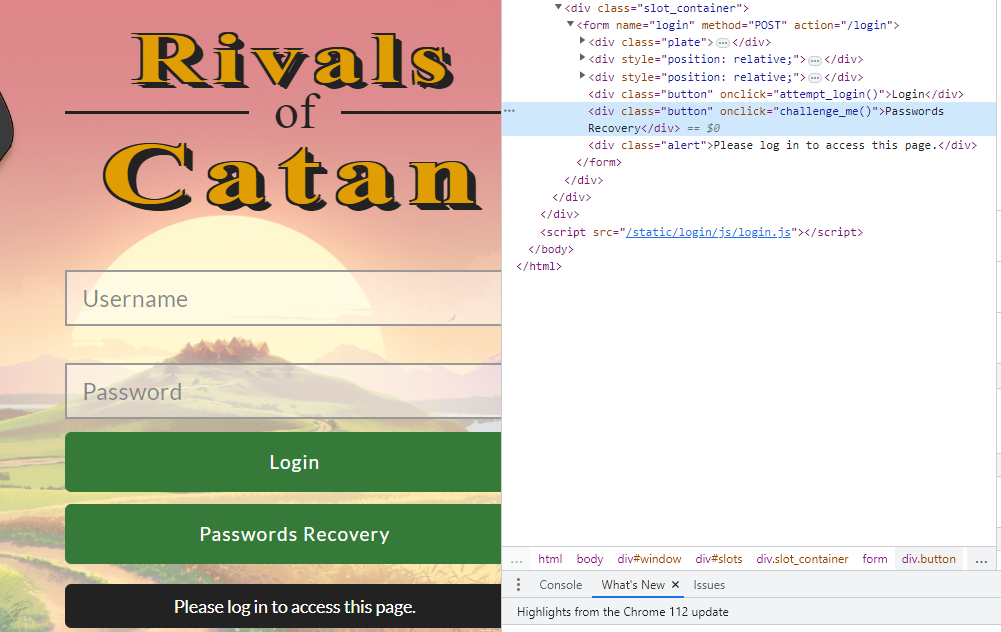
1. בשורה 47 (לאחר הוספת שתי השורות), רשומה הפקודה “jge L2”. הפקודה תגרום לכך שלא אפשר יהיה להגיע ל-L3/L5, כלומר לא ניתן לסיים את ריצת התכנית. בנוסף, זה לא מתאים ללוגיקה של התכנית. לכן, החלפנו את הפקודה ב-“jge L3” – ששם בודקים האם הניחוש נכון ואפשר לסיים את התכנית, או שהמספר קטן מדי, ויש לנסות שנית.

חלק ג – היריבים של קטאן

ראשית, נכנסנו לאתר, פתחנו את קוד ה-HTML של האתר, וזיהינו כפתור חבוי –



לאחר מחיקת המילה "disabled", הכפתור נגלה ולחיצה עליו הובילה לאתר ה-passwords recovery.



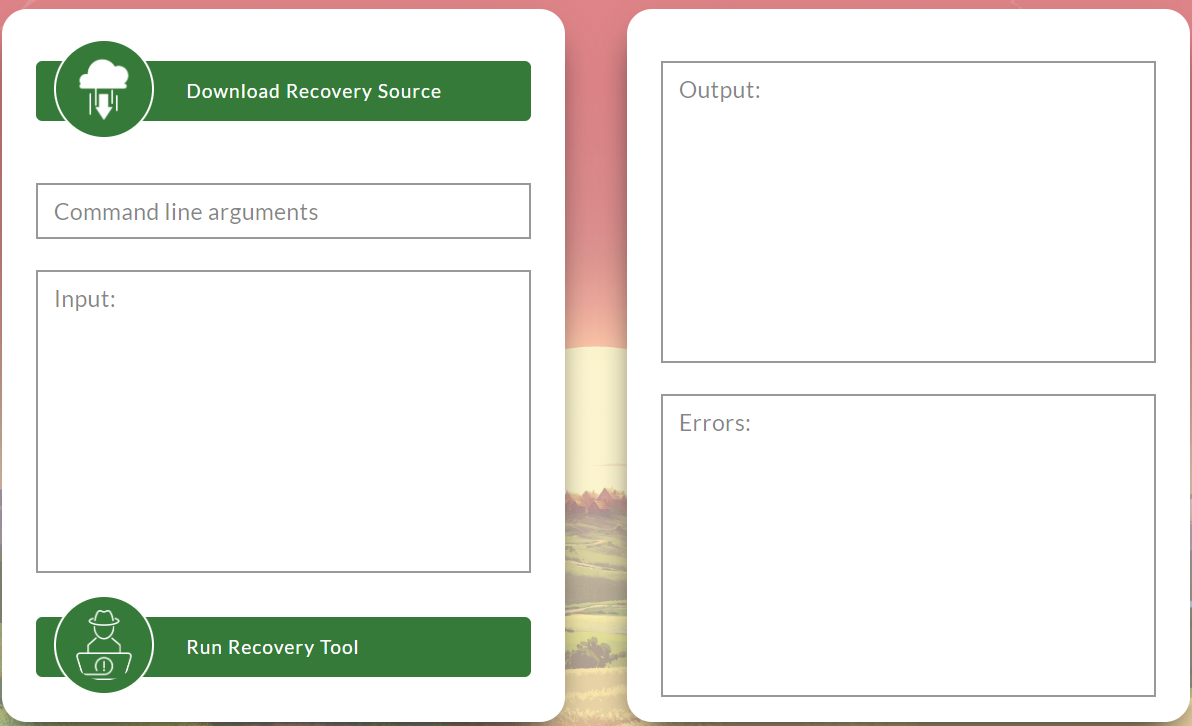
בדף הבא –



נתבקשנו לכתוב תכנית מסוימת ולכווצה ל-2KB ומטה:

כתבנו תכנית פשוטה ב-C שעושה את הנדרש. העברנו אותה ב-compiler explorer באינטרנט, כדי לקבל קוד אסמבלי. לאחר מכן, שינינו את כל הקריאות ל-printf \ scanf, וטענו אותן דינמית כפי שהראו בהדגמה בתרגול 2 ( LoadLibraryA וכן הלאה ). בקוד האסמבלי, המחרוזות היו נתונות תחת לייבלים, ושינינו זאת לדחיפה ידנית על המחסנית תוך כדי ריצה – כפי שראינו בהדגמה.

קמפלנו את קובץ האסמבלי לקובץ בינארי וחיברנו אותו למימוש של FindFunction, ואחר כך ל-PE header מתאים, תוך התאמת גודל ה-text section. העלינו את קובץ ה-exe לאתר, ועברנו לשלב הבא:



הורדנו את ה-recovery source, והתחלנו לקרוא אותו.

level 1:

תחילה, הסתכלנו על “\_main”, ושמנו לב שלפני ההדפסה של “Level 1 passed!”, משווים את הארגומנט הראשון שהפונקציה מקבלת, עם 1. גילינו שזו השוואה שבודקת האם argc גדול מ-1, ואם כן אז עברנו את השלב הראשון. מכיוון שתמיד , כי תמיד שם התכנית הוא הארגומנט הראשון, אז השלב הראשון יעבור אם הכנסנו לפחות ארגומנט אחד נוסף.

level 2:

לפני הקריאה ל-level1, מתבצעת קריאה ל-atoi, על האיבר השני במערך argv – שהוא בעצם ה-command line argument הראשון. את ערך החזרה (integer) מעבירים כארגומנט ל-level2.

התכנית קוראת ל-scanf על מנת לקבל start offset ו-end offset, שאותם מוסיפים לכתובת של תחילת מערך בזכרון, ולבסוף מבוצע עליהן alignment לגודל של 4 בתים.

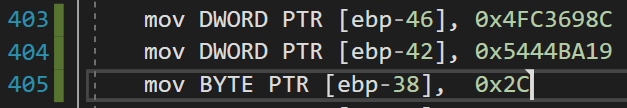
לאחר מכן, נקראת הפונקציה printArray במידה והקלט שהכנסנו תקין – שהמספר הראשון הוא אי-שלילי. הפונקציה הזו מדפיסה למסך את תוכן הזכרון בין שתי הכתובות הללו. השתמשנו בפלט הפונקציה כדי לראות מה בדיוק קיים בזכרון, והשתמשנו במידע הזה כדי לעבור את רמה 2.

ראינו כי על מנת שיודפס “Level 2 passed!”, עלינו לעבור בהצלחה סדרה של 9 השוואות בין בתים שונים בזכרון. כברירת מחדל, ההשוואות לא עוברות – הבתים אינם זהים.

**סדרת הבתים הראשונה:** ראינו כי עם seed מקובע (0), אנחנו מקבלים באופן עקבי את 12 הבתים (הראשונים) הבאים, על ידי קריאה ל-printArray בריצת התכנית:

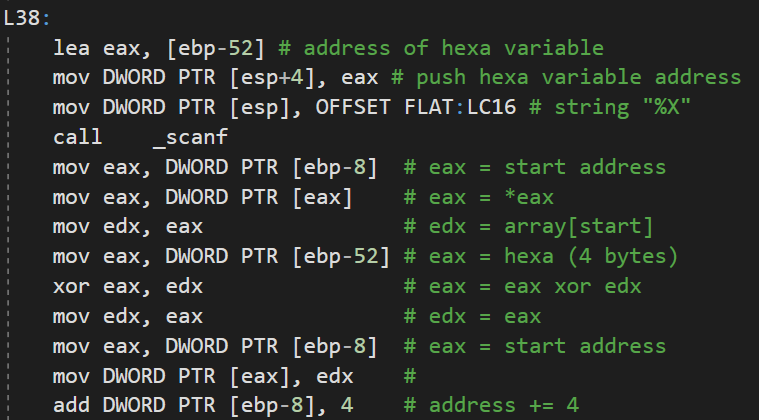


**סדרת הבתים השנייה:**

המספרים שנקבעו מראש בתכנית, והם: 

ראשית, שמים לב כי הסדרה הראשונה באורך 12 והשנייה באורך 9, לכן 3 בתים מהסדרה הראשונה הם מיותרים, ולא נערכת איתם השוואה. אלה הם שלושה הבתים הראשונים ברצף – התייחסנו אליהם כ-don’t-care.

נתבונן כעת בחלק הזה של התכנית, שרץ במסגרת level2:



החלק הזה מתבצע בתכנית מספר פעמים כמספר הdouble words שמודפסות למסך בprintArray. הפונקציה scanf נקראת כדי לקרוא קלט הקסה דצימלי. הקלט (4 בתים) עובר פעולת xor יחד עם 4 בתים מתאימים במערך, והתוצאה דורסת את תוכן המערך.

**למשל:**

הקלט הראשון שהכנסנו הוא 0x88000000.

מתבצע : 0x88000000 XOR 042C5444. התוצאה של ה-xor היא 8C2C5444. כעת, דורסים את הזכרון שבו היה שמור 042C5444, עם התוכן החדש – 8C2C5444.

ידענו כי פעולת ה-xor מקיימת שאם A XOR B = C, אז גם C XOR B = A.

אז הסקנו כי אם נבצע XOR בין סדרת המספרים הראשונה לסדרת המספרים השנייה, תוך כדי התעלמות משלושת הבתים הראשונים בסדרה הראשונה – אז נקבל סדרת קלט מתאימה.

ואז כשנכניס את הקלט הזה לתכנית בחלק של L38, תוכן המערך בזכרון(סדרת הבתים הראשונה) יידרס עם סדרת הבתים השנייה – מה שיגרום לשתי סדרות הבתים להיות זהות, ובסוף לסיום השלב.

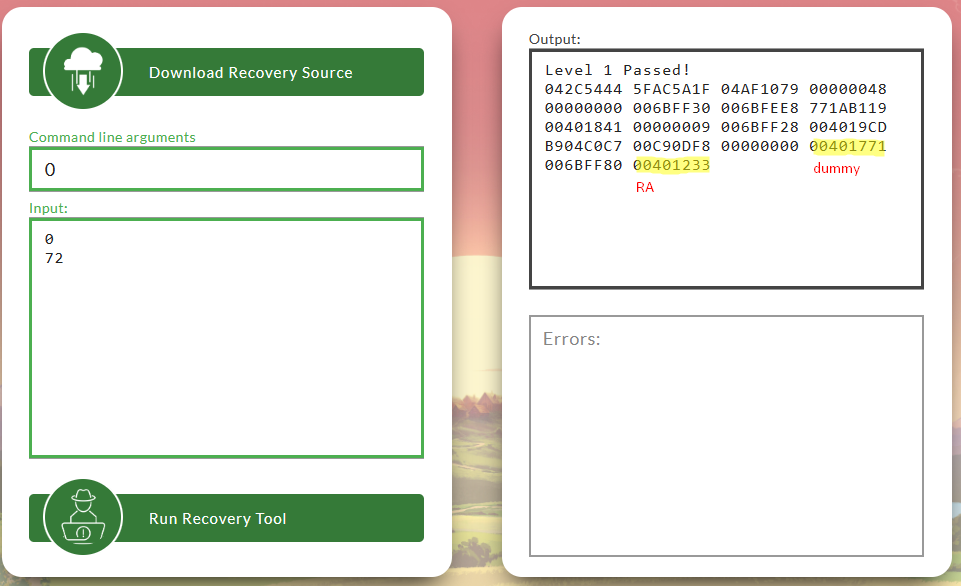
הוכחה לסיום השלב השני:

A picture containing text, screenshot

Description automatically generated

Level 3:

לאחר התבוננות נוספת בקוד ראינו שצריך לקרוא לפונקציה \_dummy\_\_ על מנת לעבור את שלב 3. לצערנו הרב (☹) הפונקציה אינה נקראת באף שורה בקוד. לכן הרעיון היה לשנות את כתובת החזרה מmain כדי שתהיה הכתובת של dummy (חשבנו גם על לשנות את כתובת החזרה של level 2 אבל כבר ראינו שנצטרך לשלב הבא את המשך main ולכן הלכנו עם הגישה הראשונה) בשביל לכתוב לstack השתמשנו בחלק משלב 2 אשר כותב על המחסנית. כעת היינו צריכים לגלות מה הכתובת של dummy ומה הכתובת חזרה מmain (נזכור שהכתיבה בשלב 2 היא xor ולא פשוט החלפת התוכן לכן צריך לדעת את שניהם). לפי הקוד, בתחילת main הכתובת נשמרת על המחסנית בדיוק 8 בתים מתחת לreturn address מmain (ebp+4 אל מול ebp-4). בנוסף הframe של main נמצא בדיוק מעל הframe של level ולאחר חישוב מהיר ראינו שאם בשלב 2 ניתן את המספרים 0 ו-72 print array תכתוב מספיק בתים מהמחסנית ונוכל לגלות את כתובת החזרה והכתובת של dummy:



00401771 xor 00401233 = 542. בנוסף, ישנה שורה בין level2 לחזרה מmain שקוראת מהcommand line argument השני ולכן הוספנו ארגומנט נוסף אחרת מקבלים segfault.

לכן עבור הcommand line argument: 0 0 והקלט:

0

72

88000000

46e39976

28fb54c3

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

00000000

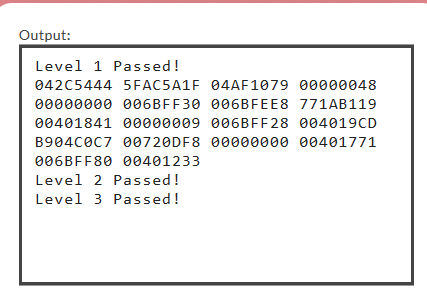
00000000

00000000

00000000

00000542

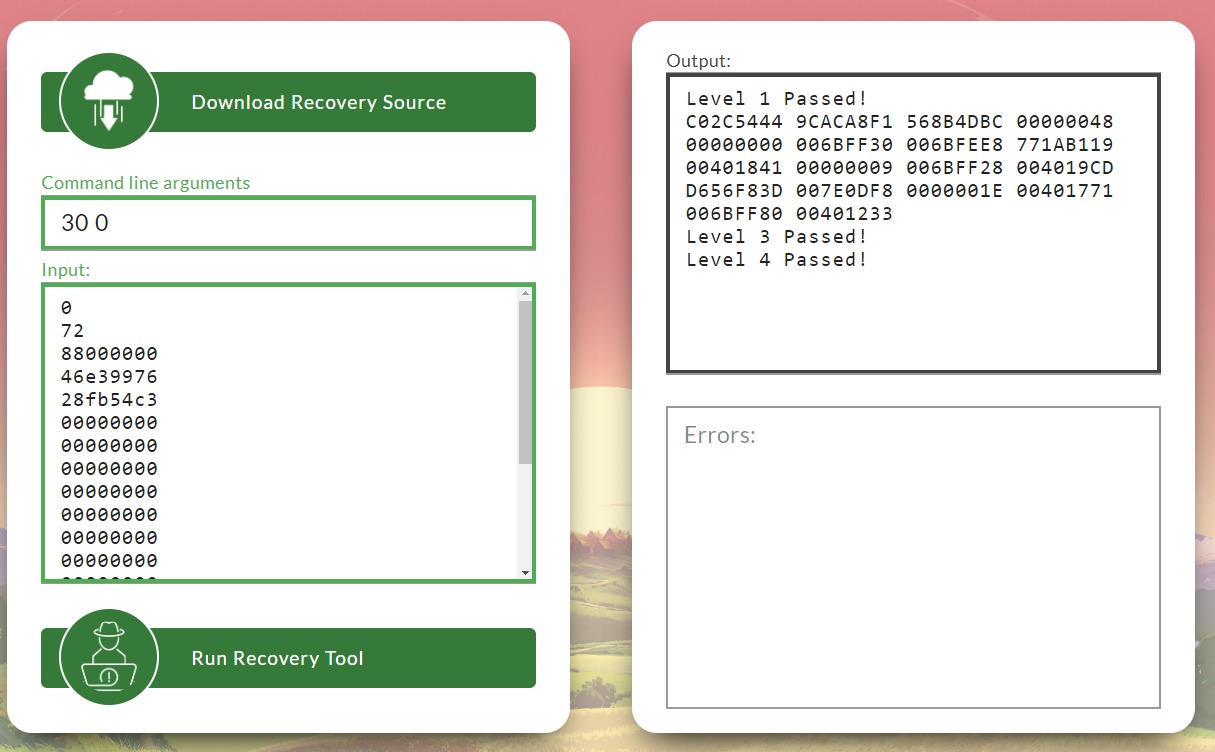
קיבלנו:

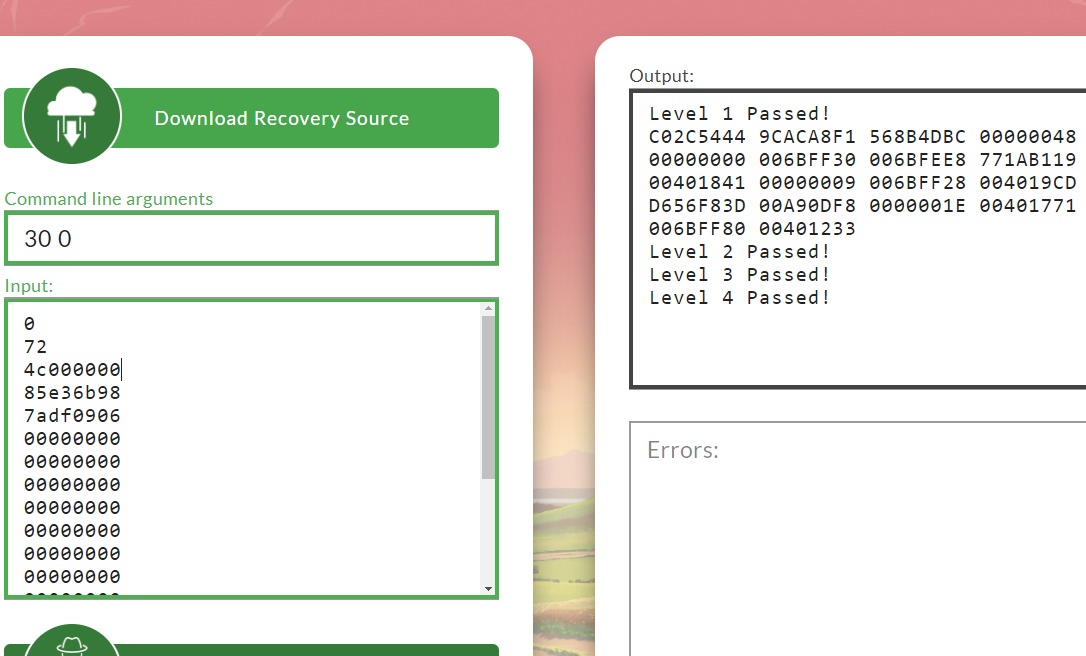


Level 4:

בתחילת הקוד קוראים לפונקציה signal עם הפונקציה handler ומספר הסיגנל 8. הקריאה מגדירה כי כעת שגרת הטיפול לסיגנל מספר 8 (floating point exception) שמתרחש בחלוקה ב0 היא handler.

החלוקה היחידה בתכנית הינה בפונקציה dummy שבה מחלקים בdivider, אשר ערכו מתקבל על ידי שימוש בארגומנט הראשון בתור seed להגרלת מספר רנדומלי. לאחר מכן יש מניפולציות אלגבריות על התוצאה כשלקראת הסוף עושים AND עם המספר 31. כלומר, modulo 32. לכן ניחשנו מספרים עבור הארגומנט הראשון עד שהמספר שנכנס לdivider היה 0 ועברנו את שלב 4:



אך כעת, מכיוון ששינינו את הארגומנט הראשון גם הseed בשלב 2 השתנה ולכן המידע בstack array שמוגרל באקראי השתנה ואנחנו לא עוברים את הבדיקה של השלב. לכן התאמנו את הinput למידע החדש ככה שהבדיקה בשלב 2 תעבור וקיבלנו: 

**db\_access**:

ראינו כי בקוד של dummy נקראת הפונקציה db\_access. חשדנו שבעזרתה נוכל לחלץ את פרטי ההתחברות של המשתמשים.   
ראינו שלפני db\_access, מופיעה השאילתא הבאה:



ותוך כדי ריצת db\_access, מתבצע:

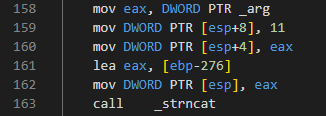
1. חיבור ל-database.
2. שאילתא ל-database.
3. סגירת החיבור ל-database.

השאילתא ששואלים הינה:

select username, password from users where username=’ || argv[2] || ‘

כאשר ראינו את השאילתא שנוצרת, עלה לנו הרעיון של SQL Injection, כלומר – אם נוכל להכניס ארגומנט מתאים ב-argv[2], כך שהתנאי של ה-where תמיד יתקיים (טאוטולוגיה) אז יחזרו כל ה-usernames וה-passwords.

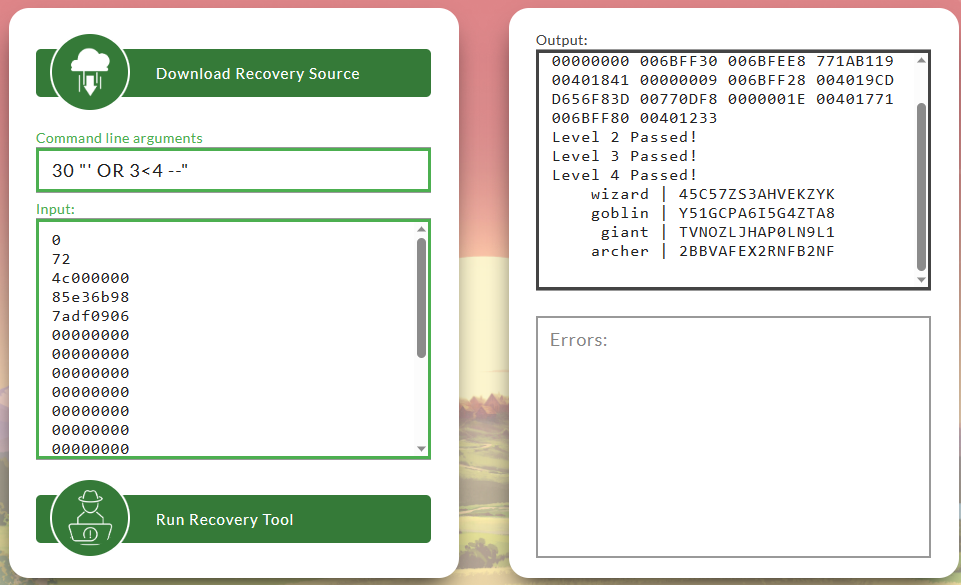
בנוסף, לפי הפונקציה, הוגבלנו ב-11 תווים (לפי שורה 159) –



בשביל לבצע SQL Injection, המחרוזת צריכה להכיל:

1. גרש (התו ') שסוגר את הגרש שכבר קיים במחרוזת – ואז מקבלים empty string.
2. תנאי נוסף ל-where כך שיהיה טאוטולוגיה.
3. הערת שורה (שני מקפים --) לביטול הגרש שמתווסף אחר כך.

הגענו למחרוזת הבאה, כפי שרואים בתמונה:



קיבלנו פרטים של ארבעה משתמשים, והצלחנו להתחבר בעזרתם למערכת.