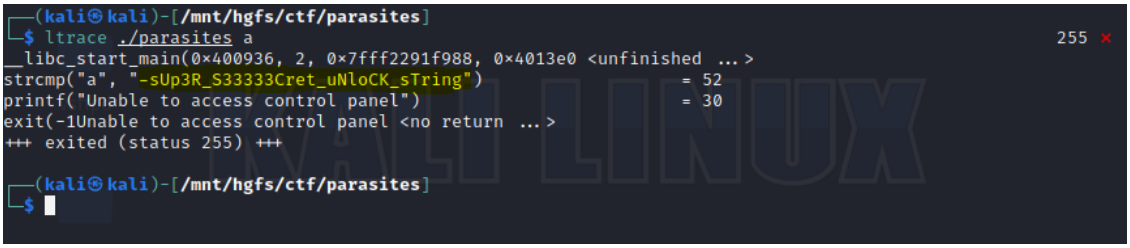
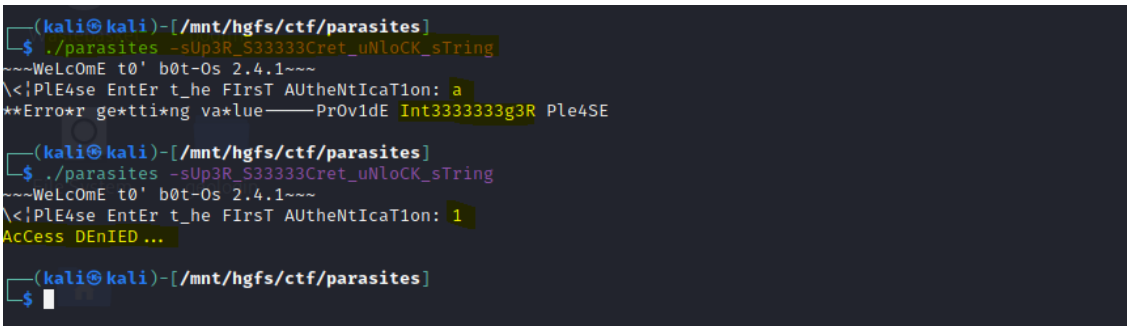
1. If we execute the program without any arguments, the program instantly exists & an error message appears when an argument is entered.

2. Use ltrace to track the library calls made by the program. A string comparison is made with "-sUp3R_S33333Cret_uNloCK_sTring".



3. Add the string from Step 2 as an argument. If a string/character is entered now, the program requests an integer. If an (/the wrong) integer is entered, the program exits with an "Access Denied" Message.



4. Now try running the program (including the string from the last step) with strace to trace the system calls / signals and enter an integer.
   The program exits because of an invalid instruction (SIGILL) & then a lot of ptrace calls are noticeable (especially PTRACE_PEEKDATA). This might seem like nothing interesting at first, but it is important to keep in mind when analyzing the code.

```
clone(child_stack=NULL, flags=CLONE_CHILD_CLEARTID|CLONE_CHILD_SETTID|SIGCHLD, child_tidptr=0×7f86ea08e810)
 = 3471
wait4(3471, ~~~WeLcOmE t0' b0t-Os 2.4.1~~~
\<¦PlE4se EntEr t_he FIrsT AUtheNtIcaT1on: 1
[{WIFSTOPPED(s) && WSTOPSIG(s) == SIGILL}], 0, NULL) = 3471
--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_TRAPPED, si_pid=3471, si_uid=1000, si_status=SIGILL, si_utime=0,
 si_stime=0} ---
ptrace(PTRACE_GETREGS, 3471, NULL, 0×7ffe31c80e40) = 0
ptrace(PTRACE_PEEKDATA, 3471, 0×400c24, [0×48e5894855c30b0f]) = 0
ptrace(PTRACE_SETREGS, 3471, NULL, 0×7ffe31c80e40) = 0
ptrace(PTRACE_CONT, 3471, NULL, 0AcCess DEnIED ... )      = 0
--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_EXITED, si_pid=3471, si_uid=1000, si_status=57, si_utime=0, si_s
time=0} ---
wait4(3471, [{WIFEXITED(s) && WEXITSTATUS(s) == 57}], 0, NULL) = 3471
ptrace(PTRACE_CONT, 3471, NULL, 0)      = -1 ESRCH (No such process)
exit_group(0)                           = ?
+++ exited with 0 +++
```

5. Open the program in Ghidra (https://ghidra-sre.org/) & analyze the main() function. The string comparison can be seen that was displayed when executing the program with ltrace. On successful comparison, the Parent() function is called.
   Analyze the Parent() function. A child process is created with fork() & the Child() function is called. The parent process then waits (waitpid) until the child process changes its state. At some point HandleException() might be called.

```
undefined8 main(int param_1,long param_2)

{
  int iVar1;
  undefined8 uVar2;
  long in_FS_OFFSET;
  undefined8 local_38;
  undefined8 local_30;
  undefined8 local_28;
  undefined8 local_20;
  long local_10;

  local_10 = *(long *)(in_FS_OFFSET + 0x28);
  local_38 = 0x535f52337055732d;
  local_30 = 0x6572433333333333;
  local_28 = 0x4b436f6c4e755f74;
  local_20 = 0x676e697254735f;
  if (param_1 < 2) {
    uVar2 = 0;
  }
  else {
    iVar1 = strcmp(*(char **)(param_2 + 8),(char *)&local_38);
    if (iVar1 != 0) {
      printf("Unable to access control panel");
                    /* WARNING: Subroutine does not return */
      exit(-1);
    }
    uVar2 = Parent();
  }
  if (local_10 != *(long *)(in_FS_OFFSET + 0x28)) {
                    /* WARNING: Subroutine does not return */
    __stack_chk_fail();
  }
  return uVar2;
}
```

Main()

```
undefined8 Parent(void)

{
  bool bVar1;
  undefined8 uVar2;
  long in_FS_OFFSET;
  uint local_24;
  uint local_20;
  __pid_t local_1c;
  uint local_18;
  int local_14;
  long local_10;

  local_10 = *(long *)(in_FS_OFFSET + 0x28);
  local_20 = fork();
  if (local_20 == 0) {
    ptrace(PTRACE_TRACEME,0,0,0);
    uVar2 = Child();
  }
  else {
    bVar1 = false;
    while (!bVar1) {
      local_24 = 0;
      local_1c = waitpid(local_20,(int *)&local_24,0);
      if ((local_24 & 0x7f) == 0) {
        bVar1 = true;
      }
      else {
        if ((char)(((byte)local_24 & 0x7f) + 1) >> 1 < '\x01') {
          if (((local_24 & 0xff) == 0x7f) &&
             (local_14 = (int)(local_24 & 0xff00) >> 8, local_14 == 4
            HandleException(local_20);
          }
        }
        else {
          local_18 = local_24 & 0x7f;
          bVar1 = true;
        }
      }
      ptrace(PTRACE_CONT,(ulong)local_20,0,0);
    }
    uVar2 = 0;
  }
```

Parent()

6. Analyze the Child() function. The Code repeats itself (Authentication Step 1 until 5 + Final) but follows the same principle. The Dispatch() function is called & the output is checked with a hex value (*0x4babd7ac*).
   Analyze the Dispatch() function. The UD2 Assembly Instruction raises an invalid opcode exception.

```
local_10 = *(long *)(in_FS_OFFSET + 0x28);
puts("~~~WeLcOmE t0\' b0t-Os 2.4.1~~~");
printf(&DAT_004014b0);
iVar1 = scanf("%d",&local_94);
if (iVar1 != 1) {
  printf("**Erro*r ge*tti*ng va*lue-----PrOv1dE Int3333333g3R P1e4SE");
                    /* WARNING: Subroutine does not return */
  exit(0x539);
}
local_78 = Dispatch(local_94,0x499602d2);
if (local_78 != 0x4babd7ac) {
  printf("AcCess DEnIED...");
                    /* WARNING: Subroutine does not return */
  exit(0x539);
}
sprintf(local_38,"%i",(ulong)local_94);
local_90 = 0;
while (local_90 < 8) {
  sprintf(local_48,"%c%c",(ulong)(uint)(int)local_38[local_90],
          (ulong)(uint)(int)local_38[local_90 + 1]);
  lVar2 = strtol(local_48,(char **)0x0,10);
  local_74 = (int)lVar2;
  putchar(local_74);
  fflush(stdout);
  local_90 = local_90 + 2;
}
```

Child()

```
*************************************************
*                          FUNCTION
*************************************************
undefined Dispatch()
  AL:1           <RETURN>
Dispatch

     UD2
     ??         C3h
```

Dispatch()

7. Finally, analyze the HandleException() function called by Parent(). It takes the PID of the child as a parameter & gets some data from it. This data is XORed with a hex value (*0x4fb30a91*).

```
void HandleException(int param_1)

{
  long lVar1;
  undefined8 *puVar2;
  long in_FS_OFFSET;
  undefined8 local_3a8 [10];
  long local_358;
  undefined8 local_338;
  long local_328;
  char local_18;
  char local_17;
  long local_10;

  local_10 = *(long *)(in_FS_OFFSET + 0x28);
  lVar1 = 0x72;
  puVar2 = local_3a8;
  while (lVar1 != 0) {
    lVar1 = lVar1 + -1;
    *puVar2 = 0;
    puVar2 = puVar2 + 1;
  }
  ptrace(PTRACE_GETREGS,(ulong)(uint)param_1,0,local_3a8);
  lVar1 = ptrace(PTRACE_PEEKDATA,(ulong)(uint)param_1,local_328,0);
  local_18 = (char)lVar1;
  local_17 = (char)((ulong)lVar1 >> 8);
  if ((local_18 == '\x0f') && (local_17 == '\v')) {
    local_358 = (long)(int)((uint)local_338 ^ 0x4fb30a91);
    local_328 = local_328 + 2;
    ptrace(PTRACE_SETREGS,(ulong)(uint)param_1,0,local_3a8);
  }
  if (local_10 != *(long *)(in_FS_OFFSET + 0x28)) {
                    /* WARNING: Subroutine does not return */
    __stack_chk_fail();
  }
  return;
}
```

8. We can see that the ptrace calls made by the HandleException function are identical to the ptrace calls that was seen when executing the program in strace.

   We can assume the following: When an input (integer) is entered, the Dispatch function returns an exception that causes the child process to exit. The HandleException fetches the input (with PTRACE_PEEKDATA) and XORs it with the hex value *0x4fb30a91*. This means, we can XOR the hex values in the Child() function with *0x4fb30a91.*

9. Write a short python script to print all valid numbers (*get_valid_nums.py*).

```
┌──(kali㉿kali)-[/mnt/hgfs/ctf/parasites]
└─$ python3 get_valid_nums.py
Solution Authentication 1: 68738365
Solution Authentication 2: 66766995
Solution Authentication 3: 84726995
Solution Authentication 4: 78657879
Solution Authentication 5: 77738469
Solution Authentication 6: 83954936
```

10. Enter the numbers & concatenate all flag parts printed out by the program. The concatenated string wrapped with “shc2021{}” is the flag.

```
┌──(kali㉿kali)-[/mnt/hgfs/ctf/parasites]
└─$ ./parasites -sUp3R_S33333Cret_uNloCK_sTring
~~~WeLcOmE t0' b0t-Os 2.4.1~~~
\<¦PlE4se EntEr t_he FIrsT AUtheNtIcaT1on: 68738365
DISA
.::.::.PlE4se EntEr AUtheNtIcaT1on PaRT 2: 66766995
BLE_
PlE4se EntEr AUtheNtIcaT1on PaRT 3: 84726995
THE_
^3——ç*PlE4se EntEr AUtheNtIcaT1on PaRT 4: 78657879
NANO
————————PlE4se EntEr AUtheNtIcaT1on PaRT 5: 77738469
MITE
*~+*^*PlE4se EntEr F1NAl AUtheNtIcaT1on:83954936
S_1$
UnLocKed B0tS cOntroL SySTem!
[*] D1sAble AutonOmous DeFen5e Bot5 and opEN d0oRS

UnLock1ng DooRs ... iN PROgrEs
UNLocKing doOrs ... SucCessFUL!
DiSarm1ng BoTS ... in ProgreS
disArming bOts ... succEssFul!
YoU aRe fREE to LeAve HuMan
```