МІНІСТЕРСТВО ОСВІТИ І НАУКИ, МОЛОДІ ТА СПОРТУ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»
ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ
Кафедра Інформаційної Безпеки

Захист програмного забезпечення

Лабораторна робота 5

Автоматизований пошук вразливостей у вихідних текстах програмного забезпечення, що написані на мові високого рівня

Мета роботи:

- пошук потенційних вразливостей і помилок програмування в вихідних текстах на мові програмування C / C++ з використанням автоматизованого засобу виявлення поширених помилок програмування RATS;
- дослідження виявленої проблеми: визначення типу та категорії помилки, локалізація, розроблення пропозицій щодо усунення. Ранжування виявлених проблемних місць за ступенем серйозності.

Перевірив:

_____

Виконав:

студент III курсу

групи ФБ-01

Сахній Н.Р.

Київ 2023

Завдання:

/* Попередньо встановимо RATS із відповідного репозиторію GitHub */

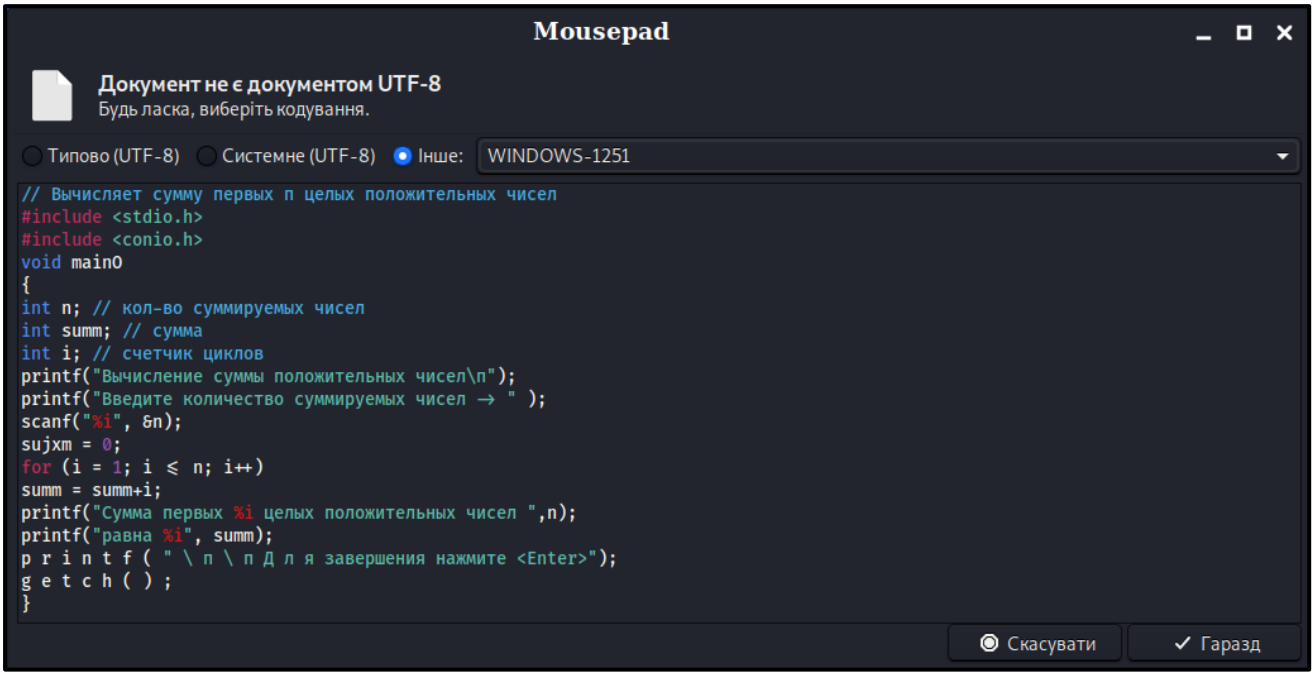
```
┌──(nazar㉿snz24)-[/home/nazar/KPI/RevEng]
└─$ git clone https://github.com/andrew-d/rough-auditing-tool-for-security.git
Cloning into 'rough-auditing-tool-for-security'...
remote: Enumerating objects: 89, done.
remote: Total 89 (delta 0), reused 0 (delta 0), pack-reused 89
Unpacking objects: 100% (89/89), 396.85 KiB | 1.64 MiB/s, done.

┌──(nazar㉿snz24)-[/home/nazar/KPI/RevEng]
└─$ cd rough-auditing-tool-for-security

┌──(nazar㉿snz24)-[/home/nazar/KPI/RevEng/rough-auditing-tool-for-security]
└─$ ./configure
checking for gcc... gcc
checking for C compiler default output... a.out
checking whether the C compiler works... yes
checking whether we are cross compiling... no
checking for suffix of executables...
checking for suffix of object files... o
checking whether we are using the GNU C compiler... yes
checking whether gcc accepts -g... yes
checking for gcc option to accept ANSI C... none needed
checking for flex... no
checking for lex... no
checking for yywrap in -lfl... no
checking for yywrap in -ll... no
checking for a BSD-compatible install... /usr/bin/install -c
checking for XML_ParserCreate in -lexpat... yes
checking how to run the C preprocessor... gcc -E
checking for ANSI C header files... yes
checking for sys/types.h... yes
checking for sys/stat.h... yes
checking for stdlib.h... yes
checking for string.h... yes
checking for memory.h... yes
checking for strings.h... yes
checking for inttypes.h... yes
checking for stdint.h... yes
checking for unistd.h... yes
checking expat.h usability... yes
checking expat.h presence... yes
checking for expat.h... yes
checking xmlparse.h usability... no
checking xmlparse.h presence... no
checking for xmlparse.h... no
configure: creating ./config.status
config.status: creating Makefile

┌──(nazar㉿snz24)-[/home/nazar/KPI/RevEng/rough-auditing-tool-for-security]
└─$ rats -h
RATS v2.4 - Rough Auditing Tool for Security
Copyright 2001, 2002 Secure Software Inc
http://www.securesoftware.com

usage: rats [-adhilrwxR] [--help] [--database|--db]  name1 name2 ... namen

    -a <fun>       report any occurence of function 'fun' in the source file(s)
    -d <filename>  specify an alternate vulnerability database.
    --db
    --database
    -h             display usage information (what you're reading)
    --help
    -i             report functions that accept external input
    --input
    -l <language>  force the specified langauge to be used
    --language <language>
    -r             include references that are not function calls
    --references
    -w <1,2,3>     set warning level (default 2)
    --warning <1,2,3>
    -x             do not load default databases
    -R             don't recurse subdirectories scanning for matching files
```

```
    --no-recursion
    --xml           Output in XML.
    --html          Output in HTML.
    --follow-symlinks
                    Follow symlinks and process files found.
    --noheader
                        Don't print initial header in output
    --nofooter
                    Don't show timing information footer at end of analysis
    --quiet
                    Don't print status information regarding what file is being analyzed
    --resultsonly
                    No header, footer, or status information
    --columns
                    Show column number of hte line where the problem occured.
    --context
                    Display the line of code that caused the problem report
```

1. ----------

- Оберемо файл на мові програмування, що підтримується утилітою RATS (наприклад, файли з ім'ям **`test{i}.c`**, що примутній в комплекті л/р);

– test.c



– test2.c

```
// вычисляет сопротивление цепи n-го порядка
float rcep(int n)
{
float r; // сопротивление цепи порядка n-1
if (n == 1)
return(r1 + r2 + r3);
else
{
r = rcep(n-1);
return (r1 + r2*r/(r2+r) + r3);
}
}
void main0
{
int n; // количество звеньев (порядок) цепи
float re; // сопротивление цепи
puts("ХпВычисление сопротивления электрической цепи");
puts("Введите величины сопротивлений (Ом):");
printf("r1 →");
scanf("%f", &r1);
printf("r2 →");
scanf("%f", &r2);
printf("r3 →");
scanf("%f", &r3);
printf("Порядок цепи →");
scanf("%i", &n);
```

```
re = rcep(n); // величины сопротивлений передаются
// функции rcep через глобальные
// переменные
printf("Сопротивление цепи:");
if (rc > 100)
{
rc /= 1000;
printf("%5.2f кОм\n", rc);
}
else
printf("%5.2f Ом\n", re);
puts("\nДля завершения нажмите <Enter>");
getch();
}
```

Скасувати ✓ Гаразд

– test3.c

/home/nazar/KPI/RevEng/test3.c - Mousepad

Файл Редагувати Пошук Перегляд Документ Допомога

```
1 /*
2  * Sample code with a few buffer overflows. (Plain file here)
3  */
4
5 #include <stdio.h>
6 #include <stdlib.h>
7
8 int main( int argc, char *argv[] )
9 {
10     char dir[1024];
11     char cmd[1200];
12     char buff[1024];
13     FILE *fp = NULL;
14     int i = 0;
15
16     if ( argc == 2 )
17     {
18       strcpy( dir, argv[ 1 ] );
19     }
20     else
21     {
22       if ( getenv( "HOME" ) != NULL )
23       {
24         sprintf( dir, "%s", getenv( "HOME" ) );
25       }
26       else
27       {
28         strcpy( dir, "/" );
29       }
```

```c
    }

    snprintf(cmd, sizeof(cmd)-1, "%s %s", "ls", dir );
    fp = popen( cmd, "r" );
    if ( fp == NULL )
    {
      printf("Failed to invoke: %s\n", cmd );
      return -1;
    }

    while( i = fread( buff, 1, sizeof(buff), fp ) )
    {
      printf( buff );
    }

    pclose( fp );

    return 0;
}
```

– test4.c

/home/nazar/KPI/RevEng/test4.c - Mousepad

Файл Редагувати Пошук Перегляд Документ Допомога

```c

#include <stdio.h>
#include<string.h>
int main()
{
   char str1[50];
   char str2[50];
   char str3[] =  " C Programming Language";
   char str4[50], str5[50];

   printf("\n Please enter the String you want to Copy: \n");
   gets(str1);

   strcpy(str2, str1);
   //puts(str1);
   puts(str2);

   strcpy(str4, str3);
   puts(str4);
   //puts(str3);

   strcpy(str5, " we provide free tutorials");
   puts(str5);
}
```

– test5.c

/home/nazar/KPI/RevEng/test5.c - Mousepad

Файл Редагувати Пошук Перегляд Документ Допомога

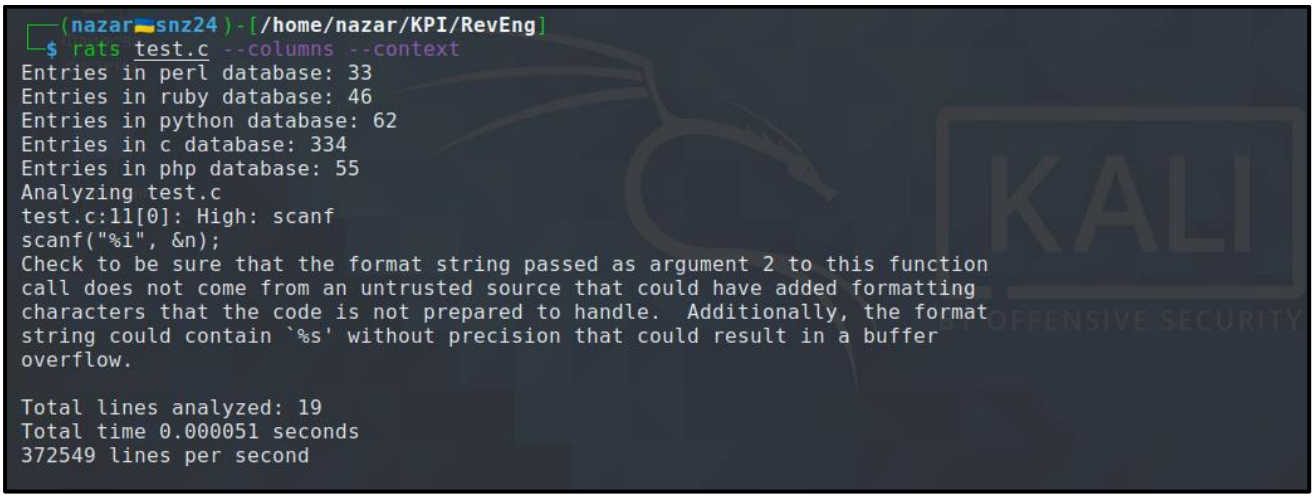
```c

#include <stdio.h>

int main()
{
        char name[50];

        printf("\n Please Enter your Full Name: \n");
        gets(name);

        printf("==========\n");
        printf("%s", name);

        return 0;
}
```

==========

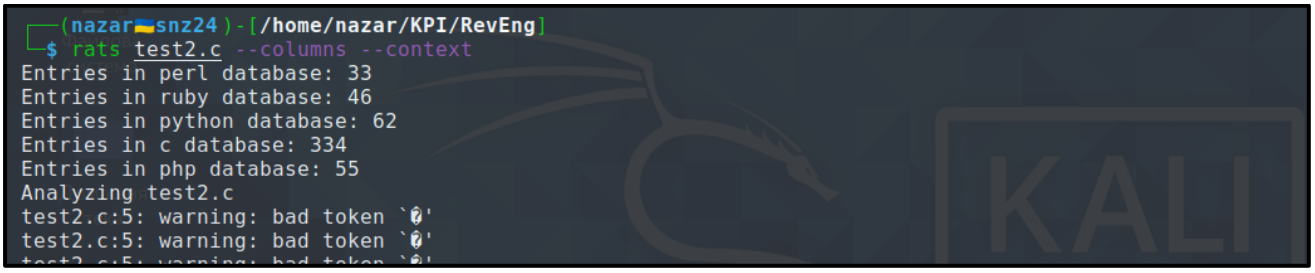
2. ----------

- Проскануємо даний файл:

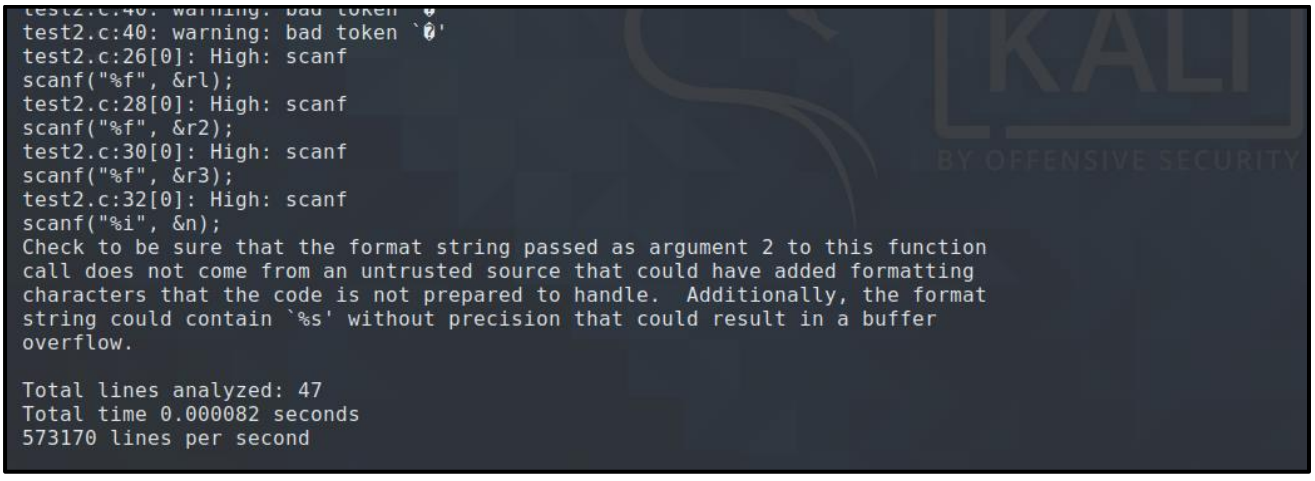
– test.c



– test2.c



.....



– test3.c

```
test3.c:10[5]: High: fixed size local buffer
    char dir[1024];
test3.c:11[5]: High: fixed size local buffer
    char cmd[1200];
test3.c:12[5]: High: fixed size local buffer
    char buff[1024];
Extra care should be taken to ensure that character arrays that are allocated
on the stack are used safely.  They are prime targets for buffer overflow
attacks.

test3.c:18[7]: High: strcpy
      strcpy( dir, argv[ 1 ] );
Check to be sure that argument 2 passed to this function call will not copy
more data than can be handled, resulting in a buffer overflow.
```

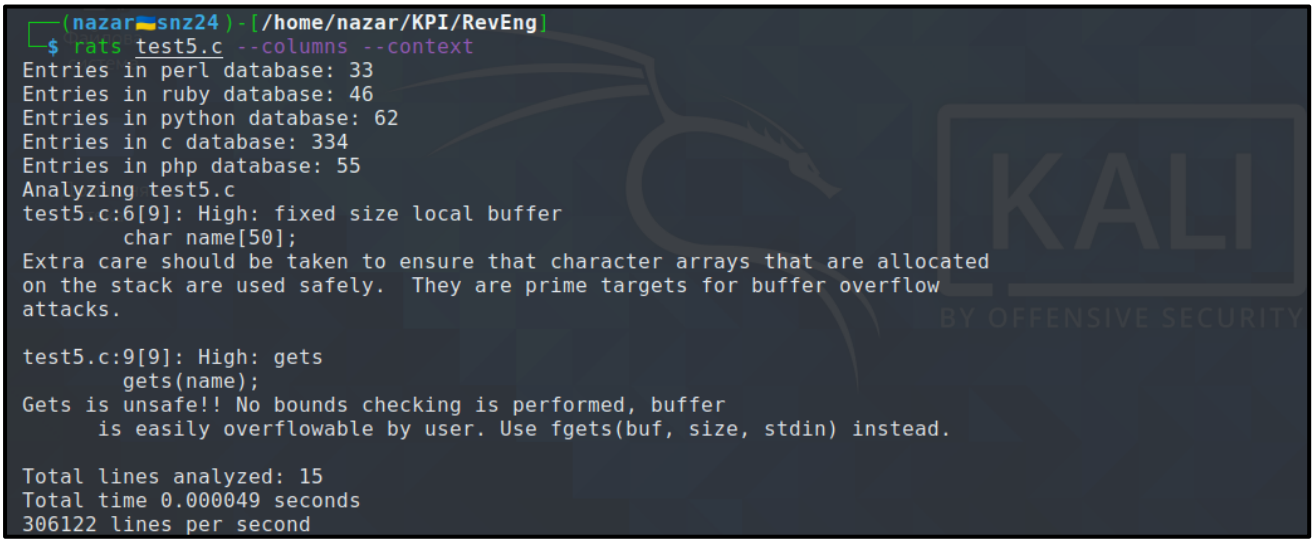
```
test3.c:22[12]: High: getenv
      if ( getenv( "HOME" ) != NULL )
test3.c:24[29]: High: getenv
       sprintf( dir, "%s", getenv( "HOME" ) );
Environment variables are highly untrustable input. They may be of any length, and contain any data. Do not make any assumptions regarding
 content or length. If at all possible avoid using them, and if it is necessary, sanitize them and truncate them to a reasonable length.

test3.c:24[9]: High: sprintf
       sprintf( dir, "%s", getenv( "HOME" ) );
Check to be sure that the format string passed as argument 2 to this function
call does not come from an untrusted source that could have added formatting
characters that the code is not prepared to handle.  Additionally, the format
string could contain `%s' without precision that could result in a buffer
overflow.

test3.c:33[10]: High: popen
    fp = popen( cmd, "r" );
Argument 1 to this function call should be checked to ensure that it does not
come from an untrusted source without first verifying that it contains nothing
dangerous.

test3.c:42[7]: High: printf
      printf( buff );
Check to be sure that the non-constant format string passed as argument 1 to
this function call does not come from an untrusted source that could have added
formatting characters that the code is not prepared to handle.

Total lines analyzed: 48
Total time 0.000058 seconds
827586 lines per second
```

– test4.c

```
┌──(nazar㉿snz24)-[/home/nazar/KPI/RevEng]
└─$ rats test4.c --columns --context
Entries in perl database: 33
Entries in ruby database: 46
Entries in python database: 62
Entries in c database: 334
Entries in php database: 55
Analyzing test4.c
test4.c:6[4]: High: fixed size local buffer
   char str1[50];
test4.c:7[4]: High: fixed size local buffer
   char str2[50];
test4.c:9[4]: High: fixed size local buffer
   char str4[50], str5[50];
Extra care should be taken to ensure that character arrays that are allocated
on the stack are used safely.  They are prime targets for buffer overflow
attacks.

test4.c:12[4]: High: gets
   gets(str1);
Gets is unsafe!! No bounds checking is performed, buffer
      is easily overflowable by user. Use fgets(buf, size, stdin) instead.
```

```
test4.c:12[4]: High: gets
   gets(str1);
Gets is unsafe!! No bounds checking is performed, buffer
      is easily overflowable by user. Use fgets(buf, size, stdin) instead.

test4.c:14[4]: High: strcpy
   strcpy(str2, str1);
test4.c:18[4]: High: strcpy
   strcpy(str4, str3);
Check to be sure that argument 2 passed to this function call will not copy
more data than can be handled, resulting in a buffer overflow.

Total lines analyzed: 24
Total time 0.000051 seconds
470588 lines per second
```

– test5.c



3. ----------

- Проаналізуємо вихідну інформацію утиліти:

Для початку, наголосимо на тому, що у кожному із наведених вище результатів аналізу файлів, ідентифікатор характеру кожної вразливості мав значення "**High**". Цей ідентифікатор вказує на серйозний рівень вразливості в програмному коді.

Ну, а далі вже пройдемось по кожному аналізу окрему та переглянемо знайдені вразливості. Коротко опишемо, що вони із себе представляють:

– test.c

❖ {High: **scanf**}: Використання неконтрольованих форматних рядків у функції “**scanf**” може призводити до переповнення буфера або некоректного опрацювання вхідних даних. Ця вразливість особливо небезпечна, якщо форматний рядок поступає з ненадійного джерела, оскільки це може призвести до її експлуатування зловмисником.

– test2.c

❖ {warning: **bad token `**&#xFFFD;**'** }: У звіті згадується кілька помилкових токенів (&#xFFFD;), що вказує на наявність проблеми зі синтаксисом або кодуванням файлу. Це може бути зумовлено некоректною обробкою символів, неправильною кодуванням файлу або відсутністю підтримки конкретного символу.

❖ {High: **scanf**}: Аналогічна вразливість, що і в попередньому програмному коді. Недостатня перевірка форматного рядка при зчитуванні даних за допомогою “**scanf**” може призвести до переповнення буфера або некоректного опрацювання вхідних даних.

- test3.c
  - {High: **fixed size local buffer**}: У коді використовуються статичні буфери з фіксованим розміром (як от "**char dir[1024]**"). Це може призводити до переповнення буфера, якщо вхідні дані будуть перевищувати заданий розмір буфера.
  - {High: **strcpy**}: Використання функції "**strcpy**" для копіювання рядків може призвести до переповнення буфера, якщо розмір вхідних даних перевищуватиме розмір цільового буфера, що потенційно може привести до виконання шкідливого коду.
  - {High: **getenv**}: Використання функції "**getenv**", яка призначена для отримання значення змінної середовища може бути небезпечною, оскільки значення змінної середовища може бути ненадійним або некоректним.
  - {High: **sprintf**}: Використання функції "**sprintf**" з неконтрольованим форматним рядком може призвести до переповнення буфера або некоректного опрацювання даних, якщо цей рядок містить '%s' без вказання точності або ж він поступає з ненадійного джерела.
  - {High: **popen**}: Використання функції "**popen**" з аргументами, які можуть походити з ненадійних джерел, може створювати потенційну вразливість. Необхідно перевіряти перший аргумент цього виклику функції, щоб переконатися, що він не містить небезпечних даних перед використанням.
  - {High: **printf**}: Використання функції "**printf**" з неконстантним форматним рядком може бути небезпечним, оскільки форматний рядок може поступати з ненадійного джерела. Недостатня перевірка форматного рядка може призвести до виконання небезпечного коду або некоректного форматування вихідних даних.

- test4.c
  - {High: **fixed size local buffer**}: Аналогічна вразливість, що і в попередньому випадку. Використання фіксованого розміру локальних масивів (у цьому випадку **str1[50]**, **str2[50]**, **str4[50]**, **str5[50]**) може призвести до переповнення буфера, якщо введені дані перевищують призначений розмір масиву.
  - {High: **gets**}: Наступна функція "**gets**" може бути небезпечною, так як вона не здійснює перевірку меж масиву. Користувач може ввести дані, які перевищують розмір буфера, що може призвести до переповнення буфера.
  - {High: **strcpy**}: Аналогічно, як і в попередньому випадку, використання функції "**strcpy**" для копіювання рядків (**str2**, **str4**) може призвести до переповнення цільового буфера, якщо джерело рядка перевищує розмір буфера.

- test5.c
  - {High: **fixed size local buffer**}: Отже, знову у цьому випадку було виявлено раніше уже згадану вразливість, що через наявність у програмному коді локального масиву **name** з довжиною фіксованого розміру 50 символів може відбутися переповнення буфера.
  - {High: **gets**}: Як раніше вже згадувалось, використання функції "**gets**" є небезпечним, так як вона не здійснює перевірку меж масиву. Тому рекомендується використовувати функцію **fgets(buf, size, stdin)** з попередньо вказаною межею буфера.

4. ----------

- Знайдемо позицію уразливості у вихідному коді та одразу ж виправимо її:

– test.c

```
10 printf("Введите количество суммируемых чисел → " );
11 scanf_s("%i", &n, sizeof(n)); // Before: scanf("%i", &n);
```

– test2.c

```
24 puts("Введите величины сопротивлений (Ом):");
25 printf("rl →");
26 scanf_s("%f", &rl, sizeof(rl)); // Before: scanf("%f", &rl);
27 printf("r2 →");
28 scanf_s("%f", &r2, sizeof(r2)); // Before: scanf("%f", &r2);
29 printf("r3 →");
30 scanf_s("%f", &r3, sizeof(r3)); // Before: scanf("%f", &r3);
31 printf("Порядок цепи →");
32 scanf_s("%i", &n, sizeof(n)); // Before: scanf("%i", &n);
```

– test3.c

```
10    char *dir = malloc(1024); // Before: char dir[1024];
11    char *cmd = malloc(1200); // Before: char cmd[1200];
12    char *buff = malloc(1024); // Before: char buff[1024];
```

```
/* Із статичного зробити буфер динамічним */
```

```
16    if ( argc == 2 )
17    {
18      strcpy_s( dir, argv[ 1 ] ); // Before: strcpy( dir, argv[ 1 ] );
19    }
20    else
21    {
22      if ( getenv_s( "HOME" ) != NULL ) // Before: if ( getenv( "HOME" ) != NULL )
23      {
24        sprintf_s( dir, "%s", getenv_s( "HOME" ) ); // Before: sprintf( dir, "%s", getenv( "HOME" ) );
25      }
26      else
27      {
28        strcpy_s( dir, "/" ); // Before: strcpy( dir, "/" );
29      }
30    }
31
32    snprintf(cmd, sizeof(cmd)-1, "%s %s", "ls", dir );
33    fp = popen_s( cmd, "r" ); // Before: fp = popen( cmd, "r" );
34    if ( fp == NULL )
35    {
36      printf_s("Failed to invoke: %s\n", cmd ); // Before: printf("Failed to invoke: %s\n", cmd );
37      return -1;
38    }
39
40    while( i = fread( buff, 1, sizeof(buff), fp ) )
41    {
42      printf_s( buff ); // Before: printf( buff );
43    }
```

– test4.c

```
6   char str1 = malloc(50); // Before: char str1[50];
7   char str2= malloc(50); // Before: char str2[50];
8   char str3[] =  " C Programming Language";
9   char str4, str5 = malloc(50), malloc(50); // Before:  char str4[50], str5[50];
```

```
/* Із статичного зробити буфер динамічним */
```

```
11  printf_s("\n Please enter the String you want to Copy: \n");
12  gets_s(str1); // Before: gets(str1);
13
14  strcpy_s(str2, str1); // Before: strcpy(str2, str1);
15  //puts(str1);
16  puts(str2);
17
```

```
18    strcpy_s(str4, str3); // Before: strcpy(str4, str3);
19    puts(str4);
20    //puts(str3);
21
22    strcpy(str5, " we provide free tutorials");
23    puts(str5);
24 }
```

– test5.c

```
6       char name = malloc(50); // Before: char name[50];
```

/* Із статичного зробити буфер динамічним */

```
8       printf("\n Please Enter your Full Name: \n");
9       gets_s(name); // Before: gets(name);
```

- Виконаємо повторну RATS-перевірку, після усунення деяких вразливостей:

– test.c

```
┌──(nazar㉿snz24)-[/home/nazar/KPI/RevEng]
└─$ rats test.c --columns --context
Entries in perl database: 33
Entries in ruby database: 46
Entries in python database: 62
Entries in c database: 334
Entries in php database: 55
Analyzing test.c
Total lines analyzed: 19
Total time 0.000056 seconds
339285 lines per second
```

– test2.c

```
┌──(nazar㉿snz24)-[/home/nazar/KPI/RevEng]
└─$ rats test2.c --columns --context
Entries in perl database: 33
Entries in ruby database: 46
Entries in python database: 62
Entries in c database: 334
Entries in php database: 55
Analyzing test2.c
test2.c:5: warning: bad token `'
test2.c:5: warning: bad token `'
test2.c:5: warning: bad token `'
test2.c:5: warning: bad token `'
test2.c:8: warning: bad token `'
test2.c:8: warning: bad token `'
test2.c:8: warning: bad token `'
test2.c:8: warning: bad token `'
test2.c:8: warning: bad token `'
test2.c:10: warning: bad token `'
test2.c:12: warning: bad token `'
test2.c:12: warning: bad token `'
test2.c:12: warning: bad token `'
test2.c:15: warning: bad token `'
test2.c:37: warning: bad token `'
test2.c:37: warning: bad token `'
test2.c:39: warning: bad token `'
test2.c:39: warning: bad token `'
test2.c:40: warning: bad token `'
test2.c:40: warning: bad token `'
Total lines analyzed: 47
Total time 0.000189 seconds
248677 lines per second
```

– test3.c

```
┌──(nazar㉿snz24)-[/home/nazar/KPI/RevEng]
└─$ rats test3.c --columns --context
Entries in perl database: 33
Entries in ruby database: 46
Entries in python database: 62
Entries in c database: 334
Entries in php database: 55
Analyzing test3.c
Total lines analyzed: 48
Total time 0.000056 seconds
857142 lines per second
```

– test4.c

```
┌──(nazar㉿snz24)-[/home/nazar/KPI/RevEng]
└─$ rats test4.c --columns --context
Entries in perl database: 33
Entries in ruby database: 46
Entries in python database: 62
Entries in c database: 334
Entries in php database: 55
Analyzing test4.c
Total lines analyzed: 24
Total time 0.000055 seconds
436363 lines per second
```

– test5.c

```
┌──(nazar㉿snz24)-[/home/nazar/KPI/RevEng]
└─$ rats test5.c --columns --context
Entries in perl database: 33
Entries in ruby database: 46
Entries in python database: 62
Entries in c database: 334
Entries in php database: 55
Analyzing test5.c
Total lines analyzed: 15
Total time 0.000237 seconds
63291 lines per second
```