undo

This binary encodes our input data and is compared against a buffer. We can use Ghidra to determine how the flag is encrypted and reverse the encryption to get the flag.

This will be a good introduction to reversing obfuscated functions in Ghidra.

Running `checksec` on the binary we notice all security features are enabled:

```
$ checksec undo
[*] '/home/joybuzzer/Documents/vunrotc/public/reverse-engineering/11-ghidra/undo/src/undo'
    Arch:     i386-32-little
    RELRO:    Full RELRO
    Stack:    Canary found
    NX:       NX enabled
    PIE:      PIE enabled
```

Static Analysis

Let's check the raw decompilation of `main()`:

```
void main(undefined param_1)
{
  int in_GS_OFFSET;
  int local_44;
  char local_3d [41];
  int local_14;
  undefined1 *local_10;

  local_10 = &param_1;
  local_14 = *(int *)(in_GS_OFFSET + 0x14);
  puts("Enter your password");
  fflush(_stdout);
  fgets(local_3d,0x29,_stdin);
  printf("You entered: %s\n",local_3d);
  fflush(_stdout);
  encode((int)local_3d);
  local_44 = 0;
  do {
    if (0x28 < local_44) {
      win();
LAB_00011382:
      if (local_14 != *(int *)(in_GS_OFFSET + 0x14)) {
        __stack_chk_fail_local();
      }
      return;
    }
```

```
      if (enc_key[local_44] != buffer[local_44]) {
        puts("You lose!");
        goto LAB_00011382;
      }
      local_44 = local_44 + 1;
    } while( true );
  }
```

This code is a **mess**. There is a label in the middle of the function, a do-while loop, and lots of wonky variables. Let's discuss how we can clean this up:

- First, we'll rename variables to something more meaningful. This immediately makes the code easier to read.
- Second, we'll remove the label by moving the code after the label to the jump.
- Third, we can remove the canary check. We know it's there and it doesn't affect control flow.
- We notice that `param_1` is actually never used, so we'll remove it. This changes the signature to `void main(void)`.
- Once this is done, we'll remove variables we no longer need and variables that aren't used.

After these changes, we get the following code:

```
void main(void)
{
  int i;
  char buf[41];

  puts("Enter your password");
  fflush(stdout);
  fgets(buf, 0x29, stdin);
  printf("You entered: %s\n",buf);
  fflush(stdout);
  encode((int)buf);
  i = 0;
  do {
    if (0x28 < i) {
      win();
      return;
    }
    if (enc_key[i] != buffer[i]) {
      puts("You lose!");
      return;
    }
    i = i + 1;
  } while( true );
}
```

This is a lot easier to read. We can simplify this further by switching the do-while loop for a `while` loop where the condition is checked.

```
while (i <= 0x28) {
    if (enc_key[i] != buffer[i]) {
      puts("You lose!");
      return;
    }
    i = i + 1;
}
win();
```

> *Make sure you understand how we made this conversion.* This is a tough thing to analyze and change. Although it's not required to rearrange the loop, it makes the code easier to read.

This means that we're checking the first `0x28 = 40` bytes of `enc_key` against `buffer`. If they match, we call `win()`. If they don't match, we call `puts("You lose!")` and return.

We have a few things to check out:

- What is `enc_key`?
- What is `buffer`?
- What does the `encode()` function do?

If we look at `enc_key` and `buffer`, we notice these variables are colored *aqua*. This means they are **global** variables. Local variables are colored *yellow*. Double-clicking on `enc_key` and `buffer` takes us to the location where they are defined.

This is the definition of `enc_key`:

```
                                enc_key
        00014080                    undefine    ??
00014080                 undefined1??                    [0]
00014081                 undefined1??                    [1]
00014082                 undefined1??                    [2]
00014083                 undefined1??                    [3]
00014084                 undefined1??                    [4]
00014085                 undefined1??                    [5]
00014086                 undefined1??                    [6]
00014087                 undefined1??                    [7]
00014088                 undefined1??                    [8]
00014089                 undefined1??                    [9]
0001408a                 undefined1??                    [10]
0001408b                 undefined1??                    [11]
0001408c                 undefined1??                    [12]
0001408d                 undefined1??                    [13]
0001408e                 undefined1??                    [14]
0001408f                 undefined1??                    [15]
00014090                 undefined1??                    [16]
00014091                 undefined1??                    [17]
00014092                 undefined1??                    [18]
00014093                 undefined1??                    [19]
00014094                 undefined1??                    [20]
```

```
00014095                    undefined1??                    [21]
00014096                    undefined1??                    [22]
00014097                    undefined1??                    [23]
00014098                    undefined1??                    [24]
00014099                    undefined1??                    [25]
0001409a                    undefined1??                    [26]
0001409b                    undefined1??                    [27]
0001409c                    undefined1??                    [28]
0001409d                    undefined1??                    [29]
0001409e                    undefined1??                    [30]
0001409f                    undefined1??                    [31]
000140a0                    undefined1??                    [32]
000140a1                    undefined1??                    [33]
000140a2                    undefined1??                    [34]
000140a3                    undefined1??                    [35]
000140a4                    undefined1??                    [36]
000140a5                    undefined1??                    [37]
000140a6                    undefined1??                    [38]
000140a7                    undefined1??                    [39]
```

We notice that `enc_key` is `40` bytes big and starts at `0x14080`. The data in this array is undefined.

> *Why?* We'll notice that the array is defined during `encode()`. The program has not started running, but since the variable is global, the space is allocated at compile-time.

Now, let's check `buffer`:

```
                                buffer
        00014020 c7 65 2b        undefine
                 7d 47 d3
                 fb 5f 30
00014020 c7                 undefined1C7h                   [0]
00014021 65                 undefined165h                   [1]
00014022 2b                 undefined12Bh                   [2]
00014023 7d                 undefined17Dh                   [3]
00014024 47                 undefined147h                   [4]
00014025 d3                 undefined1D3h                   [5]
00014026 fb                 undefined1FBh                   [6]
00014027 5f                 undefined15Fh                   [7]
00014028 30                 undefined130h                   [8]
00014029 80                 undefined180h                   [9]
0001402a 1d                 undefined11Dh                   [10]
0001402b e3                 undefined1E3h                   [11]
0001402c 23                 undefined123h                   [12]
0001402d 59                 undefined159h                   [13]
0001402e 34                 undefined134h                   [14]
0001402f db                 undefined1DBh                   [15]
00014030 ab                 undefined1ABh                   [16]
00014031 48                 undefined148h                   [17]
00014032 ed                 undefined1EDh                   [18]
00014033 93                 undefined193h                   [19]
00014034 49                 undefined149h                   [20]
```

```
00014035 b0            undefined1B0h            [21]
00014036 68            undefined168h            [22]
00014037 30            undefined130h            [23]
00014038 9e            undefined19Eh            [24]
00014039 8d            undefined18Dh            [25]
0001403a 37            undefined137h            [26]
0001403b 1c            undefined11Ch            [27]
0001403c 7d            undefined17Dh            [28]
0001403d 48            undefined148h            [29]
0001403e 7b            undefined17Bh            [30]
0001403f 6f            undefined16Fh            [31]
00014040 34            undefined134h            [32]
00014041 45            undefined145h            [33]
00014042 e7            undefined1E7h            [34]
00014043 ca            undefined1CAh            [35]
00014044 9f            undefined19Fh            [36]
00014045 8e            undefined18Eh            [37]
00014046 50            undefined150h            [38]
00014047 cb            undefined1CBh            [39]
```

On the left side, we see this data is initialized. The second column of data is the value at each byte. We can actually get Ghidra to copy this as a Python List using `Right Click -> Copy Special -> Python List`. With no extra effort, here is the value of `buffer`:

```
buffer = [ 0xc7, 0x65, 0x2b, 0x7d, 0x47, 0xd3, 0xfb, 0x5f, 0x30, 0x80,
0x1d, 0xe3, 0x23, 0x59, 0x34, 0xdb, 0xab, 0x48, 0xed, 0x93, 0x49, 0xb0,
0x68, 0x30, 0x9e, 0x8d, 0x37, 0x1c, 0x7d, 0x48, 0x7b, 0x6f, 0x34, 0x45,
0xe7, 0xca, 0x9f, 0x8e, 0x50, 0xcb ]
```

Now that we have our data, we can look at `encode()`:

```
undefined1 * encode(int param_1)
{
  int local_c;

  for (local_c = 0; *(char *)(param_1 + local_c) != '\0'; local_c = local_c
+ 1) {
    enc_key[local_c] = (*(byte *)(param_1 + local_c) ^ 0x55) + 8;
  }
  return enc_key;
}
```

From this, we notice three things:

- As it stands, `encode()` takes an `int` argument.
- `encode()` must cast the input to an `int`, and then casts it again inside the function.
- Ghidra doesn't know the return type.

From this, we can gather that **Ghidra got the paramter type wrong**. We can help out Ghidra by changing the type to what we think it is. It appears that this function is casting to a `char*` and a `byte*`. A `byte*` is simply a signed `char*`. Let's change the type in Ghidra to `char*` and see what happens. At the same time, we know that `enc_key` is a `char*`, so we'll change the return type too.

```
char* encode(char *param_1)
{
  int i;

  for (i = 0; param_1[i] != '\0'; i = i + 1) {
    enc_key[i] = (param_1[i] ^ 0x55U) + 8;
  }
  return enc_key;
}
```

This is a lot better. This function takes a string and performs byte-wise operations on each byte and stores it in `enc_key`. It performs the following operation:

```
out = (in ^ 0x55) + 8
```

Both these operations are undoable, meaning:

```
in = (out - 8) ^ 0x55
```

This is our key to solving this problem! If we take the ouput `buffer`, and perform this operation on each byte, we'll get the correct input.

Writing a Solve Script

The first thing we must do is write a `decode()` function that reverses the input. This will take in the list and return a list with the decoded bytes.

```
def decode(in_list):
    out_list = []
    for i in in_list:
        out_list.append((i - 8) ^ 0x55)
    return out_list
```

We'll take our `buffer` and decode it:

```
buffer = [ 0xc7, 0x65, 0x2b, 0x7d, 0x47, 0xd3, 0xfb, 0x5f, 0x30, 0x80,
0x1d, 0xe3, 0x23, 0x59, 0x34, 0xdb, 0xab, 0x48, 0xed, 0x93, 0x49, 0xb0,
0x68, 0x30, 0x9e, 0x8d, 0x37, 0x1c, 0x7d, 0x48, 0x7b, 0x6f, 0x34, 0x45,
```

```
0xe7, 0xca, 0x9f, 0x8e, 0x50, 0xcb ]
payload = decode(buffer)
```

Now, we need to convert this list to a string. We can do this with the `bytes()` function:

```
payload = bytes(payload)
```

Finally, we can send this payload to the server:

```
proc = remote('vunrotc.cole-ellis.com', 11200)
proc.sendline(payload)
proc.interactive()
```

This gives us the flag:

```
$ python3 exploit.py
[+] Opening connection to vunrotc.cole-ellis.com on port 11200: Done
[*] Switching to interactive mode
Enter your password
You entered: \xev j\x9e\xa6}-
@\x8eN\x04y\x86\xf6\xb0\xde\x14\xfd5}\xc3\xd0zA
\x15&2yh\x8a\x97\xc2\xd3\x1d\x96
flag{ghidra_is_awesome}You win! Here you go:
[*] Got EOF while reading in interactive
$
[*] Interrupted
[*] Closed connection to vunrotc.cole-ellis.com port 11200
```