Зображення, що містить будівля, ескіз, малюнок, замок

Автоматично згенерований опис  
Національний технічний університет України

"Київський політехнічний інститут імені Ігоря Сікорського"

Кібербезпека та захист інформації:

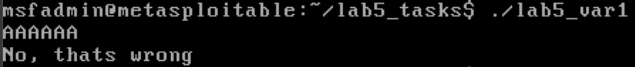
Архітектура комп’ютерних систем

Студент: Бичок Вадим Вячеславович

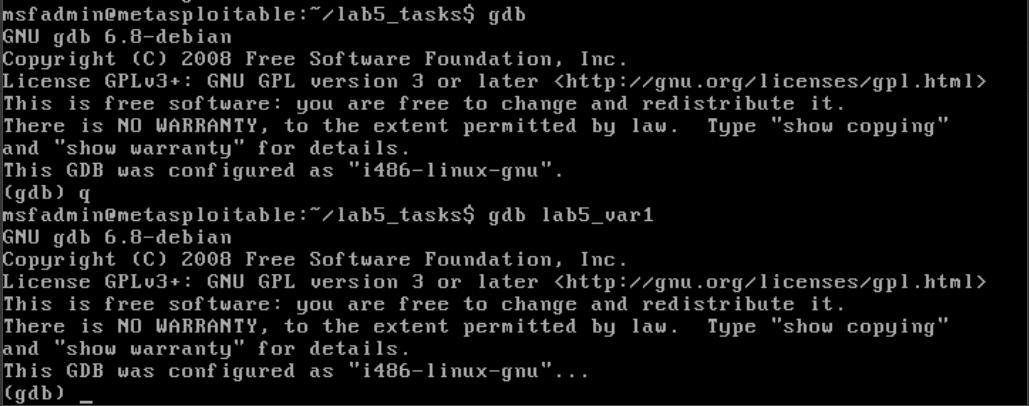
Робота: 6

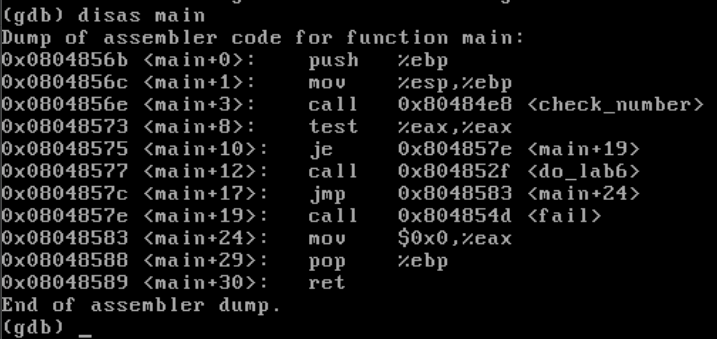
Назва: Дослідження вразливості buffer overflow в архітектурі ІА-32

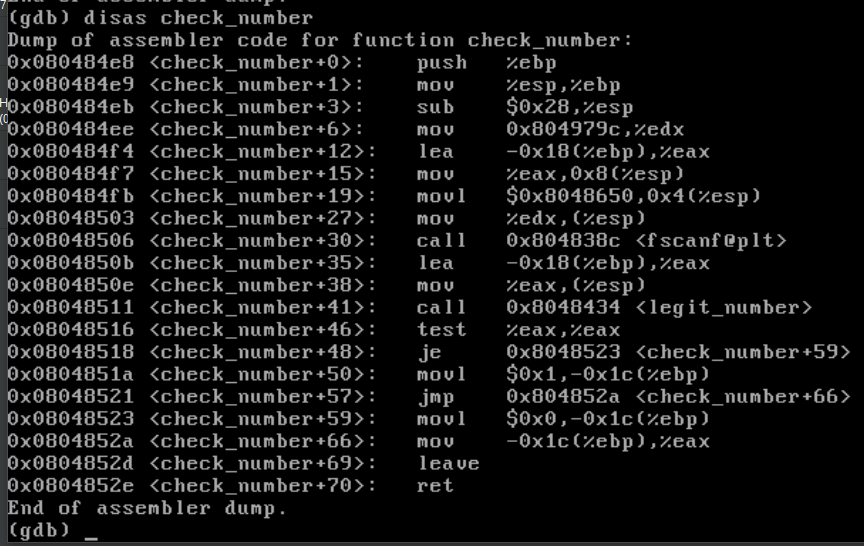
Використані програми: Oracle VirtualBox, OS msfadmin@metaspoitable

Програма полягає в навчанні використання переповнення буфера, якщо ми не навчилися, вона буде видавати “No, that’s wrong”.  


Принципи введення в програму аналогічні до прикладу в методичці.

Відкриваємо var\_lab1 через gdb  


Дизасемблюємо функцію main, бачимо виклик функціїї check\_number.  


Задизамемблюємо її. Бачимо дві функції та ret.  


Поставимо на них breakpoint, щоб потім подивитися що робиться в стеку.  
Зображення, що містить текст, Шрифт, знімок екрана

Автоматично згенерований опис

Запускаємо програму.  
Зображення, що містить текст, знімок екрана, Шрифт, чорний

Автоматично згенерований опис

Переглядаємо стек перед вводом значення, бачимо нашу адресу test eax, eax.  
Зображення, що містить текст, знімок екрана, Шрифт, число

Автоматично згенерований опис

Нажимаємо c(continue). Щось вписуємо. І переглядаємо стек. Бачимо які регістри змінилися, приймаємо рішення надалі вписувати один символ для того щоб було краще видно регістри які змінюються.  
Зображення, що містить текст, знімок екрана, Шрифт

Автоматично згенерований опис

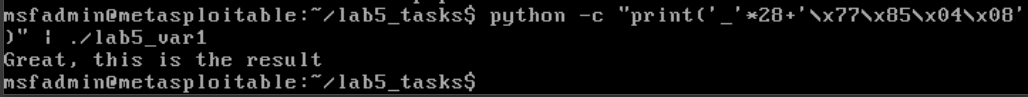
Зрозуміли принцип заповнення, тепер порахуємо кількість байтів, які нам потрібно вписати щоб дійти до адреси test eax, eax, щоб її можна було перезаписати. Це 28 знаків(байт).

Запускаємо заново програму. І переглядаємо стек.  
Зображення, що містить текст, знімок екрана, Шрифт

Автоматично згенерований опис

Перевіряємо наші здогатки. Так, все правильно.  
Зображення, що містить текст, знімок екрана, Шрифт, чорний

Автоматично згенерований опис

Тепер по аналогії з прикладу записуємо адресу do\_lab6.  
  
Все, лабораторна робота виконана.

1. Що таке Buffer Overflow?  
   Buffer Overflow (переповнення буфера) — це вразливість у програмному забезпеченні, яка виникає, коли програма записує більше даних у буфер (область пам'яті, зарезервовану для даних), ніж він здатний вмістити. Це може призвести до неконтрольованого впливу на сусідні області пам’яті, що часто використовується для порушення роботи програми або отримання контролю над нею.
2. Що таке Heap?  
   **Heap** (купа) — це область пам’яті, яка використовується для динамічного виділення пам’яті під час виконання програми. На відміну від **стеку**, де пам’ять виділяється і звільняється автоматично (під час входу і виходу з функцій), в **купі** пам’ять керується вручну програмою або автоматично системою управління пам’яттю.
3. Як працює стек?  
   **Стек** — це структура даних і частина пам’яті, яка працює за принципом **LIFO** (Last In, First Out), тобто останній елемент, який був доданий у стек, перший звідти вилучається. У контексті програмування, стек використовується для зберігання інформації, необхідної для виконання функцій (зокрема локальних змінних, адрес повернення тощо).
4. Яка структура стеку?  
   **Структура стека**

Стек у пам'яті організований як лінійна структура даних, що дотримується принципу **LIFO** (Last In, First Out). Його побудова дозволяє легко додавати та видаляти елементи тільки з одного кінця — **верхівки стека**.

1. В чому різниця між операціями push та pop?  
   Push:

Це операція додавання елемента до вершини стека.

Після виконання операції вершина стека змінюється на новий елемент.

Якщо стек має обмежену ємність, операція push може завершитися помилкою, якщо стек заповнений (так звана переповненість стека).

Pop:

Це операція видалення елемента з вершини стека.

Після видалення повертається значення елемента, який був на вершині, і вершина стека змінюється на попередній елемент.

Якщо стек порожній, операція pop може завершитися помилкою (так зване підпорожніння стека).

Аналогія: зняти верхню тарілку з купи.

1. Що таке little endian?  
   **Little endian** — це спосіб упорядкування байтів у багатобайтових даних (наприклад, числах), при якому **найменш значущий байт (LSB)** зберігається в пам'яті за найменшою адресою, а **найбільш значущий байт (MSB)** — за більшою адресою.  
     
   Уявімо 32-бітне ціле число 0x12345678 (шістнадцяткове представлення).  
   У little endian воно зберігається у пам'яті так:

Адреса : 0x00 0x01 0x02 0x03

Дані : 0x78 0x56 0x34 0x12

Тобто, порядок байтів у пам'яті виглядає обернено до логічного порядку числа.

1. Що таке heap based buffer overflow?  
   **Heap-based buffer overflow** (переповнення буфера на купі) — це тип уразливості в програмному забезпеченні, який виникає, коли програма записує більше даних у виділений в динамічній пам'яті буфер, ніж він здатен вмістити. Це може призвести до пошкодження інших областей пам'яті в купі, викликання помилок або навіть виконання зловмисного коду.
2. Що таке big endian?  
   У **big endian** найбільш значущий байт (MSB) зберігається за найменшою адресою. Наприклад, для числа 0x12345678 порядок у пам'яті буде таким:

Адреса : 0x00 0x01 0x02 0x03

Дані : 0x12 0x34 0x56 0x78

1. За що відповідає регістр esp?  
   **ESP** — це регістр у процесорах архітектури x86, який вказує на **поточну вершину стеку**.  
   Завжди містить адресу останнього (верхнього) елемента в стеку.  
   Коли виконується операція PUSH (додавання даних у стек), ESP зменшується на розмір доданих даних.  
   Коли виконується операція POP (зняття даних із стека), ESP збільшується.  
   ESP використовується для динамічного керування пам'яттю під час виклику функцій, передачі параметрів і збереження змінних.
2. За що відповідає регістр ebp?  
   **EBP** — це регістр, який зазвичай використовується як **базовий вказівник** для роботи з локальними змінними і параметрами функцій у стеку.  
   На відміну від ESP, значення EBP зазвичай не змінюється протягом виконання функції, що дозволяє легко отримати доступ до параметрів і локальних змінних у стеку.  
   EBP встановлюється на початку виконання функції для визначення початку її "фрейму" в стеку.  
   Локальні змінні зберігаються за від'ємними зміщеннями від EBP (наприклад, [EBP-4]), а параметри функції — за додатними (наприклад, [EBP+8]).  
   Перед викликом функції значення EBP попередньої функції зберігається у стеку. Це дозволяє правильно "повертатися" по стеку під час завершення вкладених викликів.  
     
   ESP постійно змінюється під час додавання або видалення даних зі стека.

EBP залишається стабільним під час виконання функції, слугуючи фіксованою точкою для доступу до параметрів і локальних змінних.

1. Що таке база?  
   База (base) — це початкова точка відліку для доступу до локальних змінних та параметрів функції в стеку.

Регістр EBP часто називають базовим вказівником, оскільки він використовується для визначення бази (основи) поточного фрейму стеку.

1. Що таке фрейм стеку?  
   Фрейм стеку (stack frame) — це частина стеку, яка виділяється для виконання однієї функції під час її виклику. Кожна функція отримує власний фрейм стеку, який зберігає:  
   Параметри, передані функції.  
   Локальні змінні функції.  
   Адресу повернення, яка вказує, куди потрібно повернутися після завершення функції.  
   Значення попереднього базового вказівника (EBP) для підтримки вкладеності викликів.  
     
   **Як формується фрейм стеку:**
2. **Перед викликом функції**:
   * Параметри передаються у стек.
   * Зберігається адреса повернення (адреса, на яку потрібно перейти після завершення функції).
3. **На початку функції**:
   * Поточне значення **EBP** зберігається у стеку.
   * **EBP** оновлюється, щоб вказувати на початок нового фрейму.
   * Локальні змінні виділяються у стеку за допомогою зміни значення **ESP**.
4. **Після завершення функції**:
   * Відновлюється значення **EBP** і адреса повернення.
   * Стек очищається від параметрів.