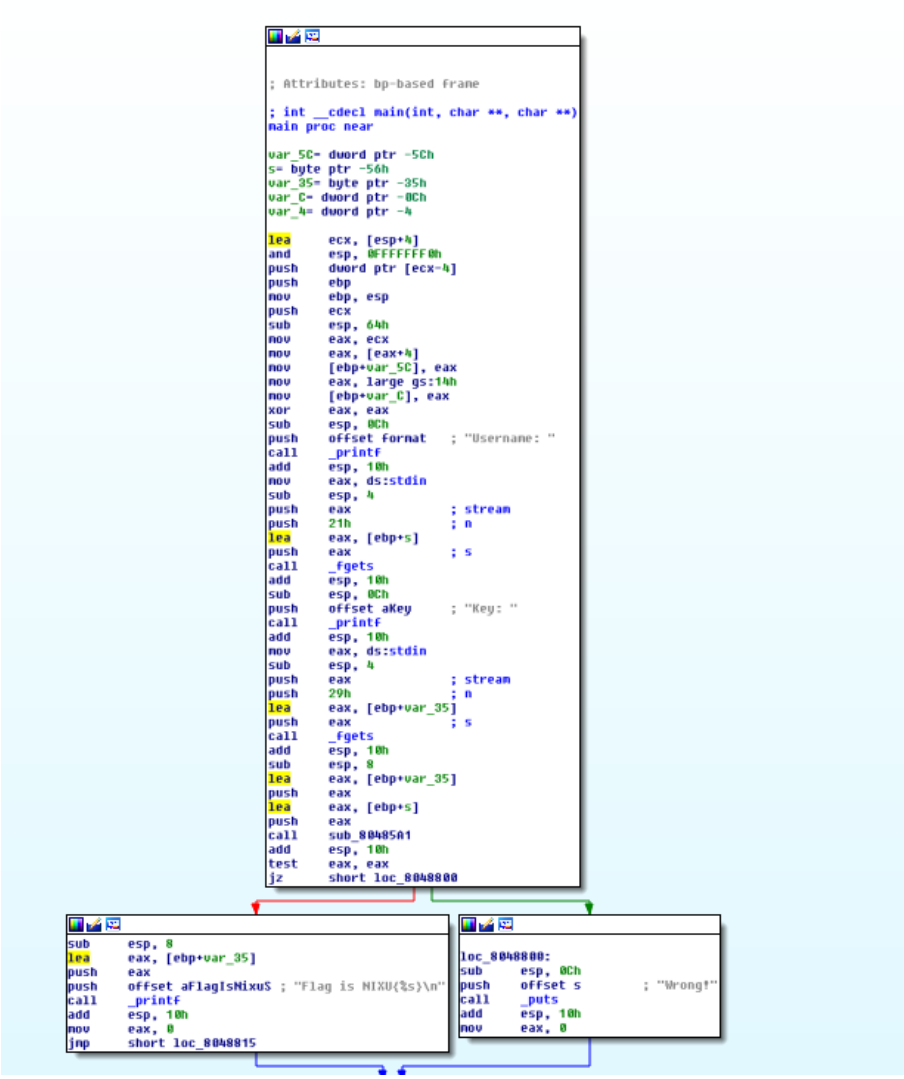
Crackme1 Writeup

I started the challenge by loading the binary in IDA Pro.



We can see that it does a logical compare on eax and jumps if zero to either “wrong” or prints the flag. I tried my luck by just patching the instruction to “jnz” to see if we could print the flag. After a while I noticed that it was printing the password we input in main.

So, I took a deeper look into the main and generated some pseudocode.

```
int __cdecl main()
{
  int result; // eax@2
  int v1; // edx@4
  char s; // [sp+12h] [bp-56h]@1
  char v3; // [sp+33h] [bp-35h]@1
  int v4; // [sp+5Ch] [bp-Ch]@1

  v4 = *MK_FP(__GS__, 20);
  printf("Username: ");
  fgets(&s, 33, stdin);
  printf("Key: ");
  fgets(&v3, 41, stdin);
  if ( sub_80485A1(&s, &v3) )
  {
    printf("Flag is NIXU{%s}\n", &v3);
    result = 0;
  }
  else
  {
    puts("Wrong!");
    result = 0;
  }
  v1 = *MK_FP(__GS__, 20) ^ v4;
  return result;
}
```

After a little renaming it was a bit clearer

```
int __cdecl main()
{
  int result; // eax@2
  int f_ptr; // edx@4
  char username; // [sp+12h] [bp-56h]@1
  char password; // [sp+33h] [bp-35h]@1
  int f_ptr2; // [sp+5Ch] [bp-Ch]@1

  f_ptr2 = *MK_FP(__GS__, 20);
  printf("Username: ");
  fgets(&username, 33, stdin);
  printf("Key: ");
  fgets(&password, 41, stdin);
  if ( compare(&username, &password) )
  {
    printf("Flag is NIXU{%s}\n", &password);
    result = 0;
  }
  else
  {
    puts("Wrong!");
    result = 0;
  }
  f_ptr = *MK_FP(__GS__, 20) ^ f_ptr2;
  return result;
}
```

Then I went to the sub routine I named “compare” which had a lot more going on.

```
signed int __cdecl compare(int a1, int a2)
{
  signed int result; // eax@7
  char v3; // ST23_1@22
  int v4; // ecx@27
  char v5; // [sp+12h] [bp-26h]@16
  char v6; // [sp+12h] [bp-26h]@21
  int v7; // [sp+14h] [bp-24h]@22
  int i; // [sp+18h] [bp-20h]@1
  int j; // [sp+1Ch] [bp-1Ch]@8
  int k; // [sp+20h] [bp-18h]@16
  int l; // [sp+24h] [bp-14h]@21
  int v12; // [sp+28h] [bp-10h]@22
  int v13; // [sp+2Ch] [bp-Ch]@1

  v13 = *MK_FP(__GS__, 20);
  for ( i = 0; *(_BYTE *)(i + a1) && *(_BYTE *)(i + a1) != 13 && *(_BYTE *)(i + a1) != 10; ++i )
    ;
  *(_BYTE *)(i + a1) = 0;
  if ( i == 10 )
  {
    for ( j = 0; j <= 39; ++j )
    {
      if ( *(_BYTE *)(j + a2) <= 47 || *(_BYTE *)(j + a2) > 57 && (*(_BYTE *)(j + a2) <= 96 || *(_BYTE *)(j + a2) > 102) )
      {
        result = 0;
        goto LABEL_27;
      }
    }
    v5 = -86;
    for ( k = 0; k < i; ++k )
    {
      v5 = sub_804853B(k, *(_BYTE *)(k + a1), (unsigned __int8)v5);
      if ( *((_BYTE *)off_804A02C + k) != v5 )
      {
        result = 0;
        goto LABEL_27;
      }
    }
    v6 = -1;
    for ( l = 0; l <= 19; ++l )
    {
      v12 = 2 * l + a2;
      __isoc99_sscanf(v12, (const char *)&unk_80488BB, &v7);
      v3 = v7;
      v6 = sub_804853B(l, *(_BYTE *)(l % i + a1), (unsigned __int8)v6);
      if ( v3 != v6 )
      {
        result = 0;
        goto LABEL_27;
      }
```

After some analysis on the pseudocode we can determine that username is a 10 sized char array and can consist any letter or number. and password is a 40 length and contains only hex.

```
f_ptr = *MK_FP(__GS__, 20);
for ( i = 0; username[i] && username[i] != '\r' && username[i] != '\n'; ++i )
  ;
username[i] = 0;
if ( i == 10 )
{
  for ( j = 0; j <= 39; ++j )
  {
    if ( password[j] <= '/' || password[j] > '9' && (password[j] <= '`' || password[j] > 'f') )
    {
      result = 0;
      goto LABEL_27;
    }
  }
  LOBYTE(xorkey) = -86;
  for ( index = 0; index < i; ++index )
  {
    LOBYTE(xorkey) = xor(index, username[index], xorkey);
    if ( encUname[index] != xorkey )
    {
      result = 0;
      goto LABEL_27;
    }
  }
  LOBYTE(xorkey) = 255;
  for ( l = 0; l <= 19; ++l )
  {
    buffer = &password[2 * l];
    __isoc99_sscanf(buffer, "%2x", &xorkey + 2);
    BYTE1(xorkey) = BYTE2(xorkey);
    LOBYTE(xorkey) = xor(l, username[l % i], xorkey);
    if ( BYTE1(xorkey) != xorkey )
    {
      result = 0;
      goto LABEL_27;
    }
  }
  result = 1;
}
else
{
  result = 0;
}
```

After some analysis I found out that sub_804953b is a xor routine and off_804a02c was the encrypted username

```
int __cdecl xor(unsigned int index, char letter, BYTE secret)
{
  return (secret ^ ((letter >> index % 8) | (letter << (8 - index % 8))));
}
```

```
encUname        dd offset EncUname      ; DATA XREF: compare+FA↑r
_data           ends                    ; "×¼¸Ð7«j\f?ñ"
```

Now that we have the encrypted username, we can write a script to force it. Starting secret is -86 and new secret is the outcome of the xor routine. so, we have all the secrets and the length. After some brute forcing the password turned out to be “4dm1n31337”. Password seems to be dereferenced password index times 2 which is copied to a buffer with the format “%2x”. That tells us that it is a memory address, so the hexadecimal bit makes sense now.

```
for ( a1 = 0; a1 <= 19; ++a1 )
{
  buffer = &password[2 * a1];
  __isoc99_sscanf(buffer, "%2x", &output);
  cmpKey = output;
  xorkey_1 = xor(a1, username[a1 % i], xorkey_1);
  if ( cmpKey != xorkey_1 )
  {
    result = 0;
    goto LABEL_27;
  }
}
result = 1;
```

so, the password memory offset is compared to the xor routine output of username [index % 10]. So, we have all the info we need to crack the password. After running the xor routine with the key from the source and input accordingly we got the following “cbf9a28462fb3f596af1fc70a62f96f0c15894” and the flag was NIXU{cbf9a28462fb3f596af1fc70a62f96f0c15894}.