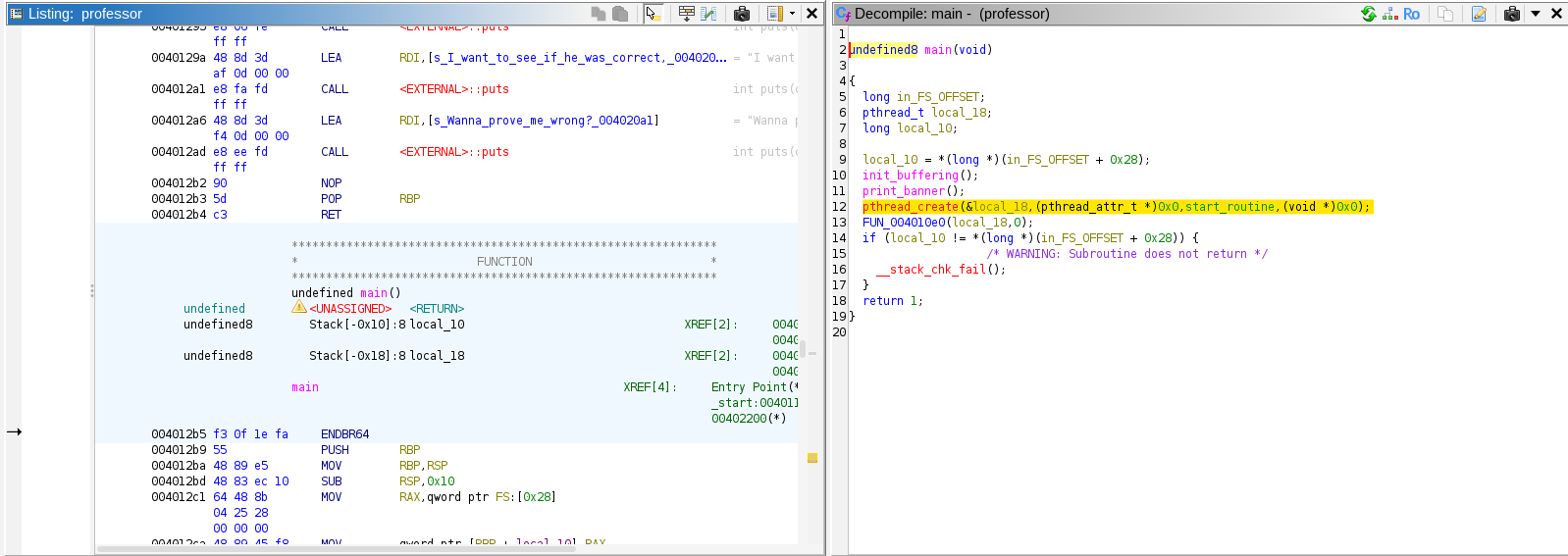
**Professor - Writeup**

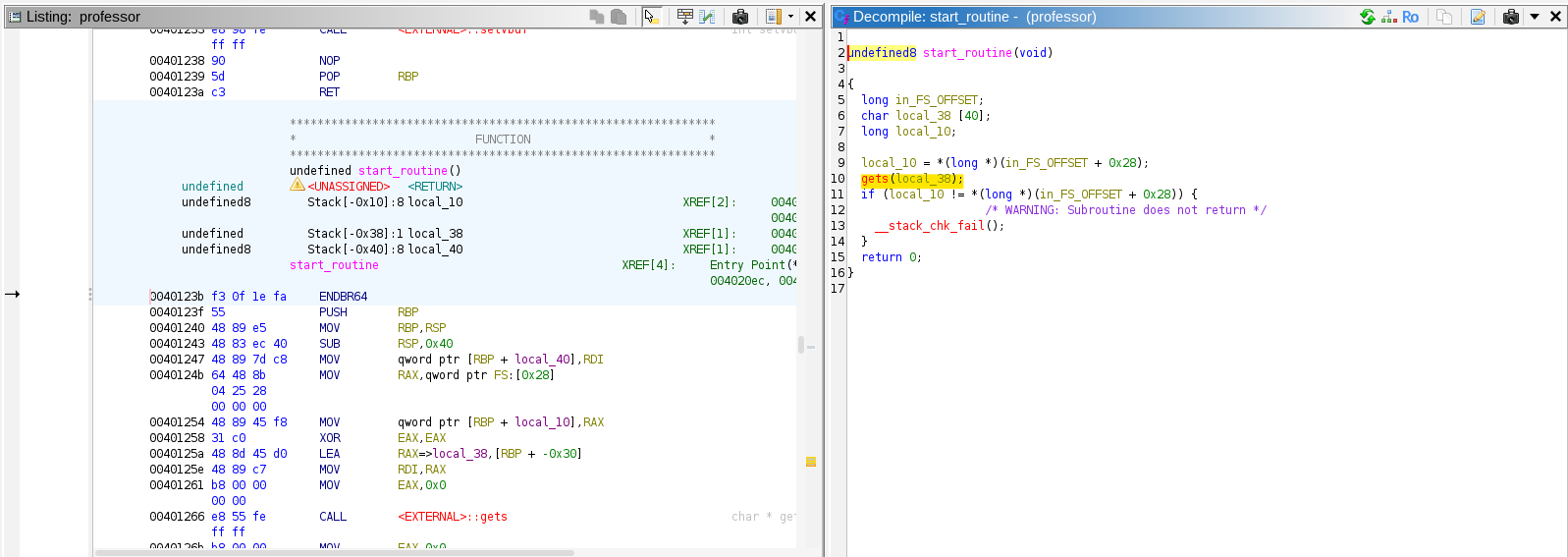
**באתגר Professor מנוצלת פונקציית gets()שנקראת מתוך. Start\_routine**

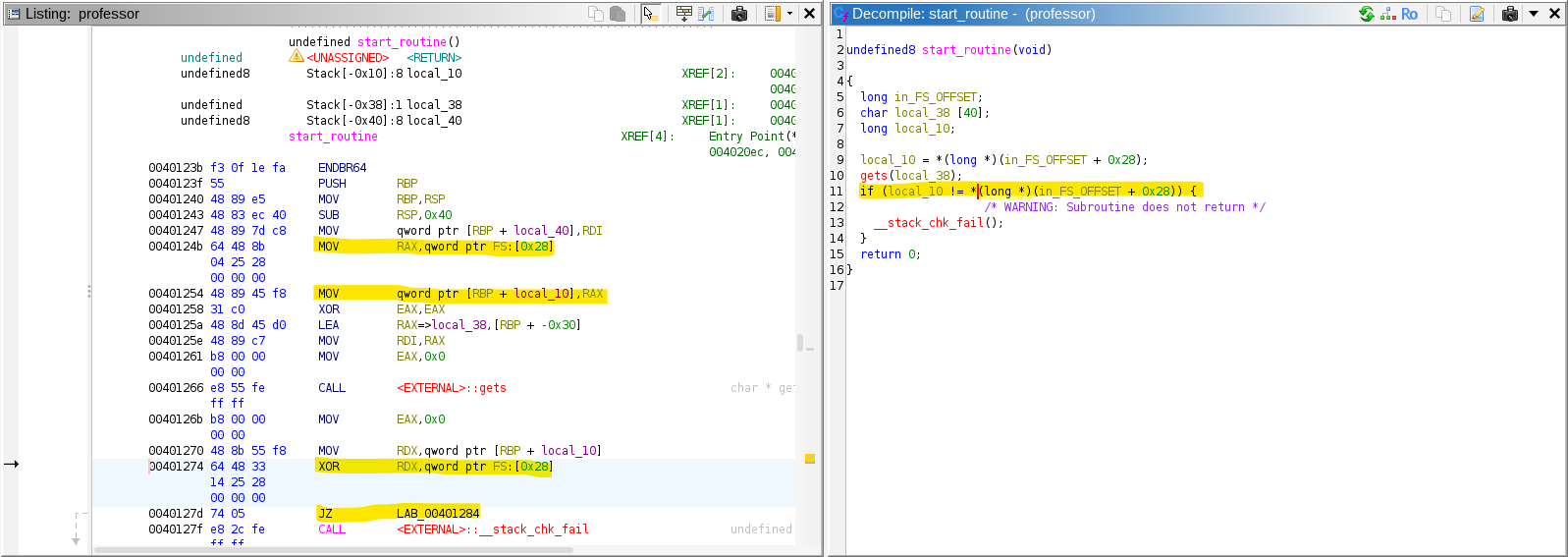
**הפונקציה gets אינה מבצעת בדיקת אורך קלט ולכן ניתן לבצע buffer overflow ולדרוס את כל הערכים שנרצה, את כתובת הRET ואת ערך הcanary ואם נדרוס מספיק נוכל גם להגיע עד לfs\_base.**

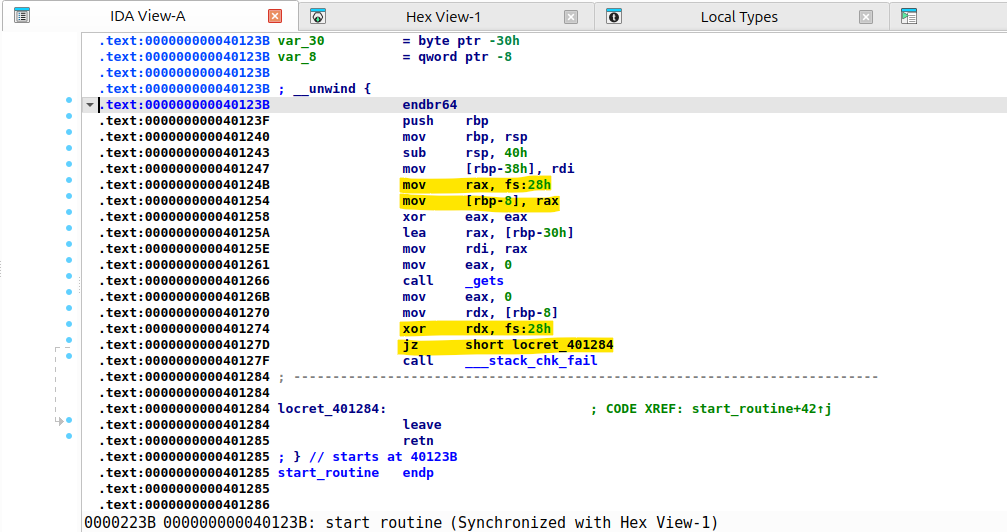
**המתקפה מנצלת את מנגנון stack canary על ידי דריסת הsaved canary במחסנית וגם את דריסת הערך הראשי בTLS canary. בסופו של דבר נגרום להפעלת shell באמצעות כתיבת ROP(return oriented programming) וכתיבה לGOT של PUTS, כל שPUTS יבציע על ONE GADGET שתפקידו להריץ פקדות SHELL.**

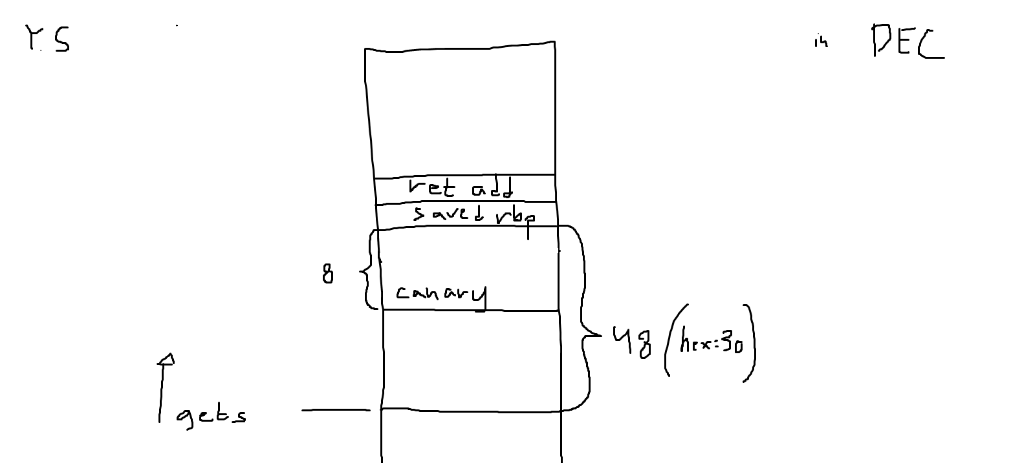
**גילוי ופירוט הפגיעות**

* **הפונקציה הפגיעה start\_routineובתוכה gets : קוראת קלט ללא הגבלת אורך פתיחת אפשרות לstack buffer overflow**

****

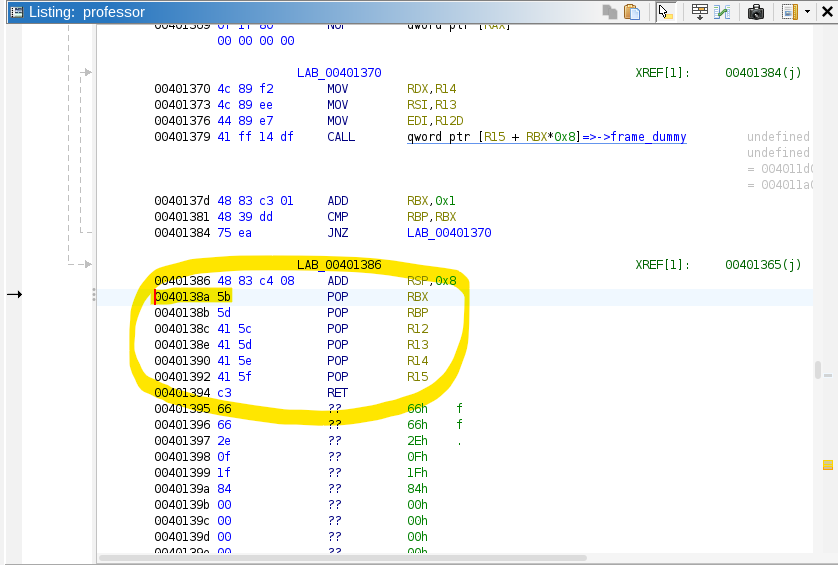
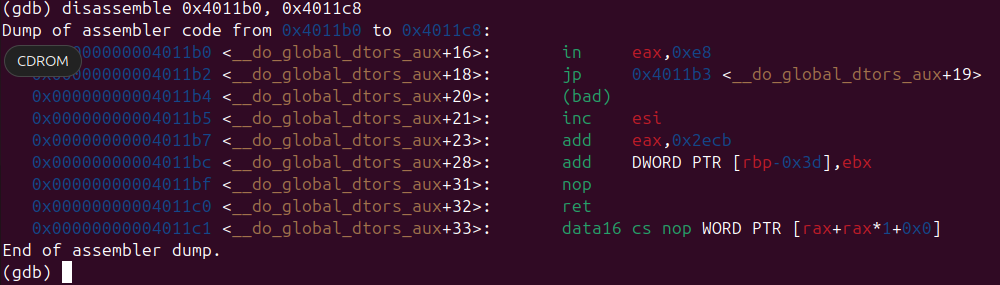
* **המערת כוללת מנגנון stack canary, עושים השוואה עם הערך בTLS כדי לבדוק האם לא בוצע שום שינוי במהלך הפונקציה לערך הנשמר במחסנית בתחילת הפונקציה.**
  1. **אנחנו נדרוס את הsaved canary ואנחנו גם נשנה את הערך שאליו מתבצעת ההשוואה – TLS canary, הערך הראשי כדי שההשוואה תעבור למרות הדריסה.**

****

****

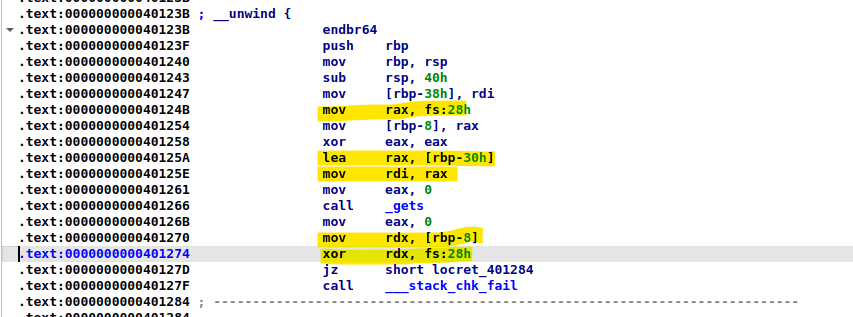
* **לאחר דריסת הRET אנחנו נדחוף מספר ערכים כדי שאחרי זה הgadget יבצע את הפעולות עם הערכים שאנחנו נרצה.**

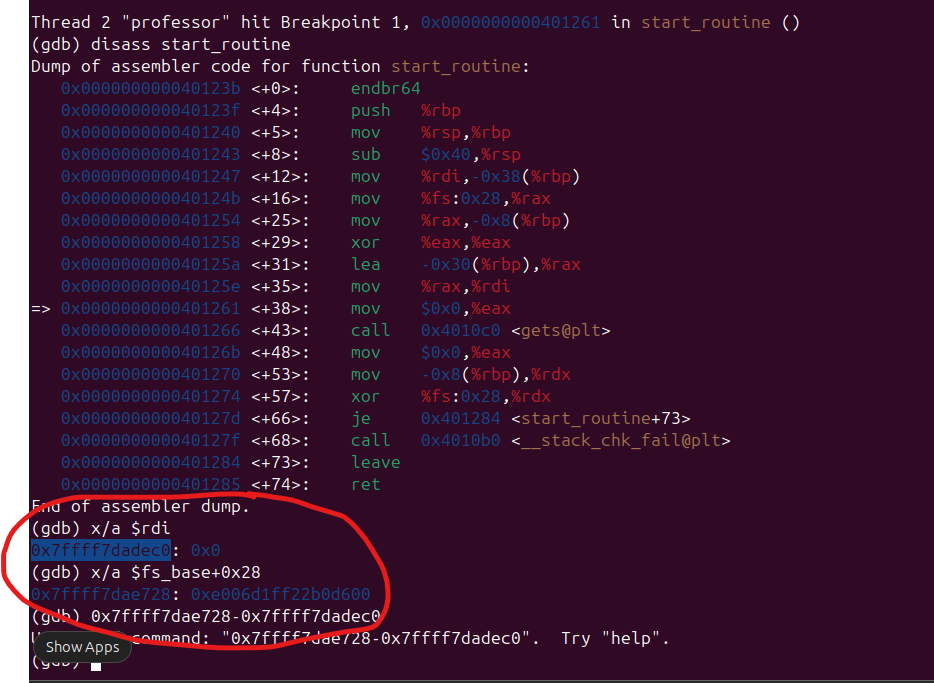
**שיטה וטכניקה (בנקודות)**

* **איתור gadgets- מה זה gadget? – כתובת בזיכרון שמכילה רצף של הוראות שימושיות שניתן לשרשר, הוראות מוכנות.** 
  1. **דוגמאות:**
* **pop\_rbx\_rbp\_r12\_r13\_r14\_r15 = p64(0x000000000040138a)**
* **add\_ref\_rbp\_rbx = p64(0x00000000004011bc)**
* **כלים כמו Ghidra and IDA FREE.**

**דילוג על הcanary:**

* **יש לשלב בדריסה גם שינוי בערך השמור במחסנית של הפונקציה start\_routine וגם במחסנית הכללית של הthread – TLS.**
  1. **בשביל לבצע את זה נצטרך לבצע חישוב של כמה בייטים נצטרך לכתוב מתחילת הכתיבה במחסנית ועד איפה שהוא שמור.** 
     + **לפי הקוד אסמלי שכבר צורף ניתן לראות הGETS מתחיל לכתוב מהrbp-0x30 והערך canary המקומי שמור בrbp-0x08 לכן נצטרך לכתוב 0x28 בייטים כדי להתחיל דריסה של הcanary הראשון. אחרי זה נבצע חישוב של כמה בייטים אנחנו צריכים מrbp-0x30 ועד הfs\_base+0x28 כי לפי הקוד זה הכתובת שממנה אנחנו לוקחים/מבצעים השוואה לcanary. את הכתובות תוכל להשיג על ידי הרצת פקודות כמו:  
       x/a $rdi לאחר שקבעת נקודת עצירה בזמן הנכון כדי לראות הכתובת שנמצאת כrbp-0x30 ולגבי הכתובת של הTLS CANARY אפשר לראות עם הפקודה: x/a $fs\_base+0x28 ואז נבצע חיסור כדי לדעת כמה בייטים לכתוב.**

****

****

* **שינוי GOT של PUTS להציע לONE GADGET**
  1. **הרעיון המרכזי הוא לשנות את כתובת הGOT של PUTS כך שתבציע לכתובת ONE GADGET בתוך LIBC ואז לקרוא לPUTS, מה שיריץ את הONE GADGET ויפתח SHELL.**
  2. **חישוב הפרש:**
     + **delta = offset\_one\_gadget - offset\_puts**
* **באמצעות gadget שמבצע כתיבה/חיבור בזיכרון מוסיפים את ה- delta לערך שבGOT של PUTS ואחר כך קוראים לפונקציה PUTS מה שיוביל לSHELL.**

**כלים ושורות פקודה שימושיות**

* **ניתוח בינארי עם Ghidra, IDA Free**
  1. **מציאת one\_gadget בlibc.**
     + **One\_gadget ./libc.so.6**
  2. **מציאת OFFSET של PUTS בLIBC** 
     + **objdump -T ./libc.so.6 | grep ' puts$'**
  3. **בדיקת PLT של PUTS:**
     + **objdump -d ./professor | grep "<puts@plt>"**

**מבנה הROP:**

* **gadget ראשון: הכנת רישומים POP... לערכים בregisters. נכניס בעצם ערכים שאנחנו רוצים לכל מיני registers שאחרי זה נצטרך כדי לבצע פעולה כלשהי.**
* **gadget שני: ביצוע הכתיבה והחיבור לכתובת הGOT של PUTS, אנחנו נוסיף לה בעצם סכום ככה שהיא תצביע על הפונקציה One gadget כשייקראו לputs.**
* **קריאה לputs בעזרת הRET, הקריאה שתופעל תקרא לONE GADGET ולא לPUTS ובכך תפעיל SHELL.**

**דגשים חשובים**

* **בדוק את מצב ההגנות שיש על הבינארי.**

**סיכום**

* **ניצלנו את חוסר ההגבלה של gets() כדי לגרום לbuffer overflow.**
* **עקפנו את מגנון הcanary.**
* **בעזרת הROP שינינו את הכתובת שנמצאת בGOT של PUTS שתבציע על one\_gadget והרצת SHELL. One\_gadget גם היא פונקציה הקיימת בLIBC.  
  ‎‬**