

逆向第二周wp

逆向题目wp

张程思

一,pyz3

1,根据题目名称可知这是一道关于py的逆向题目,之前在ctfbase做过几乎一模一样的

 **Ezpy** 250 pts

出题: A1gorithms
难度: 中等

既是加密也是解密

💡 如果你使用 `pyinstxtractor`, 请在 `pyinstxtractor.py` 找到 `# Skip PYZ extraction if not running under the same python version` 然后将它下面五行的 `return` 注释掉 (让 `return` 不生效)。否则, 你运行 `pyinstxtractor.py` 时使用的 Python 小版本号必须与附件相同。或者, 你也可以尝试使用更方便的工具, 例如 `pydumpck`。

下载附件 [外部链接](#)

该题目已被解出 提交 flag

2,先按这个流程解包 PyInstaller 的文件;

名称	修改日期	类型	大小
PYZ.pyz_extracted	2025/10/28 22:41	文件夹	
_bz2.pyd	2025/10/28 22:41	Python Extension ...	83 KB
_decimal.pyd	2025/10/28 22:41	Python Extension ...	251 KB
_hashlib.pyd	2025/10/28 22:41	Python Extension ...	65 KB
_lzma.pyd	2025/10/28 22:41	Python Extension ...	155 KB
_socket.pyd	2025/10/28 22:41	Python Extension ...	82 KB
base_library.zip	2025/10/28 22:41	压缩(zipped)文件夹	1,302 KB
libcrypto-3.dll	2025/10/28 22:41	应用程序扩展	5,110 KB
pyi_rth_inspect.pyc	2025/10/28 22:41	Compiled Python Fi...	3 KB
pyiboot01_bootstrap.pyc	2025/10/28 22:41	Compiled Python Fi...	2 KB
pyimod01_archive.pyc	2025/10/28 22:41	Compiled Python Fi...	5 KB
pyimod02_importers.pyc	2025/10/28 22:41	Compiled Python Fi...	32 KB
pyimod03_ctypes.pyc	2025/10/28 22:41	Compiled Python Fi...	7 KB
pyimod04_pywin32.pyc	2025/10/28 22:41	Compiled Python Fi...	2 KB
python312.dll	2025/10/28 22:41	应用程序扩展	6,755 KB
PYZ.pyz	2025/10/28 22:41	Python Zip Applica...	1,245 KB
select.pyd	2025/10/28 22:41	Python Extension ...	31 KB
struct.pyc	2025/10/28 22:41	Compiled Python Fi...	1 KB
task.pyc	2025/10/28 22:41	Compiled Python Fi...	90 KB
unicodedata.pyd	2025/10/28 22:41	Python Extension ...	1,112 KB
VCRUNTIME140.dll	2025/10/28 22:41	应用程序扩展	118 KB

3.使用pycdc反编译其中的task.pyc文件提取py代码

```
# Source Generated with Decompyle++
# File: task.pyc (Python 3.12)
```

```
def check(flag):
    if 47 * flag[0] + 41 * flag[1] + 32 * flag[2] + 56 * flag[3] + 52 * flag[4] + 67 * flag[5] + 13 * flag[6] + 25 * flag[7] + 20 * flag[8] + 98 * flag[9] + 88 * flag[10] + 65 * flag[11] + 82 *
    flag[12] + 92 * flag[13] + 3 * flag[14] + 29 * flag[15] + 93 * flag[16] + 88 * flag[17] + 45 * flag[18] + 58 * flag[19] + 40 * flag[20] + 72 * flag[21] + 99 * flag[22] + 10 * flag[23] +
    94 * flag[24] + 62 * flag[25] + 82 * flag[26] + 92 * flag[27] + 23 * flag[28] + 46 * flag[29] + 55 * flag[30] + 72 * flag[31] + 44 * flag[32] + 9 * flag[33] + 65 * flag[34] + 42 *
    flag[35] == 176386 and 10 * flag[0] + 98 * flag[1] + 5 * flag[2] + 28 * flag[3] + 68 * flag[4] + 20 * flag[5] + 2 * flag[6] + 22 * flag[7] + 65 * flag[8] + 44 * flag[9] + 85 * flag[10] +
    97 * flag[11] + 33 * flag[12] + 74 * flag[13] + 93 * flag[14] + 74 * flag[15] + 41 * flag[16] + 65 * flag[17] + 32 * flag[18] + 93 * flag[19] + 22 * flag[20] + 69 * flag[21] + 68 *
    flag[22] + 57 * flag[23] + 47 * flag[24] + 29 * flag[25] + 74 * flag[26] + 54 * flag[27] + 91 * flag[28] + 90 * flag[29] + 26 * flag[30] + 11 * flag[31] + 89 * flag[32] + 57 * flag[33] +
    100 * flag[34] + 95 * flag[35] == 186050 and 25 * flag[0] + 22 * flag[1] + 5 * flag[2] + 5 * flag[3] + 8 * flag[4] + 3 * flag[5] + 12 * flag[6] + 70 * flag[7] + 25 * flag[8] + 61 *
    flag[9] + 68 * flag[10] + 12 * flag[11] + 27 * flag[12] + 42 * flag[13] + 83 * flag[14] + 91 * flag[15] + 67 * flag[16] + 46 * flag[17] + 8 * flag[18] + 45 * flag[19] + 94 * flag[20] +
    80 * flag[21] + 69 * flag[22] + 95 * flag[23] + 12 * flag[24] + 21 * flag[25] + 94 * flag[26] + 82 * flag[27] + 93 * flag[28] + 41 * flag[29] + 4 * flag[30] + 56 * flag[31] + 92 *
    flag[32] + 77 * flag[33] + 15 * flag[34] + 30 * flag[35] == 154690 and 33 * flag[0] + 49 * flag[1] + 56 * flag[2] + 40 * flag[3] + 90 * flag[4] + 59 * flag[5] + 82 * flag[6] + 6 *
    flag[7] + 81 * flag[8] + 32 * flag[9] + 23 * flag[10] + 76 * flag[11] + 93 * flag[12] + 83 * flag[13] + 10 * flag[14] + 44 * flag[15] + 58 * flag[16] + 33 * flag[17] + 79 * flag[18] + 77 *
    flag[19] + 82 * flag[20] + 56 * flag[21] + 70 * flag[22] + 34 * flag[23] + 45 * flag[24] + 76 * flag[25] + 57 * flag[26] + 43 * flag[27] + 100 * flag[28] + 19 * flag[29] + 11 *
    flag[30] + 90 * flag[31] + 3 * flag[32] + 60 * flag[33] + 57 * flag[34] + 23 * flag[35] == 172116 and 65 * flag[0] + 70 * flag[1] + 20 * flag[2] + 32 * flag[3] + 75 * flag[4] + 30 *
    flag[5] + 3 * flag[6] + 78 * flag[7] + 35 * flag[8] + 45 * flag[9] + 95 * flag[10] + 93 * flag[11] + 52 * flag[12] + 32 * flag[13] + 88 * flag[14] + 94 * flag[15] + 67 * flag[16] + 34 *
    flag[17] + 91 * flag[18] + 88 * flag[19] + 31 * flag[20] + 61 * flag[21] + 17 * flag[22] + 99 * flag[23] + 100 * flag[24] + 49 * flag[25] + 4 * flag[26] + 60 * flag[27] + 81 * flag[28] +
    88 * flag[29] + 43 * flag[30] + 34 * flag[31] + 30 * flag[32] + 52 * flag[33] + 18 * flag[34] + 100 * flag[35] == 190544 and 81 * flag[0] + 42 * flag[1] + 28 * flag[2] + 98 * flag[3] +
    31 * flag[4] + 46 * flag[5] + 64 * flag[6] + 15 * flag[7] + 49 * flag[8] + 13 * flag[9] + 100 * flag[10] + 81 * flag[11] + 32 * flag[12] + 52 * flag[13] + 59 * flag[14] + 24 * flag[15] +
    94 * flag[16] + 32 * flag[17] + 93 * flag[18] + 32 * flag[19] + 13 * flag[20] + 89 * flag[21] + 37 * flag[22] + 30 * flag[23] + 78 * flag[24] + 81 * flag[25] + 9 * flag[26] + 45 *
    flag[27] + 93 * flag[28] + 100 * flag[29] + 97 * flag[30] + 10 * flag[31] + 80 * flag[32] + 54 * flag[33] + 88 * flag[34] + 85 * flag[35] == 190323 and 76 * flag[0] + 54 * flag[1] + 5 *
    flag[2] + 14 * flag[3] + 62 * flag[4] + 44 * flag[5] + 24 * flag[6] + 29 * flag[7] + 85 * flag[8] + 87 * flag[9] + 19 * flag[10] + 3 * flag[11] + 65 * flag[12] + 24 * flag[13] + 92 *
    flag[14] + 37 * flag[15] + 57 * flag[16] + 20 * flag[17] + 45 * flag[18] + 5 * flag[19] + 13 * flag[20] + 91 * flag[21] + 92 * flag[22] + 75 * flag[23] + 36 * flag[24] + 79 * flag[25] +
    12 * flag[26] + 22 * flag[27] + 75 * flag[28] + 82 * flag[29] + 28 * flag[30] + 82 * flag[31] + 24 * flag[32] + 53 * flag[33] + 56 * flag[34] + 92 * flag[35] == 162017 and 53 *
    flag[0] + 52 * flag[1] + 72 * flag[2] + 23 * flag[3] + 26 * flag[4] + 13 * flag[5] + 62 * flag[6] + 96 * flag[7] + 67 * flag[8] + 96 * flag[9] + 66 * flag[10] + 41 * flag[11] + 5 * flag[12]
    + 18 * flag[13] + 37 * flag[14] + 13 * flag[15] + 61 * flag[16] + 71 * flag[17] + 91 * flag[18] + 96 * flag[19] + 56 * flag[20] + 3 * flag[21] + 65 * flag[22] + 14 * flag[23] + 57 *
    flag[24] + 69 * flag[25] + 75 * flag[26] + 68 * flag[27] + 10 * flag[28] + 60 * flag[29] + 62 * flag[30] + 95 * flag[31] + 53 * flag[32] + 19 * flag[33] + 7 * flag[34] + 56 * flag[35]
    == 165118 and 26 * flag[0] + 7 * flag[1] + 49 * flag[2] + 14 * flag[3] + 36 * flag[4] + 87 * flag[5] + 21 * flag[6] + 35 * flag[7] + 15 * flag[8] + 91 * flag[9] + 15 * flag[10] + 100 *
```

4.分析后发现这段代码的原理是通过一个36x36的线性方程组来验证输入的Flag,根据题目提示需要下载Z3 约束求解器.于是解题脚本如下

```

import z3

# f 是一个列表，包含36个Z3的整型变量，代表flag的36个字节
f = [z3.Int(f'f_{i}') for i in range(36)]

# 创建一个求解器实例
s = z3.Solver()

# --- 从代码中提取的系数矩阵 A 和结果向量 b ---

# A 是 36x36 的系数矩阵
A = [
    [47, 41, 32, 56, 52, 67, 13, 25, 20, 98, 88, 65, 82, 92, 3, 29, 93, 88,
    45, 58, 40, 72, 99, 10, 94, 62, 82, 92, 23, 46, 55, 72, 44, 9, 65, 42],
    [10, 98, 5, 28, 68, 20, 2, 22, 65, 44, 85, 97, 33, 74, 93, 74, 41, 65, 32,
    93, 22, 69, 68, 57, 47, 29, 74, 54, 91, 90, 26, 11, 89, 57, 100, 95],
    [25, 22, 54, 5, 8, 3, 12, 70, 25, 61, 68, 12, 27, 42, 83, 91, 67, 46, 8,
    45, 94, 80, 69, 95, 12, 21, 94, 82, 93, 41, 4, 56, 92, 77, 15, 30],
    [33, 49, 56, 40, 90, 59, 82, 6, 81, 32, 23, 76, 93, 83, 10, 44, 58, 33,
    79, 77, 82, 56, 70, 34, 45, 76, 57, 43, 100, 19, 11, 90, 3, 60, 57, 23],
    [65, 70, 20, 32, 75, 30, 3, 78, 35, 45, 95, 93, 52, 32, 88, 94, 67, 34,
    91, 88, 31, 61, 17, 99, 100, 49, 4, 60, 81, 88, 43, 34, 30, 52, 18, 100],
    [81, 42, 28, 98, 31, 46, 64, 15, 49, 13, 100, 81, 32, 52, 59, 24, 94, 32,
    93, 32, 13, 89, 37, 30, 78, 81, 9, 45, 93, 100, 97, 10, 80, 54, 88, 85],
    [76, 54, 5, 14, 62, 44, 24, 29, 85, 87, 19, 3, 65, 24, 92, 37, 57, 20, 45,
    5, 13, 91, 92, 75, 36, 79, 12, 22, 75, 82, 28, 82, 24, 53, 56, 92],
    [53, 52, 72, 23, 26, 13, 62, 96, 67, 96, 66, 41, 5, 18, 37, 13, 61, 71,
    91, 96, 56, 3, 65, 14, 57, 69, 75, 68, 10, 60, 62, 95, 53, 19, 7, 56],
    [26, 7, 49, 14, 36, 87, 21, 35, 15, 91, 15, 100, 8, 32, 100, 35, 66, 3,
    79, 96, 82, 95, 68, 13, 86, 51, 24, 76, 30, 60, 29, 70, 40, 90, 44, 3],
    [47, 19, 37, 93, 73, 30, 45, 47, 72, 85, 37, 68, 89, 34, 4, 50, 87, 33,
    87, 43, 9, 61, 93, 49, 74, 49, 68, 29, 54, 54, 37, 79, 33, 65, 59, 15],
    [79, 73, 60, 62, 25, 16, 77, 81, 79, 31, 82, 84, 62, 36, 18, 20, 46, 57,
    21, 40, 3, 50, 58, 80, 84, 71, 87, 3, 13, 77, 83, 39, 55, 34, 41, 63],
    [7, 50, 26, 79, 21, 42, 83, 94, 63, 83, 3, 68, 25, 91, 3, 5, 17, 61, 3,
    40, 87, 11, 27, 74, 73, 21, 56, 46, 36, 24, 14, 63, 21, 71, 30, 53],
    [57, 51, 49, 15, 94, 34, 27, 5, 100, 68, 67, 81, 10, 5, 85, 70, 80, 20,
    89, 30, 84, 35, 41, 87, 75, 67, 20, 33, 29, 6, 97, 25, 10, 18, 23, 30],
    [97, 93, 10, 44, 28, 22, 17, 41, 47, 62, 42, 47, 61, 32, 31, 52, 47, 92,
    42, 37, 7, 40, 48, 40, 11, 96, 51, 42, 66, 8, 89, 64, 30, 11, 8, 83],
    [51, 94, 58, 76, 21, 10, 75, 4, 55, 37, 71, 97, 27, 93, 82, 94, 38, 69,
    36, 58, 93, 18, 54, 59, 12, 12, 54, 83, 73, 83, 33, 12, 78, 38, 45, 57],
    [78, 29, 8, 47, 48, 88, 18, 88, 50, 58, 36, 88, 9, 74, 85, 5, 91, 58, 85,
    46, 89, 76, 61, 6, 61, 78, 4, 48, 50, 69, 23, 70, 23, 15, 22, 68],
    [75, 2, 94, 97, 72, 62, 78, 42, 69, 11, 37, 3, 29, 15, 39, 33, 18, 33, 12,
    64, 6, 18, 34, 15, 3, 100, 85, 32, 97, 93, 84, 73, 26, 31, 71, 97],

```

```

[59, 26, 48, 86, 58, 70, 61, 100, 63, 74, 26, 38, 24, 45, 52, 32, 91, 89,
19, 59, 87, 5, 15, 68, 72, 67, 2, 65, 46, 10, 33, 79, 11, 16, 73, 53],
[6, 66, 59, 76, 86, 20, 59, 34, 28, 48, 86, 5, 87, 13, 95, 87, 65, 35, 58,
10, 98, 100, 4, 78, 66, 57, 34, 86, 62, 36, 92, 28, 3, 24, 49, 28],
[25, 48, 44, 16, 99, 100, 69, 26, 65, 32, 18, 65, 58, 72, 61, 56, 10, 78,
93, 98, 39, 43, 87, 12, 42, 100, 100, 47, 31, 51, 75, 10, 63, 48, 22, 87],
[61, 13, 100, 59, 31, 9, 28, 7, 27, 63, 11, 57, 95, 79, 21, 30, 60, 81,
43, 32, 30, 34, 80, 53, 28, 39, 74, 21, 18, 92, 73, 60, 21, 69, 76, 84],
[22, 62, 61, 20, 66, 2, 11, 82, 93, 13, 69, 37, 92, 80, 66, 47, 28, 14,
62, 56, 89, 29, 39, 38, 46, 10, 6, 82, 77, 78, 45, 50, 5, 73, 17, 65],
[5, 84, 83, 77, 76, 60, 20, 48, 53, 14, 98, 50, 37, 15, 31, 69, 55, 37,
64, 35, 26, 20, 18, 67, 50, 57, 60, 71, 4, 35, 23, 52, 11, 15, 83, 51],
[33, 47, 89, 52, 89, 55, 98, 28, 48, 90, 69, 29, 68, 24, 19, 18, 44, 27,
14, 64, 15, 31, 23, 2, 36, 45, 37, 71, 61, 92, 28, 64, 13, 66, 98, 3],
[80, 88, 68, 66, 46, 75, 32, 19, 36, 83, 63, 86, 79, 30, 61, 50, 100, 52,
66, 30, 20, 97, 45, 46, 38, 21, 32, 79, 68, 43, 65, 47, 86, 30, 74, 18],
[11, 58, 95, 67, 96, 74, 60, 11, 21, 14, 100, 60, 70, 92, 92, 39, 43, 52,
5, 22, 90, 70, 12, 52, 36, 21, 45, 59, 74, 46, 11, 60, 8, 52, 14, 77],
[57, 37, 94, 43, 53, 55, 7, 83, 91, 61, 86, 6, 44, 87, 61, 92, 24, 74,
100, 22, 12, 68, 19, 88, 81, 83, 70, 39, 30, 82, 30, 35, 55, 18, 27, 80],
[80, 14, 5, 89, 71, 82, 44, 8, 33, 26, 77, 49, 36, 90, 73, 71, 66, 4, 37,
78, 38, 18, 15, 79, 6, 74, 18, 85, 56, 53, 90, 75, 52, 2, 13, 54],
[96, 29, 37, 70, 92, 80, 24, 36, 32, 29, 78, 45, 58, 55, 16, 92, 71, 82,
86, 23, 4, 58, 16, 18, 38, 53, 82, 76, 83, 73, 87, 36, 61, 85, 61, 69],
[14, 71, 53, 46, 59, 53, 22, 69, 67, 43, 23, 14, 77, 95, 19, 83, 79, 41,
12, 53, 3, 4, 65, 92, 64, 52, 3, 59, 89, 75, 12, 46, 61, 53, 97, 43],
[57, 99, 49, 100, 68, 99, 26, 65, 47, 65, 90, 68, 84, 4, 9, 43, 88, 33,
48, 88, 37, 31, 21, 94, 22, 93, 70, 14, 13, 28, 83, 12, 80, 58, 43, 97],
[33, 94, 56, 48, 13, 44, 81, 42, 19, 96, 67, 79, 12, 67, 34, 72, 45, 48,
24, 71, 65, 13, 32, 97, 48, 42, 65, 95, 54, 9, 35, 57, 18, 20, 83, 76],
[31, 38, 83, 45, 28, 97, 54, 11, 80, 45, 92, 13, 52, 94, 51, 30, 11, 61,
46, 10, 28, 72, 20, 95, 90, 39, 32, 95, 19, 3, 65, 71, 73, 80, 23, 71],
[9, 81, 80, 37, 96, 72, 95, 93, 26, 98, 50, 79, 57, 13, 49, 96, 82, 84,
89, 40, 38, 66, 81, 81, 79, 77, 86, 68, 26, 37, 15, 56, 13, 17, 50, 37],
[82, 57, 33, 32, 79, 25, 54, 27, 50, 14, 72, 31, 28, 66, 4, 6, 48, 34, 63,
51, 12, 21, 73, 66, 53, 38, 54, 59, 76, 63, 61, 30, 84, 80, 98, 46],
[69, 15, 23, 8, 46, 55, 21, 91, 37, 9, 61, 20, 23, 96, 28, 67, 19, 50, 18,
71, 30, 14, 10, 24, 100, 15, 91, 15, 93, 24, 46, 61, 67, 60, 56, 81]
]

```

b 是 36 个等式的结果

b = [

```

176386, 186050, 154690, 172116, 190544, 190323, 162017, 165118,
153332, 168472, 178706, 143852, 154052, 147899, 176754, 171970,
166497, 173887, 173189, 174138, 157623, 154943, 156078, 156158,
181770, 173577, 180922, 158596, 181072, 163777, 187620, 169266,

```

```

    162587, 198705, 160349, 148095
]

# 添加约束条件:
# 1. Flag的字符都是可打印的ASCII字符 (范围 32 到 126)
for i in range(36):
    s.add(f[i] >= 32, f[i] <= 126)

# 2. 添加36个线性方程
for i in range(36):
    # z3.Sum 计算点积 (A[i][0]*f[0] + A[i][1]*f[1] + ...)
    equation = z3.Sum([A[i][j] * f[j] for j in range(36)]) == b[i]
    s.add(equation)

# 检查是否能找到解
if s.check() == z3.sat:
    # 获取解
    m = s.model()
    # 从模型中提取flag的每个字节 (整数)
    res = [m[f[i]].as_long() for i in range(36)]
    # 将整数列表转换回字符串
    flag_str = "".join(map(chr, res))
    print(f"🎉 成功找到Flag: ")
    print(flag_str)
else:
    print("❌ 未能找到解。")

```

5,得出flag

```
flag{PyTH0n_R3v3rs1Ng_4nd_Z3_s0lV3r}
```

二,Dont-debug-me

1,ida查看伪代码,其中有提示,包括题目也是提示,需要我们进行调试代码,先分析一下代码逻辑,如下路径可看出得出flag的方法

```
IDA View-A      Pseudocode-A      Hex View-1
1 int __fastcall main(int argc, const char **argv, const char **envp)
2 {
3     char o000[26]; // [rsp+20h] [rbp-70h] BYREF
4     char o00[6]; // [rsp+3Ah] [rbp-56h] BYREF
5     char o0[44]; // [rsp+40h] [rbp-50h] BYREF
6     int a; // [rsp+6Ch] [rbp-24h] BYREF
7     char O[32]; // [rsp+70h] [rbp-20h] BYREF
8
9     _main(argc, argv, envp);
10    strcpy(O, "plz_dont_debugme");
11    if ( IsDebugged() )
12    {
13        puts("GoOoD boy !!!");
14        puts("typing 1 give you flag:");
15        a = 0;
16        scanf("%d", &a);
17        if ( a == 1 )
18        {
19            strcpy(o00, "BaseCTF{Th1s_1s_a_fak3_flag}");
20            puts(o00);
21            return -1;
22        }
23        else if ( IsDebuggerPresentFlag() )
24        {
25            strcpy(o00, "laodi");
26            puts(o00);
27            return -1;
28        }
29        else
30        {
31            exit(0);
32            return 0;
33        }
34    }
35    else
36    {
37        strcpy(o000, "What are you doing !!!");
38        puts(o000);
39        return -1;
40    }
41 }
```

IDA View-A

Pseudocode-A

```
1 void __cdecl ex1t(char *0)
2 {
3     uint8_t mg[64]; // [rsp+20h] [rbp-70h] BYREF
4     uint8_t key[32]; // [rsp+60h] [rbp-30h] BYREF
5     uint64_t n; // [rsp+88h] [rbp-8h]
6
7     strcpy((char *)key, "plz_dont_debugme");
8     key[17] = 0;
9     *(_WORD *)&key[18] = 0;
10    *(_DWORD *)&key[20] = 0;
11    *(_QWORD *)&key[24] = 0;
12    n = 128;
13    strcpy((char *)mg, "BaseCTF{Y0u_r3a11y_kn0w_debugg!!!}");
14    memset(&mg[35], 0, 29);
15    s20(mg, 0x40u, key, 0x80u);
16    o00h(mg);
17 }
```

IDA View-A

Pse

```
1 void __cdecl o00h(uint8_t *mg)
2 {
3     int i; // [rsp+2Ch] [rbp-4h]
4
5     printf("BaseCTF{");
6     for ( i = 0; i <= 31; ++i )
7         printf("%x", mg[i]);
8     putchar(125);
9     putchar(10);
10 }
```

2,知道了如何得到flag的途径后我们就分析一下asm页面,找到哪个分支是正确的绕过路径

```
mov     [rbp+0+10h], 0
call    _Z10IsDebuggedv ; IsDebugged(void)
test    eax, eax
setnz   al, al
test    al, al
jz       loc_402046

lea     rcx, Buffer ; "GoOoD boy !!!"
call    puts
lea     rcx, aTyping1GiveYou ; "typing 1 give you flag:"
call    puts
mov     [rbp+a], 0
lea     rax, [rbp+a]
mov     rdx, rax
lea     rcx, aD ; "%d"
call    scanf
mov     eax, [rbp+a]
cmp     eax, 1
jnz     short loc_402005

loc_402005:
call    _Z21IsDebuggerPresentFlagv ; IsDebuggerPresentFlag(void)
test    eax, eax
setnz   al, al
test    al, al
jz       short loc_402033

mov     rax, 7B46544365736142h
qword ptr [rbp+0], rax
mov     rax, 5F73315F73316854h
qword ptr [rbp+0+8], rax
mov     rax, 665F336B61665F61h
qword ptr [rbp+0+10h], rax
dword ptr [rbp+0+18h], 7D676131h
[rbp+0+1Ch], 0
rax, [rbp+0]
rcx, rax ; Buffer
puts
eax, 0FFFFFFFh
jmp     short loc_402084

loc_402033:
lea     rax, [rbp+0]
mov     rcx, rax ; 0
call    _Z4exitPc ; exit(char *)
mov     eax, 0
jmp     short loc_402084

loc_402046:
mov     rax, 6572612074616857h
mov     qword ptr [rbp+000], rax
mov     rax, 696F6420756F7920h
mov     qword ptr [rbp+000+8], rax
mov     dword ptr [rbp+000+10h], 2120676Eh
mov     word ptr [rbp+000+14h], 2121h
[rbp+000+16h], 0
lea     rax, [rbp+000]
mov     rcx, rax ; Buffer
call    puts
```

2024, 10:12:12] [MSC V.1938 64 Bit (AMD64)]
eam <idapython@googlegroups.com>

3,第一个判断语句在jz这一行,于是f2设置断点,并使用本地调试器进行调试,这里我自己调了好多次,最后得出的正确绕过路径是

```
mov     [rbp+0+10h], 0
call    _Z10IsDebuggedv ; IsDebugged(void)
test    eax, eax
setnz   al, al
test    al, al
jz       loc_402046

lea     rcx, Buffer ; "GoOoD boy !!!"
call    puts
lea     rcx, aTyping1GiveYou ; "typing 1 give you flag:"
call    puts
mov     [rbp+a], 0
lea     rax, [rbp+a]
mov     rdx, rax
lea     rcx, aD ; "%d"
call    scanf
mov     eax, [rbp+a]
cmp     eax, 1
jnz     short loc_402005

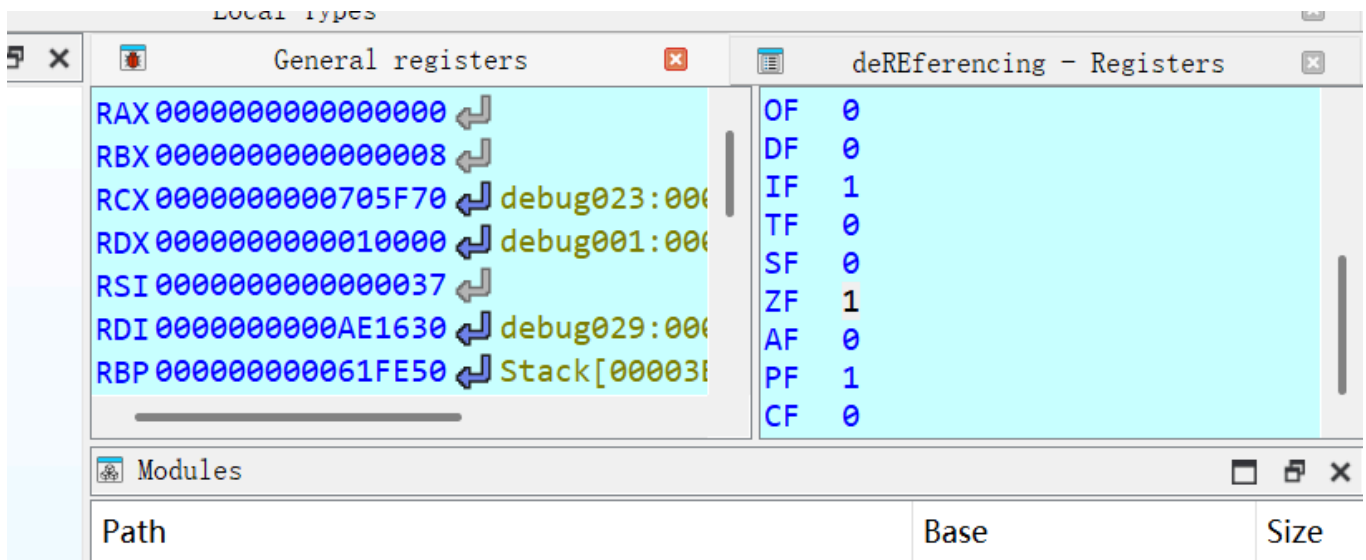
loc_402005:
call    _Z21IsDebuggerPresentFlagv ; IsDebuggerPresentFlag(void)
test    eax, eax
setnz   al, al
test    al, al
jz       short loc_402033

mov     rax, 7B46544365736142h
qword ptr [rbp+0], rax
mov     rax, 5F73315F73316854h
qword ptr [rbp+0+8], rax
mov     rax, 665F336B61665F61h
qword ptr [rbp+0+10h], rax
dword ptr [rbp+0+18h], 7D676131h
[rbp+0+1Ch], 0
lea     rax, [rbp+0]
mov     rcx, rax ; Buffer
call    puts
eax, 0FFFFFFFh
jmp     short loc_402084

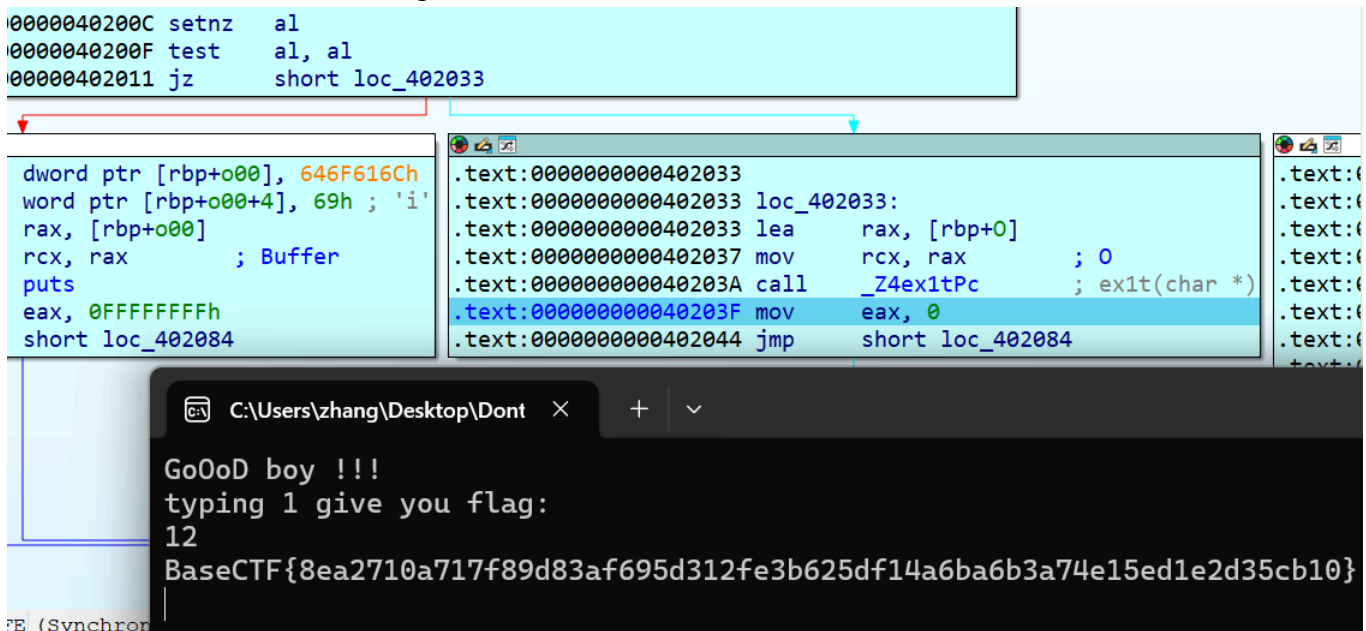
loc_402033:
lea     rax, [rbp+0]
mov     rcx, rax ; 0
call    _Z4exitPc ; exit(char *)
mov     eax, 0
jmp     short loc_402084

loc_402046:
mov     rax, 6572612074616857h
mov     qword ptr [rbp+000], rax
mov     rax, 696F6420756F7920h
mov     qword ptr [rbp+000+8], rax
mov     dword ptr [rbp+000+10h], 2120676Eh
mov     word ptr [rbp+000+14h], 2121h
[rbp+000+16h], 0
lea     rax, [rbp+000]
mov     rcx, rax ; Buffer
call    puts
```

4,调试后进入找到这个页面,由于jz 是“Zero Flag is Set”的缩写,意思是只有当 ZF=1时,才执行跳转,所以我们将1改为0,使其分支跳转到有scanf的这边,



5,f8调试到这里就可以得出flag啦



三, ezAndroid

1,apk附件可得是安卓逆向,放到jadx里看一下,查找mainactive函数,大概是base64然后又调用了一些函数,按照之前做安卓逆向的经验,我直接把apk里的so后缀文件单独解压出来放ida,发现就是简单的异或

ezAndroid.apk - Bandizip (Professional)

文件(E) 编辑(E) 查找(I) 选项(O) 视图(V) 工具(I) 帮助(H)

× 您想用Bandizip打开ZIP文件吗?

打开

解压

新建

添加

删除

测试

扫描

查看

代码页

ezAndroid.apk

- assets
 - dexopt
- kotlin
- lib
 - arm64-v8a
 - armeabi-v7a
 - x86
 - x86_64
- META-INF
- res

名称	压缩后大小	原始大小	类型
..			
libhello.so	196,052	196,052	SO 文件

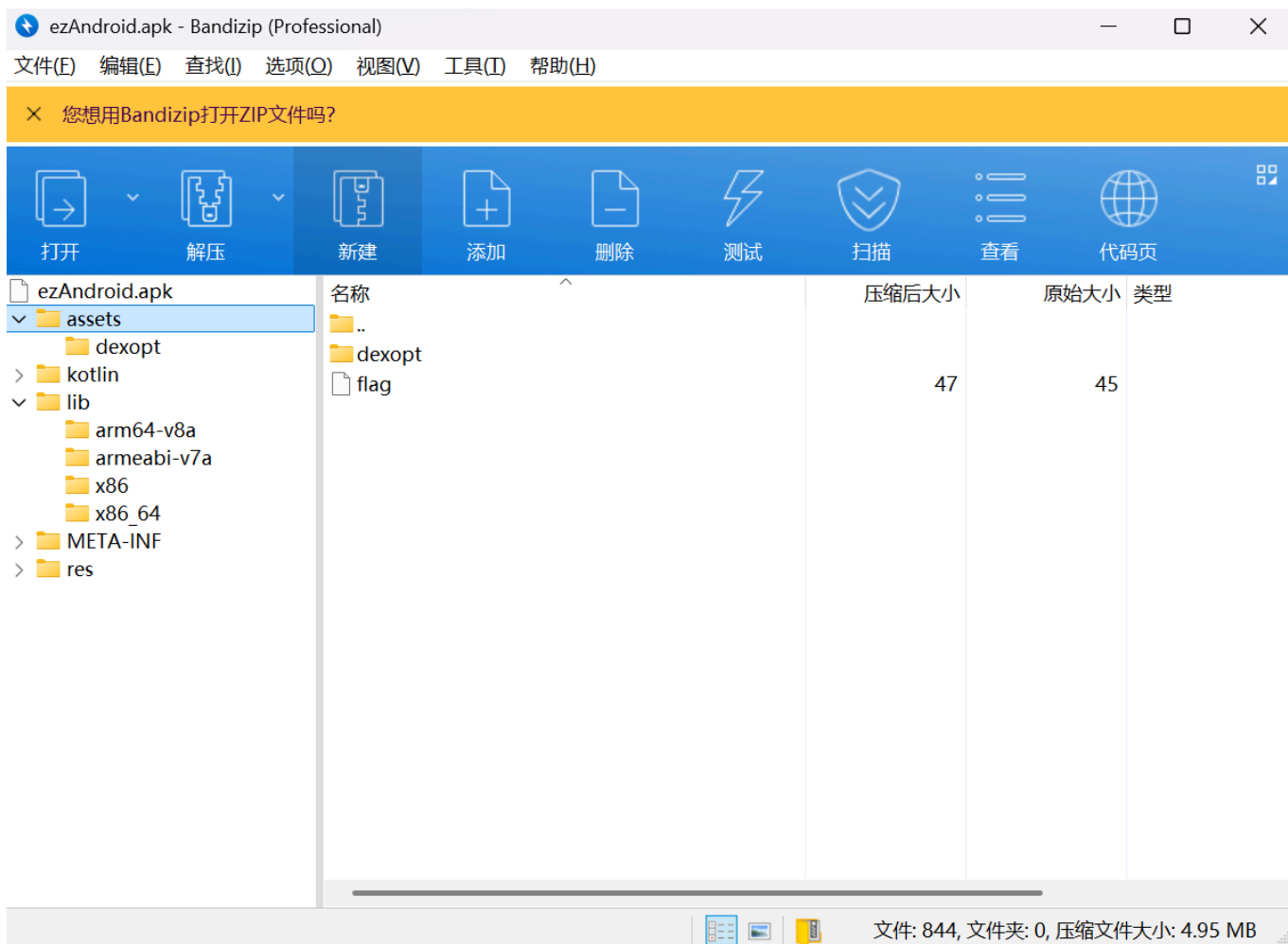
文件: 844, 文件夹: 0, 压缩文件大小: 4.95 MB

```
int __cdecl Java_com_example_hello_MainActivity_Base64encode(int a1, int a2, int a3)
{
    int v3; // ebp
    int v4; // edi
    _BYTE *ptr; // esi
    unsigned int v6; // ecx
    char *v7; // edx
    char v8; // al
    int v9; // edi
    signed int size; // [esp+4h] [ebp-18h]

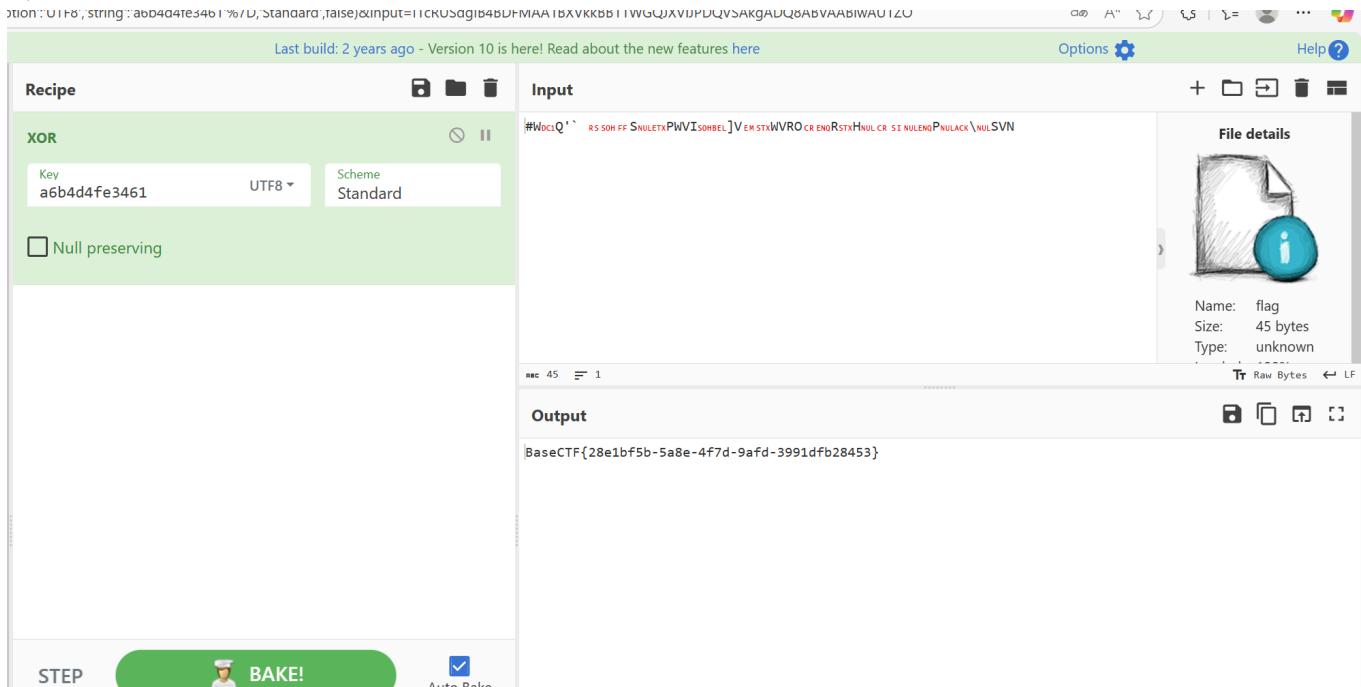
    v3 = a1;
    size = (*(int (__cdecl **)(int, int))(*(_DWORD *)a1 + 684))(a1, a3);
    v4 = (*(int (__cdecl **)(int, int, _DWORD))(*(_DWORD *)v3 + 736))(v3, a3, 0);
    ptr = (_BYTE *)operator new[](size);
    if ( size > 0 )
    {
        v6 = 0;
        if ( size != 1 )
        {
            v6 = 0;
            do
            {
                v7 = (char *)&off_30710 - 3 * ((v6 / 3) & 0FFFFFFC) - 21428;
                v8 = *(_BYTE *)(v4 + v6) ^ v7[v6];
                LOBYTE(v7) = v7[v6 + 1];
                ptr[v6] = v8;
                ptr[v6 + 1] = *(_BYTE *)(v4 + v6 + 1) ^ (unsigned __int8)v7;
                v6 += 2;
            } while ( (size & 0FFFFFFE) != v6 );
        }
        v3 = a1;
        if ( (size & 1) != 0 )
        ptr[v6] = *(_BYTE *)(v4 + v6) ^ byte_6970[v6 - 3 * ((v6 / 3) & 0FFFFFFC)];
    }
    *(void (__cdecl **)(int, int, int, int))(*(_DWORD *)v3 + 768)(v3, a3, v4, 2);
    v9 = (*(int (__cdecl **)(int, signed int))(*(_DWORD *)v3 + 704))(v3, size);
    *(void (__cdecl **)(int, int, _DWORD, signed int, _BYTE **))(*(_DWORD *)v3 + 832)(v3, v9, 0, size, ptr);
    operator delete[](ptr);
    return v9;
}
```

00011038 Java_com_example_hello_MainActivity_Base64encode:35 (11038)

2,还缺少密文,ida中找不到,于是查看之前的apk文件,发现flag文件还在其中

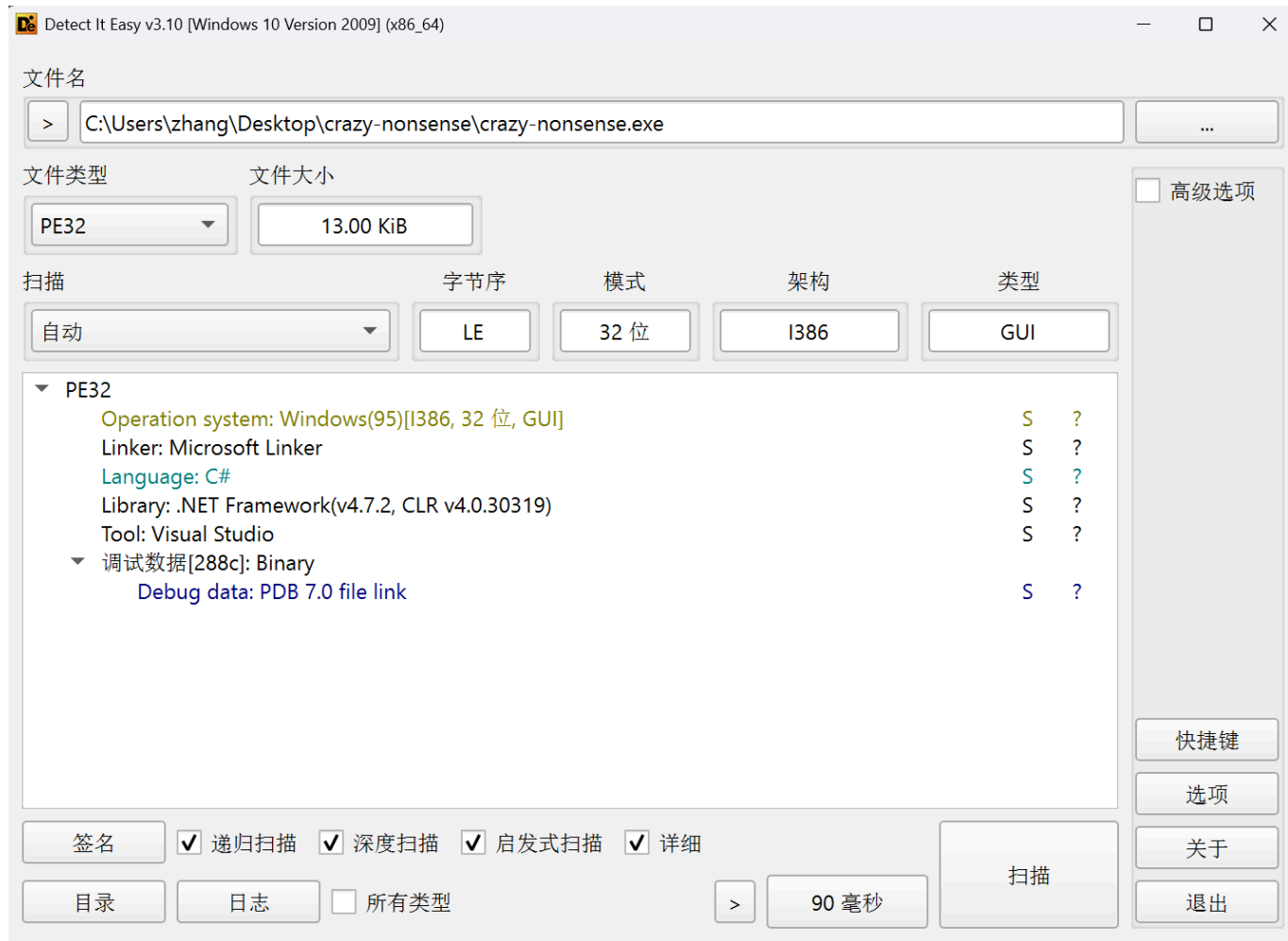


3,直接赛博厨子梭哈



四,出题人已疯

1,放die里发现是net类型,需要用dnspy打开,



2,于是查看dnspy,找到main函数

```
MainWindow() : void
1  // crazy_nonsense.MainWindow
2  // Token: 0x04000001 RID: 1
3  public string[] sentences = new string[]
4  {
5      "你以为我还会在乎吗? \ud83d\ude2c\ud83d\ude2c\ud83d\ude2c我在昆仑山练了六年的剑\ud83d\ude1f\ud83d\ude1f\ud83d\ude1f我的心早就和昆仑山的
6      雪一样冷了\ud83d\ude10\ud83d\ude10\ud83d\ude10我在大润发杀了十年的鱼\ud83d\ude2b\ud83d\ude2b\ud83d\ude2b我以为我的心早已跟我的刀一样冷
7      了\ud83d\ude29\ud83d\ude29\ud83d\ude29",
8      "我早上坐公交滴卡的时候和司机大叔说“两个人”，司机惊讶地看着我“你明明就是一个人，为什么要滴两个人的卡？”我回他，“我心中还有一个叫
9      Kengwang的。”司机回我说，“天使是不用收钱的。””，
10     "(尖叫) (扭曲) (阴暗的爬行) (扭动) (阴暗地蠕动) (翻滚) (激烈地爬动) (痉挛) (嘶吼) (蠕动) (阴森的低吼) (爬行) (分裂) (走上
11     岸) (扭曲的行走) (不分对象攻击)",
12     "地球没我照样转? 硬撑罢了! 地球没我照样转? 硬撑罢了! 地球没我照样转? 硬撑罢了! 地球没我照样转? 硬撑罢了! 地球
13     没我照样转? 硬撑罢了!",
14     "扭曲上勾拳! 阴暗的下勾拳! 尖叫左勾拳! 右勾拳爬行! 扭动扫堂腿! 分裂回旋踢! 这是蜘蛛阴暗的吃耳屎，这是龙卷风翻滚停车场! 乌鸦痉挛! 老鼠嘶
15     吼! 大象蠕动! 愤怒的章鱼! 无差别攻击! 无差别攻击! 无差别攻击!"
16 };
17 // Token: 0x04000002 RID: 2
18 public byte[] title = new byte[]
19 {
20     250,
21     81,
22     152,
23     152,
24     186,
25     78,
26     242,
27     93,
28     175,
29     117,
30     27,
31     103,
32     104,
33     84,
34     229,
35     119
36 };
37 // Token: 0x06000004 RID: 4 RVA: 0x00002094 File Offset: 0x00000294
38 public MainWindow()
39 {
40     //
41 }
```

3,脚本如下

```
# -*- coding: utf-8 -*-
"""
Reverse-engineer the encoded title from crazy_nonsense.MainWindow
Recovered flag: BaseCTF{y0u_Kn0w_UTF16_6uT_U_r_n0t_Cr@zym@n}
"""

import itertools

# ===== 1. 原始数据 =====

# 题目给出的 byte[] title
title_bytes = [
    250, 81, 152, 152, 186, 78, 242, 93,
    175, 117, 27, 103, 104, 84, 229, 119
]

# 这些字符串在程序里参与了异或
sentences = [
    "你以为我还会在乎吗? 🤪🤪🤪我在昆仑山练了六年的剑🤪🤪🤪我的心早就和昆仑山的雪一样
    冷了🤪🤪🤪我在大润发杀了十年的鱼🤪🤪🤪我以为我的心早已跟我的刀一样冷了🤪🤪🤪",
    "我早上坐公交滴卡的时候和司机大叔说“两个人”，司机惊讶地看着我“你明明就是一个人，为什么
    要滴两个人的卡？”我回他，“我心中还有一个叫Kengwang的。”司机回我说，“天使是不用收钱的。””，
```

```
"（尖叫）（扭曲）（阴暗的爬行）（扭动）（阴暗地蠕动）（翻滚）（激烈地爬动）（痉挛）（嘶
吼）（蠕动）（阴森的低吼）（爬行）（分裂）（走上岸）（扭曲的行走）（不分对象攻击）",
"地球没我照样转？硬撑罢了！" * 6,
"扭曲上勾拳！阴暗的下勾拳！尖叫左勾拳！右勾拳爬行！扭动扫堂腿！分裂回旋踢！这是蜘蛛阴暗
的吃耳屎，这是龙卷风翻滚停车场！乌鸦痉挛！老鼠嘶吼！大象蠕动！愤怒的章鱼！无差别攻击！无差别
攻击！无差别攻击！"
]
```

```
# 拼接所有文本
```

```
source_text = "".join(sentences)
```

```
source_bytes = list(source_text.encode("utf-16-le")) # Unicode 转 UTF-16 LE 字
节序
```

```
# ===== 2. C# 加密逻辑模拟 =====
```

```
def encrypt(input_text):
```

```
    """模拟原程序中的 Encoding.Unicode + mod + 异或 流程"""
```

```
    output = []
```

```
    for i, ch in enumerate(input_text):
```

```
        v = (ord(ch) * ord(ch)) % 0x10000
```

```
        # 与 source 的第 i 个字节异或（原始 C# 程序是 UTF-16 编码，因此
```

```
source_bytes[i])
```

```
        v ^= source_bytes[i]
```

```
        output.append(v)
```

```
    return output
```

```
# ===== 3. 穷举反推解密 =====
```

```
def decrypt(target):
```

```
    result_chars = []
```

```
    for i, val in enumerate(target):
```

```
        src = source_bytes[i]
```

```
        found = None
```

```
        # 只考虑常规可见 ASCII 范围
```

```
        for c in range(32, 127):
```

```
            if ((c * c) % 0x10000) ^ src == val:
```

```
                found = chr(c)
```

```
                break
```

```
        if found is None:
```

```
            raise ValueError(f"无法解出第 {i} 位")
```

```
        result_chars.append(found)
```

```
    return "".join(result_chars)
```

```
# ===== 4. 主流程 =====

if __name__ == "__main__":
    flag = decrypt(title_bytes)
    print("Recovered flag:", flag)

    # 验证: 重新加密后应与原数据一致
    assert encrypt(flag) == title_bytes
    print("Verification passed ✅")
```

五,ezbase

1,ida分析后发现找不到main函数,于是本地调试跑一下,找到了关键的逻辑

```

20  strcpy(
21      ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789_
22      "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/");
23  v6 = 0;
24  memset(buf_, 0, 0x2Eu);
25  n45 = 0;
26  v9[0] = 1;
27  v9[1] = 2;
28  v9[2] = 3;
29  v9[3] = 4;
30  memset(&v9[4], 0, sizeof(_DWORD));
31  sub_1400111AE("Please input your flag:");
32  sub_1400113F2("%s", buf_);
33  while ( buf_[n45] )
34      ++n45;
35  if ( n45 != 45 )
36  {
37      sub_1400111AE("Something was wrong!");
38      exit(0);
39  }
40  sub_140011127( ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789_ );
41  v6 = sub_14001115E((__int64)buf_, (__int64)ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789_, n45, v9);
42  if ( (unsigned int)sub_140011069(v6) )
43      sub_1400111AE("Congratulation,You get the flag success!!!");
44  else
45      sub_1400111AE("Oh,no~~~~~");
46  sub_140011325(v3, &unk_14001B880);
47  return 0;
48 }

```

0000147D sub_140011F60:40 (14001207D)

Hex View-1

000000001400113B0	00 E9 EA 2A 00 00 E9 43 4E 00 00 E9 10 38 00 00		00:0000	00000000
000000001400113C0	E9 48 34 00 00 E9 D6 29 00 00 E9 EF 4E 00 00 E9		01:0008	00000000
000000001400113D0	3C 4E 00 00 E9 D7 23 00 00 E9 62 2A 00 00 E9 77	<N.....	02:0010	00000000
000000001400113E0	4F 00 00 E9 48 00 00 00 E9 49 4F 00 00 E9 FE 0F	0.....	03:0018	00000000

2,发现是base64,但码表应该被换了,于是分析换表的函数,


```
IDA View-RIP Pseudocode-A Pseudocode-B
1 DWORD __fastcall sub_140011CC0(__int64 ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789)
2 {
3     DWORD result; // eax
4     int n16; // [rsp+24h] [rbp+4h]
5
6     result = sub_140011389(byte_140023015);
7     for ( n16 = 0; n16 < 16; ++n16 )
8     {
9         sub_14001119A(
10             4 * n16 + ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789,
11             4 * n16 + 2 + ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789);
12         sub_14001119A(
13             4 * n16 + 1 + ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789,
14             4 * n16 + 3 + ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789);
15         result = n16 + 1;
16     }
17     return result;
18 }
```

3,由此推出完整的魔改表

CDABGHEFKLIJOPMNSTQRWXUVabYZefcdijghmnklqropuvstywx23016745+/89

4,shift+f12找到目标编码字符串

TqK1YUSaQryEMHaLMnWhYU+Fe0WPenqhRXahfkV6WE2fa3iRW197Za62eEaD

5,解题脚本如下

```
def decrypt(ciphertext, modified_table, key):
    # 构建魔改表的索引映射（字符→索引）
    table_index = {char: idx for idx, char in enumerate(modified_table)}
    cipher_len = len(ciphertext)
    plaintext = []

    # 按4个字符一组处理（对应3字节明文）
    for i in range(0, cipher_len, 4):
        # 取当前组的4个字符（确保不越界）
        group = ciphertext[i:i+4]
        if len(group) < 4:
            break # 忽略不完整的组（通常不会出现）

        # 1. 查魔改表索引 → 2. 减去对应key偏移 → 3. 模64得到原始6位组索引
        indices = []
        for j in range(4):
            c = group[j]
            m = table_index[c] # 魔改表中索引
            k = key[j % 4]     # 循环取key
            original_idx = (m - k) % 64 # 原始6位组索引
            indices.append(original_idx)
```

```

# 4个6位组合并为24位整数（3字节）
combined = (indices[0] << 18) | (indices[1] << 12) | (indices[2] << 6)
| indices[3]

# 拆分24位为3个字节（8位/字节）
byte1 = (combined >> 16) & 0xFF
byte2 = (combined >> 8) & 0xFF
byte3 = combined & 0xFF

# 添加到明文（过滤空字节，通常不会有）
plaintext.append(chr(byte1))
plaintext.append(chr(byte2))
plaintext.append(chr(byte3))

# 拼接并截断到45字节（题目要求的flag长度）
return ''.join(plaintext)[:45]

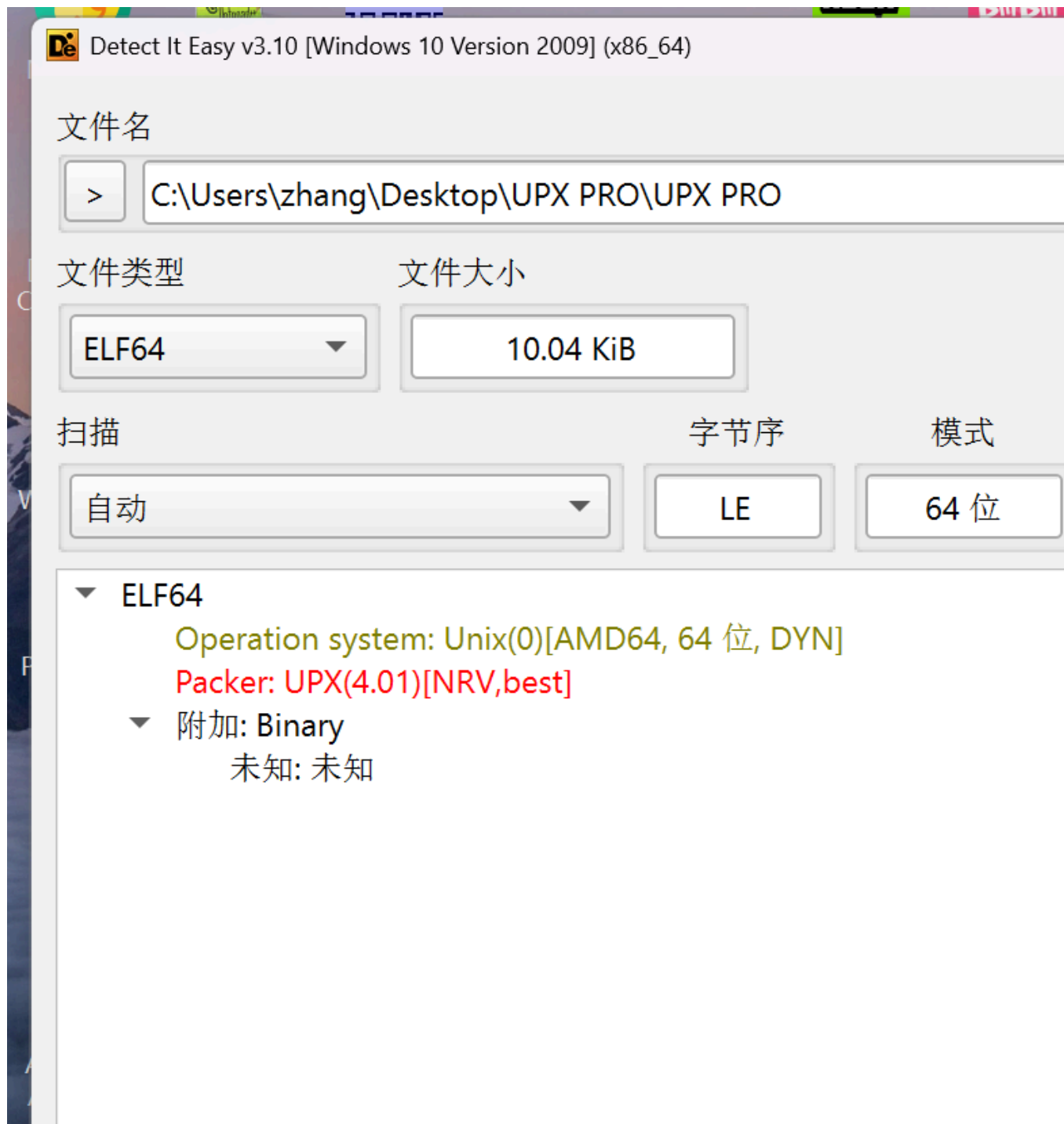
# 已知条件
modified_table =
"CDABGHEFKLIJOPMNSTQRWXUVabYZefcdijghmnklqropuvstyzwx23016745+/89"
key = [1, 2, 3, 4]
ciphertext = "TqK1YUSaQryEMHaLMnWhYU+Fe0WPenqhRXahfkV6WE2fa3iRW197Za62eEaD"

# 解密
flag = decrypt(ciphertext, modified_table, key)
print("解密结果: ", flag)

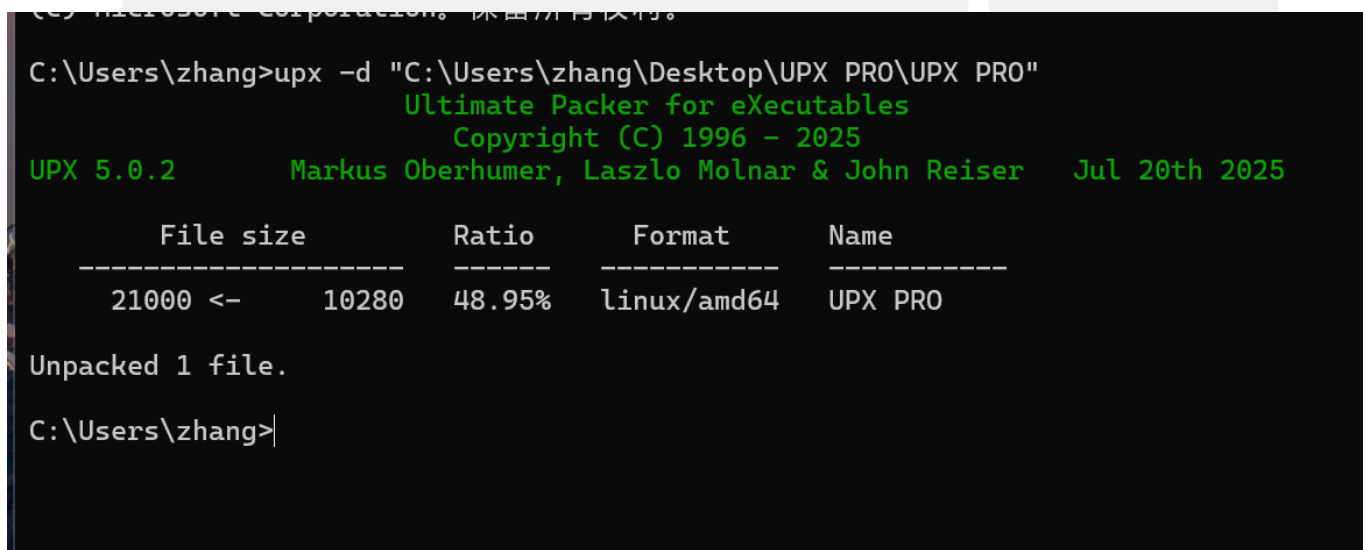
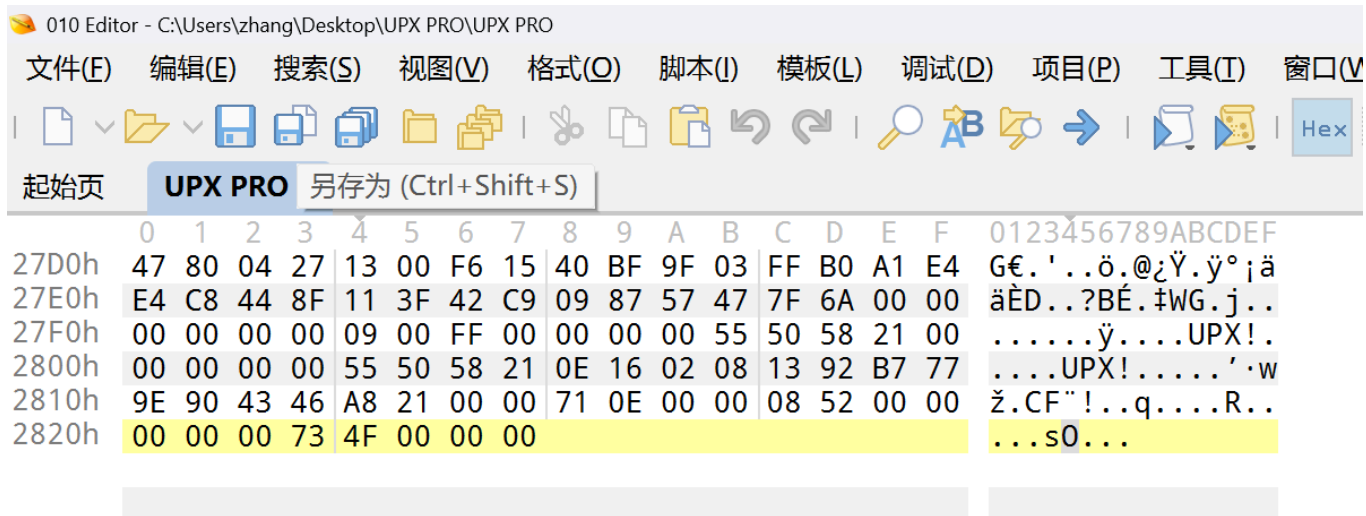
```

六,UPX PRO

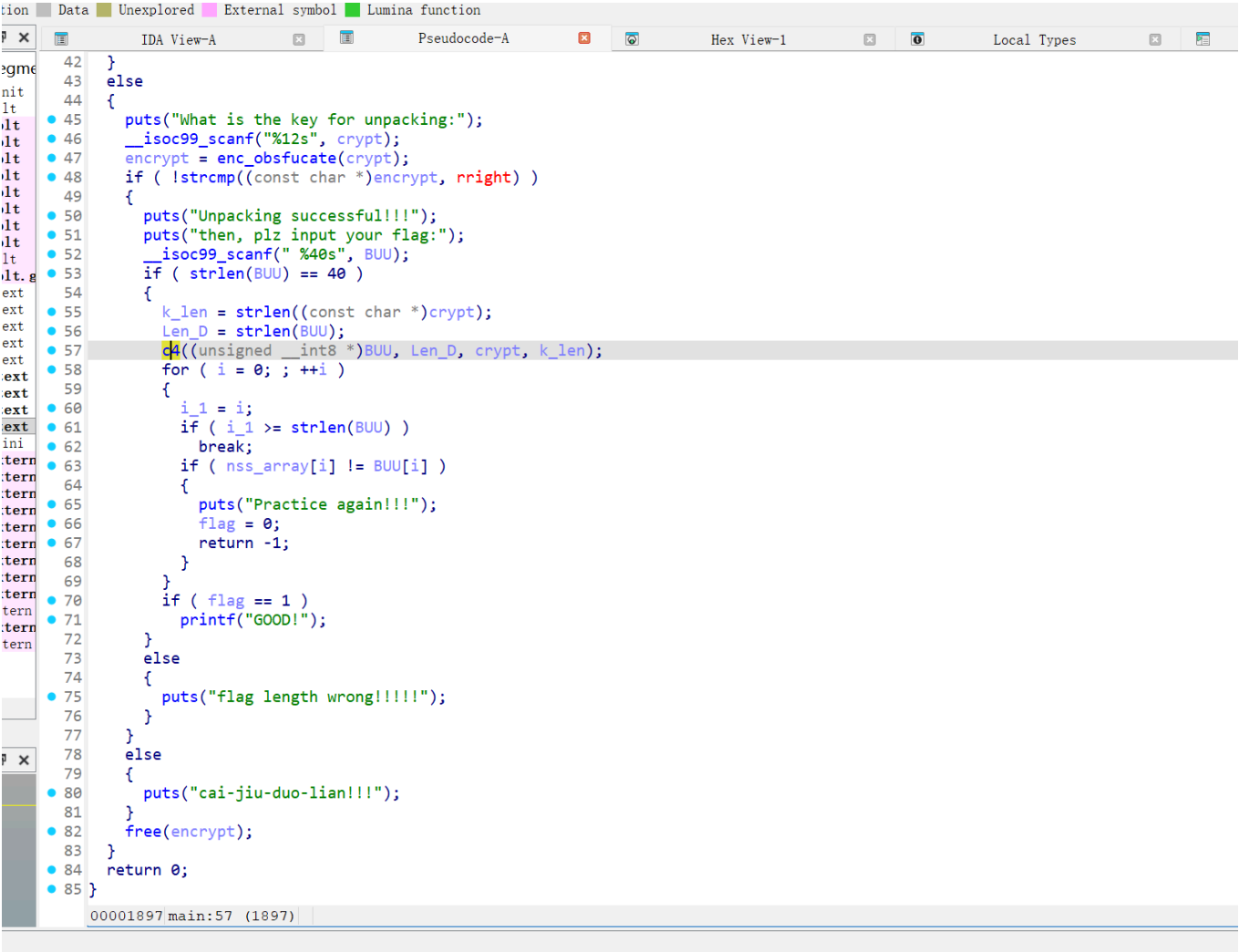
1,有壳,但尝试脱壳失败



2,放到101中看看关键的几处有没有被改,发现最后的标志位错了,将4F改为F4即可脱壳



3,ida先查看encode



```
IDA View-A Pseudocode-A Hex View-1 Local Types Imp
1 unsigned __int8 *__cdecl enc_obsfucate(unsigned __int8 *obf_input)
2 {
3     unsigned int curr; // [rsp+14h] [rbp-4Ch]
4     unsigned __int8 *end_ptr; // [rsp+18h] [rbp-48h]
5     unsigned __int8 *obf_output; // [rsp+20h] [rbp-40h]
6     size_t ret_len; // [rsp+28h] [rbp-38h]
7     size_t obf_len; // [rsp+30h] [rbp-30h]
8     unsigned __int8 rem; // [rsp+3Fh] [rbp-21h]
9     size_t i; // [rsp+40h] [rbp-20h]
10    unsigned __int8 *tmp_ptr; // [rsp+48h] [rbp-18h]
11    unsigned __int8 *ret_ptr; // [rsp+50h] [rbp-10h]
12    unsigned __int8 *interm; // [rsp+58h] [rbp-8h]
13
14    obf_len = strlen((const char *)obf_input);
15    ret_len = 3 * ((obf_len >> 1) + 1);
16    obf_output = (unsigned __int8 *)malloc(ret_len + obf_len);
17    ret_ptr = &obf_output[ret_len];
18    end_ptr = &obf_output[ret_len - 1 + obf_len];
19    interm = &obf_output[ret_len];
20    memcpy(&obf_output[ret_len], obf_input, obf_len);
21    while ( end_ptr >= interm )
22    {
23        if ( *interm )
24        {
25            rem = 0;
26            for ( tmp_ptr = interm; end_ptr >= tmp_ptr; ++tmp_ptr )
27            {
28                curr = rem << 8;
29                rem = *tmp_ptr - 58 * ((*tmp_ptr + curr) / 0x3A);
30                *tmp_ptr = (*tmp_ptr + curr) / 0x3A;
31            }
32            *--ret_ptr = map58_0[rem]; // "ABCDEFGHIJKLMNOPQRSTUVWXYZ123456789abcdefghijklmnopqrstuvwxyz"
33        }
34        else
35        {
36            ++interm;
37        }
38    }
39    for ( i = 0; i < &obf_output[ret_len] - ret_ptr; ++i )
40        obf_output[i] = ret_ptr[i];
41    obf_output[i] = 0;
42    return obf_output;
43 }
```

4.梭哈出来密钥

Recipe

From Base58

Alphabet
ABCDEFGHIJKLMNOPQRSTUVWXYZ...

☒ Remove non-alphabet chars

Input

BQ3SDnTj7vaKkMvur

ABC 17 1

Output

B2a0s2e40815

5,查看c4函数,发现就是改轮数的rc4加密

```
IDA View-A Pseudocode-A Hex View-1 Local Types Import
1 int __cdecl c4(unsigned __int8 *Data, unsigned __int64 Len_D, unsigned __int8 *p_crypt, unsigned __int64 Len_k1)
2 {
3     unsigned __int8 s[135]; // [rsp+20h] [rbp-A0h] BYREF
4     unsigned __int8 tmp; // [rsp+ABh] [rbp-15h]
5     int t; // [rsp+ACCh] [rbp-14h]
6     unsigned __int64 k; // [rsp+B0h] [rbp-10h]
7     int j; // [rsp+B8h] [rbp-8h]
8     int i; // [rsp+BCh] [rbp-4h]
9
10    init(s, p_crypt, Len_k1);
11    i = 0;
12    j = 0;
13    t = 0;
14    for ( k = 0; k < Len_D; ++k )
15    {
16        i = (i + 1) % 128;
17        j = (j + s[i]) % 128;
18        tmp = s[i];
19        s[i] = s[j];
20        s[j] = tmp;
21        t = (s[i] + s[j]) & 0x7F;
22        Data[k] ^= s[t];
23    }
24    return 0;
25 }
```

```
IDA View-A Pseudocode-A Hex View-1
1 int __cdecl init(unsigned __int8 *s, unsigned __int8 *p_crypt, unsigned __int64 Len_k2)
2 {
3     char k[135]; // [rsp+18h] [rbp-90h] BYREF
4     unsigned __int8 tmp; // [rsp+9Fh] [rbp-9h]
5     int j; // [rsp+A0h] [rbp-8h]
6     int i; // [rsp+A4h] [rbp-4h]
7
8     j = 0;
9     memset(k, 0, 128);
10    tmp = 0;
11    for ( i = 0; i <= 127; ++i )
12    {
13        s[i] = i;
14        k[i] = k[i % Len_k2];
15    }
16    for ( i = 0; i <= 127; ++i )
17    {
18        j = (k[i] + j + s[i]) % 128;
19        tmp = s[i];
20        s[i] = s[j];
21        s[j] = tmp;
22    }
23    return 0;
24 }
```

6,解题脚本如下

```
#!/usr/bin/env python3
# coding: utf-8
# solve_flag.py
# 根据题目中的伪代码还原出 BUU (flag)
# 注意: init 函数中 k[i] = k[i % Len_k2] 的实现有 bug (k 全为 0), 脚本按原始伪代码行为实现。

def u64_to_le_bytes(x):
    # 把 64-bit 常量拆成 8 个小端字节 (低位在前)
```

```
return [(x >> (8*i)) & 0xFF for i in range(8)]
```

```
def build_nss_array():
```

```
    vals = [
```

```
        0x364A65466C271216,
```

```
        0x2447243568082139,
```

```
        0x29323C0F5A1A7D60,
```

```
        0x4D647C3C45531544,
```

```
        0x74152339130F7024
```

```
    ]
```

```
    b = []
```

```
    for v in vals:
```

```
        b.extend(u64_to_le_bytes(v))
```

```
    # 只需要前 40 bytes (5*8=40)
```

```
    return bytes(b[:40])
```

```
def init_s_with_bug(len_k2):
```

```
    # 按照题目 init 实现 (包含 bug): k 被 memset 为 0, 之后 k[i] = k[i % Len_k2]
```

```
(仍为0)
```

```
    s = list(range(128))
```

```
    # k 初始化为 128 个 0 (效果与原代码给出一致)
```

```
    k = [0] * 128
```

```
    j = 0
```

```
    for i in range(128):
```

```
        j = (k[i] + j + s[i]) % 128
```

```
        s[i], s[j] = s[j], s[i]
```

```
    return s
```

```
def rc4_prga_bytes(s, length):
```

```
    # 生成 length 个 keystream 字节, 按照题目 PRGA (128 模)
```

```
    s = s[:] # 拷贝
```

```
    i = 0
```

```
    j = 0
```

```
    out = []
```

```
    for k in range(length):
```

```
        i = (i + 1) % 128
```

```
        j = (j + s[i]) % 128
```

```
        s[i], s[j] = s[j], s[i]
```

```
        t = (s[i] + s[j]) & 0x7F
```

```
        out.append(s[t])
```

```
    return bytes(out)
```

```
def c4_xor(data_bytes, key_bytes_len):
```

```
    # 与题目 c4 对应: init 使用 p_crypt 和 Len_k2, 但因 bug 实际上与 p_crypt 无关
```

```
    s = init_s_with_bug(key_bytes_len)
```

```
    ks = rc4_prga_bytes(s, len(data_bytes))
```



```

# Data[k] ^= ks[k]
return bytes([data_bytes[i] ^ ks[i] for i in range(len(data_bytes))])

def main():
    nss = build_nss_array()
    print("nss_array (hex):", nss.hex())

    # 题目中 Len_k2 为 crypt 的长度，但 init 有 bug，实际不影响 s（仍然用 len 做参数
    # 以匹配签名）
    # 这里我们随便放一个非零长度，例如 len("B2a0s2e40815") = 12 来调用 init 的接口
    # （与实际结果无关）
    len_k2 = len("B2a0s2e40815")

    # 生成 keystream（长度 40）
    s_init = init_s_with_bug(len_k2)
    keystream = rc4_prga_bytes(s_init, 40)

    # 题目要求经过 c4(Data, Len_D, crypt, k_len) 之后 Data == nss_array
    # 即 原始 BUU ^ keystream = nss_array -> 原始 BUU = nss_array ^ keystream
    BUU = bytes([nss[i] ^ keystream[i] for i in range(40)])
    try:
        flag_str = BUU.decode('utf-8')
    except UnicodeDecodeError:
        # 若不是有效 UTF-8，强制以 latin-1 显示字节（不过通常是可读 ASCII）
        flag_str = BUU.decode('latin-1')

    print("Recovered BUU (raw bytes):", BUU)
    print("Recovered BUU (as string):")
    print(flag_str)

    # 验证：对 BUU 再运行一次 c4 应得到 nss_array
    verified = c4_xor(BUU, len_k2)
    print("Verification equals nss_array:", verified == nss)
    if verified == nss:
        print("Verified OK.")
    else:
        print("Verification failed! (unexpected)")

if __name__ == "__main__":
    main()

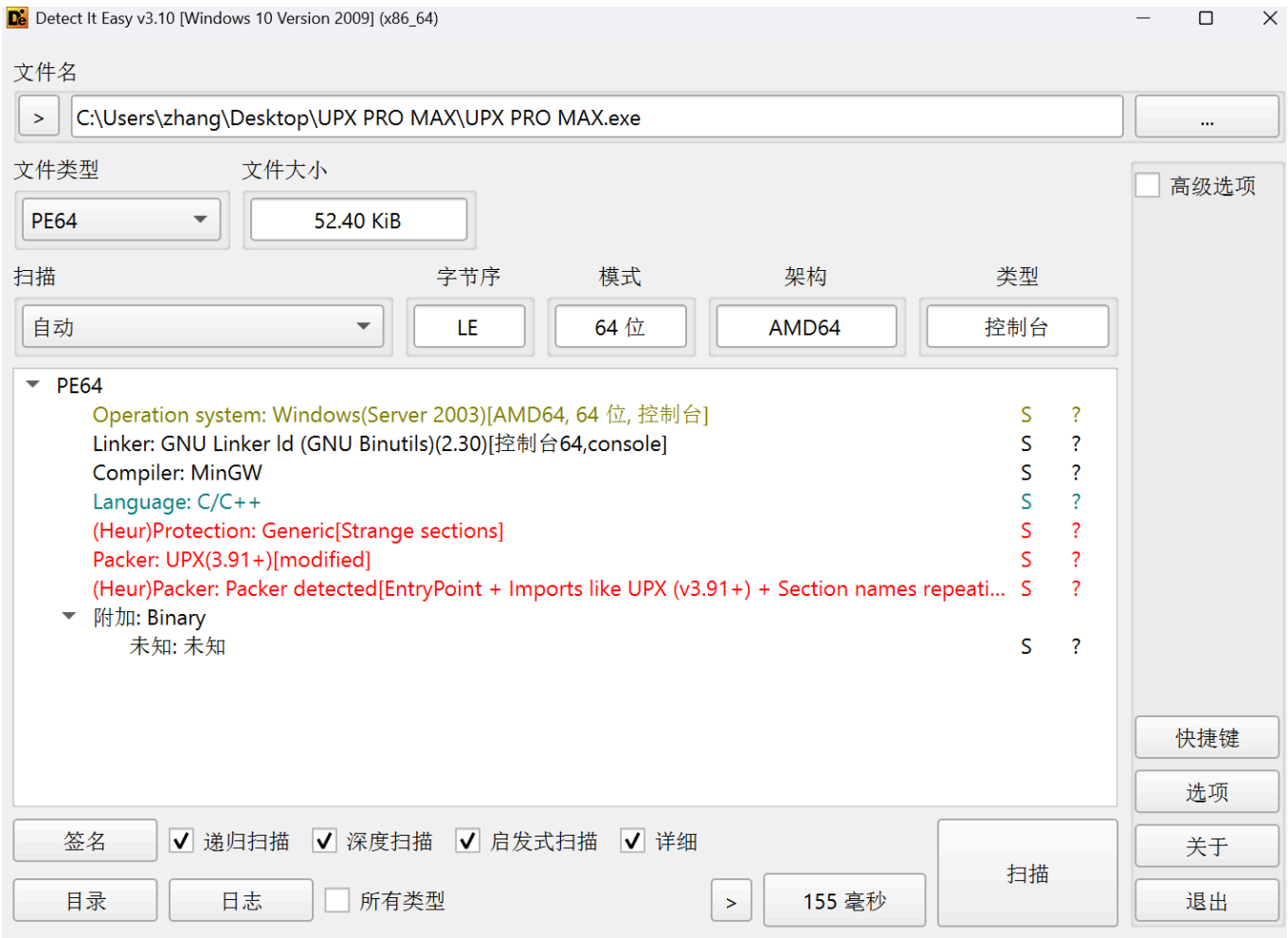
```

7,flag为

```
BaseCTF{Rc4_1$_@_G0od_3nCrypt!on_Meth0d}
```

七,UPX PROMAX

1,有壳,且无法正常脱



2,101中查看标志位,发现都被改了,于是用x64dbg进行分析

UPX PRO MAX.exe - PID: 8120 - 模块: ntdll.dll - 线程: 主线程 37236 - x64dbg

文件(F) 视图(V) 调试(D) 跟踪(N) 插件(P) 收藏夹(I) 选项(O) 帮助(H) Aug 19 2025 (TitanEngine)

CPU 日志 笔记 断点 内存布局 调用堆栈 SEH链 脚本 符号 源代码 引用 线程 句柄 跟踪

RIP → 00007FF929B7F5FE EB 00 jmp ntdll.7FF929B7F600
48:83C4 38 add rsp,38
C3 ret
int3
int3
int3
int3
int3
int3
int3
int3
int3
48:83EC 28 sub rsp,28
6548:880C25 60000000 mov rcx,qword ptr [60]
33D2 xor edx,edx
41:88 20160100 mov r8d,11620
48:8849 30 mov rcx,qword ptr ds:[rcx+30]
E8 A6CBEFFF call <ntdll.RtlAllocateHeap>
48:8905 57F20A00 mov qword ptr ds:[7FF929C2E888],rax
48:85C0 test rax,rax
74 1A je ntdll.7FF929B7F650
33D2 xor edx,edx
41:88 20160100 mov r8d,11620
48:88C8 mov rcx,rax
E8 EA290500 call <ntdll.memset>
C705 90470800 320000 mov dword ptr ds:[7FF929C33DE0],32
48:83C4 28 add rsp,28
C3 ret
int3
int3
int3
int3
int3
int3
int3
int3
int3
int3
ntdll.00007FF929B7F600
32: '2'
r8d: &"\n"|\$@
rcx: ZwQueryInformationThread+14
32: '2'
r8d: &"\n"|\$@
rcx: ZwQueryInformationThread+14

隐藏FPU
RAX 0000000000000000
RBX 00007FF929B7F148 "LdrpInitia
RCX 00007FF929B7C2094 ntdll.00007
RDX 0000000000000000
RBP 0000000000000000
RSP 0000000000000000 L"斜@"
RSI 00007FF929B84888 "minkernel\
RDI 0000000000226000
R8 00000000000061F538 &"\n"|\$@
R9 0000000000000000
R10 0000000000000000
R11 00000000000000246 L'z'
R12 0000000000000000
R13 0000000000000000
R14 0000000000000040 '@'
R15 0000000000000000
RIP 00007FF929B7F5FE ntdll.00007
RFLAGS 0000000000000246
ZF 1 PF 1 AF 0
OF 0 SF 0 DF 0
CF 0 TF 0 IF 1
LastError 00000002 (ERROR_FILE_NOT_FO
LastStatus C0000034 (STATUS_OBJECT_NAM
默认 (x64 fastcall) 5 解锁
1: rcx 00007FF929B7C2094 ntdll.00007FF92
2: rdx 0000000000000000 0000000000000000
3: r8 00000000000061F538 00000000000061F538
4: r9 0000000000000000 0000000000000000
5: [rsp+28] 0000000000000040 0000000000000000

内存 1 内存 2 内存 3 内存 4 内存 5 监视 1 [x] 局部变量 结构体
地址 十六进制 十六进制 ASCII
00007FF929A61000 CC CC CC CC 48 89 5C 24 08 48 89 6C iiiihih.\\$.H.
00007FF929A61010 24 10 48 89 74 24 18 57 41 56 41 57 48 83 EC 20 \$.H.t\$.WAWAWH.i
00007FF929A61020 65 48 8B 04 25 60 00 00 00 49 8B F0 44 0F B7 01 eh.%....T.Db..
00007FF929A61030 4C 8B F1 8B EA 33 D2 4C 8B 78 30 49 8B CF E8 80 L.n.e30L.X0T.Ie..
00007FF929A61040 81 01 00 33 DB 48 8B F8 48 85 C0 75 09 88 1E 8B z..30H.OH.Au...>
00007FF929A61050 17 00 C0 45 C6 06 01 48 8B CF 45 0F B7 06 ..AeE..H.Ee..
00007FF929A61060 49 8B 56 08 E8 D7 62 16 00 48 8B C5 B2 01 D1 ED I.V.eXb..H.A².Ni
00007FF929A61070 48 8B CF 48 D1 E8 FF C5 66 C7 04 47 2E 00 66 89 H.IHNeYafG..G..f.
00007FF929A61080 1C 6F E8 D9 15 00 00 84 C0 4C 8B C7 49 8B CF 0F .oE...AL.C.I.i.
00007FF929A61090 94 C2 8B 16 33 D2 E8 45 7C 07 00 48 8B 6C 24 48 .A..30e|..H.1\$H
00007FF929A610A0 8B C3 48 8B 5C 24 40 48 8B 74 24 50 48 83 C4 20 .AH.\\\$.H.t\$.PH.A

00000000000061F540 0000000000226000
00000000000061F548 00007FF929B5BD41
00000000000061F550 0000000000000000
00000000000061F558 00007FF929B7F148
00000000000061F560 0000000000000000
00000000000061F568 0000000000000040
00000000000061F570 0000000000226000
00000000000061F578 00007FF929B1D843
00000000000061F580 00007FF929B7F100
00000000000061F588 00007FF929B84888
00000000000061F590 00007FF929B84888
00000000000061F598 0000000000226000
00000000000061F5A0 00007FF929B7F4C0
00000000000061F5A8 0000000000796198
00000000000061F5B0 0000000000796188
返回到 ntdll.RtlQueryElevation
ntdll._fltused+F8C0
返回到 ntdll.EtwEventWriteNoR
ntdll._fltused+F878
ntdll.RtlNtdllName+36A8
ntdll.RtlNtdllName+36A8
ntdll._fltused+FC38
i "dF"

3,f9运行后观察到是esp定律

RIP RAX RDX R9 → 0000000000041CD0E 0000 FFD0 add byte ptr ds:[rax],al
0000000000041CD0F 05 F5 inc dword ptr ds:[rax]
0000000000041CD12 F5 A2 010080040000F000 mov byte ptr ds:[FF000004800001],al
0000000000041CD13 0000 add byte ptr ds:[rax],al
0000000000041CD1C 0000 add byte ptr ds:[rax],al
0000000000041CD30 53 push rbx
0000000000041CD41 56 push rsi
0000000000041CD42 57 push rdi
0000000000041CD43 55 push rbp
0000000000041CD44 48:8D35 DAA2FFFF lea rsi,qword ptr ds:[417025]
0000000000041CD45 48:8DBE DB9FEFFF lea rdi,qword ptr ds:[rsi-16025]
0000000000041CD52 57 push rdi
0000000000041CD53 31DB xor ebx,ebx
0000000000041CD55 31C9 xor ecx,ecx
0000000000041CD57 48:83CD FF or rbp,FFFFFFFFFFFFFFF
0000000000041CD58 E8 50000000 call upx.pro.max.41CD80
0000000000041CD60 01DB add ebx,ebx
0000000000041CD62 74 02 je upx.pro.max.41CD66
0000000000041CD64 F3:C3 ret
0000000000041CD66 8B1E mov ebx,dword ptr ds:[rsi]
0000000000041CD68 48:83EE FC sub rsi,FFFFFFFFFFFFFFF
0000000000041CD6C 11DB adc ebx,ebx
0000000000041CD6E 8A16 mov dl,byte ptr ds:[rsi]
0000000000041CD70 F3:C3 ret
0000000000041CD72 48:8D042F lea rax,qword ptr ds:[rdi+rbp]
0000000000041CD76 83D9 05 cmp ecx,5
0000000000041CD79 8A10 mov dl,byte ptr ds:[rax]
0000000000041CD7B 76 21 jbe upx.pro.max.41CD9E
OptionalHeader.AddressOfEntryPoint

4,打开rsp的内存布局,设置断点然后再调试,即可发现第一部分解压完成了,变成了pop形式

RAX	00007FF927A48620	<kerne132.Get
RBX	00007FF927A30000	kerne132.0000
RCX	5DB0A01B0F760000	
RDX	00007FF927A48620	<kerne132.Get
RBP	00007FF927A30000	kerne132.0000
RSP	0000000000061FF00	L"斜@"
RSI	00000000000401000	upx.pro.max.(
RDI	0000000000041B071	upx.pro.max.(
R8	00000000000000245	L'Δ'
R9	00000000000000246	L'Δ'
R10	00000000000000245	L'Δ'
R11	00000000000000245	L'Δ'
R12	00000000000000000	
R13	00000000000000000	

rsi=0
qword ptr ds:[upx pro
:000000000041CD44 upx

内存 1 内存 2

地址(A) 反汇编(D)

在反汇编中转到
在栈中转到
在内存布局中转到
给当前地址写标签
监视 QWORD(W)
修改(M) Space
断点(B)
搜索匹配特征(F)... Ctrl+B
搜索引用(R) Ctrl+R
与表达式同步(S) S
分配内存
转到(G)
十六进制(H)
文本(T)
整数(I)
浮点数(F)
地址(A)
反汇编(D)

ret
mov ebx,dword ptr ds:[rsi]
sub rsi,FFFFFFFFFFFFFFFC
adc ebx,ebx
mov dl,byte ptr ds:[rsi]
ret
lea rax,qword ptr ds:[rdi+rbp]

硬件, 访问(A)
硬件, 写入(W)
硬件, 执行(E)
内存, 访问
内存, 读取
内存, 写入
内存, 执行

字节(B)
2字节(W)
4字节(D)
8字节(Q)

6EFFFFB

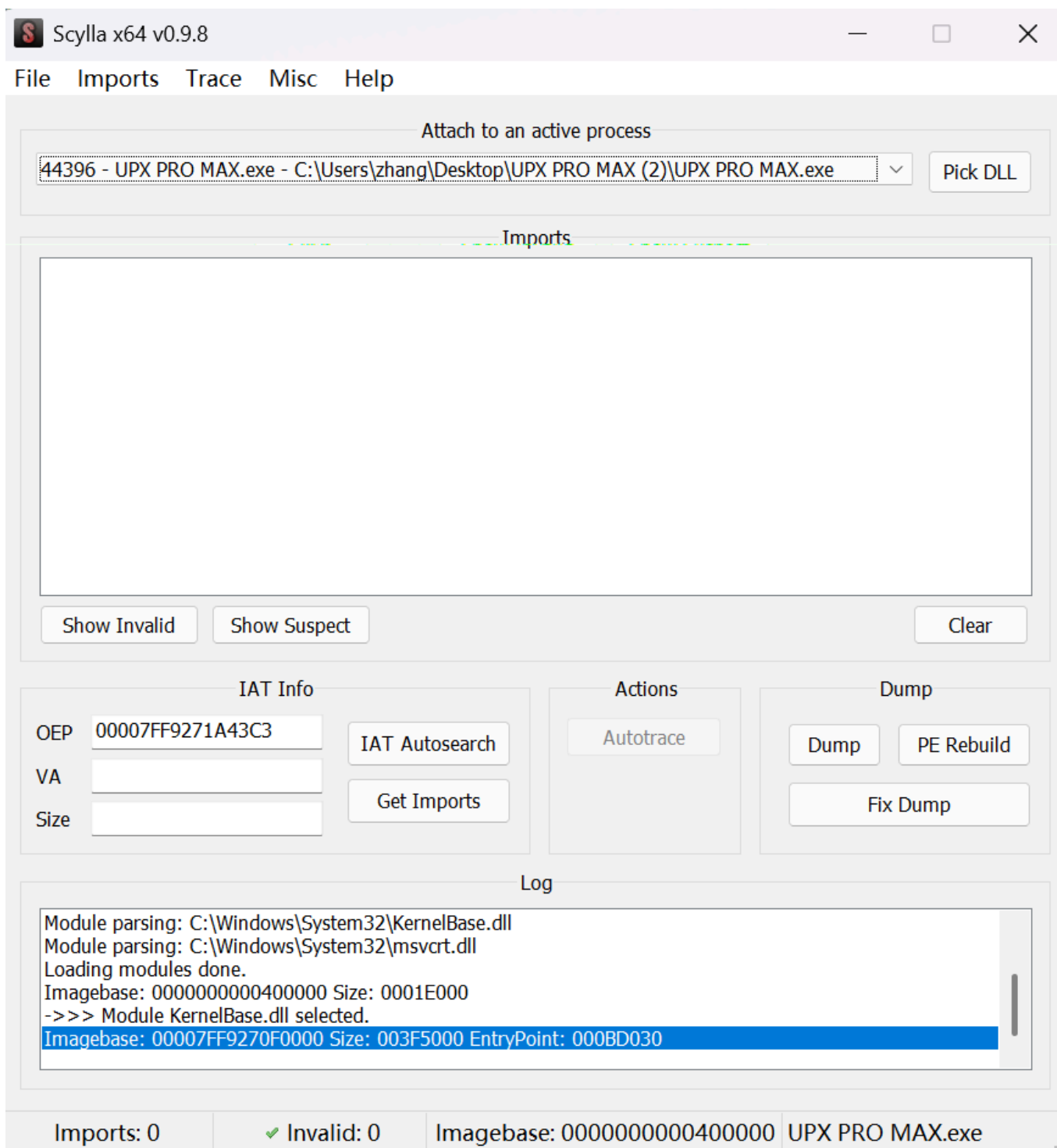
监视 1 | [x=] 局部变量 | 结构体

地址	十六进	ASCII
000000000061FF08	00 00 00 00 00 00 00 00	
000000000061FF18	00 00 00 00 00 00 00 00	
000000000061FF28	7D 25 64 3E F9 7F 00 00	}%d>ù..
000000000061FF38	00 00 00 00 00 00 00 00	
000000000061FF48	00 00 00 00 00 00 00 00	
000000000061FF58	28 AF 94 3F F9 7F 00 00	(".?ù..
000000000061FF68	00 00 00 00 00 00 00 00	
000000000061FF78	00 00 00 00 00 00 00 00	
000000000061FF88	00 00 00 00 00 00 00 00	
000000000061FF98	00 00 00 00 00 00 00 00	
000000000061FFA8	30 FB FF FF D0 04 00 00	0úÿÿð..
000000000061FFB8	00 00 00 00 00 00 00 00	0úÿÿð..!

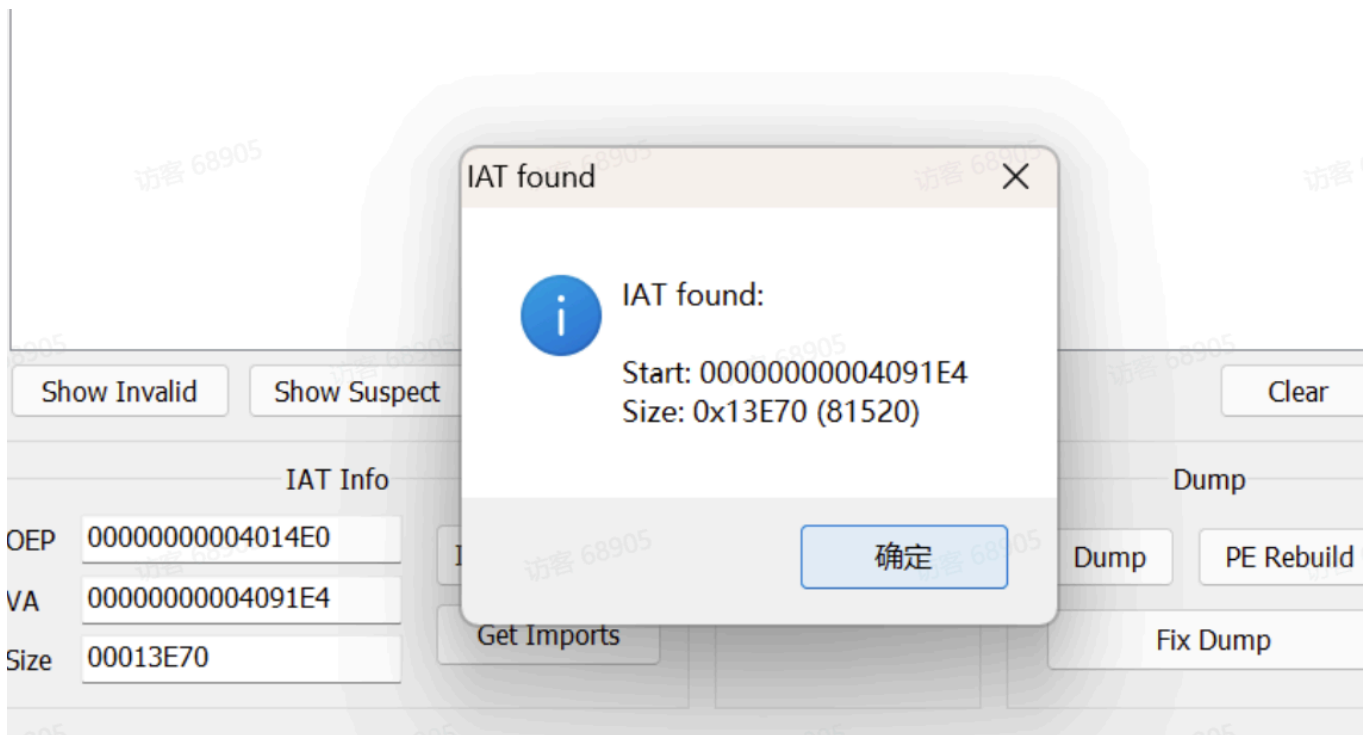
000000000041CF46	5A	pop rdx	
000000000041CF47	4D:31C0	xor r8,r8	
000000000041CF4A	50	push rax	
000000000041CF4B	E8 1A000000	call upx_pro_max.41CF6A	
000000000041CF50	58	pop rax	
000000000041CF51	5D	pop rbp	
000000000041CF52	5F	pop rdi	
000000000041CF53	5E	pop rsi	
000000000041CF54	5B	pop rbx	
000000000041CF55	48:8D4424 80	lea rax,qword ptr ss:[rsp-80]	
000000000041CF5A	6A 00	push 0	
000000000041CF5C	48:39C4	cmp rsp,rax	
000000000041CF5F	^ 75 F9	jne upx_pro_max.41CF5A	
000000000041CF61	48:83EC 80	sub rsp,FFFFFFFFFFFFFF80	
000000000041CF65	- E9 7645FEFF	jmp upx_pro_max.4014E0	
000000000041CF6A	FC	cld	
000000000041CF6B	56	push rsi	
000000000041CF6C	48:8D35 CDD0FEFF	lea rsi,qword ptr ds:[40A040]	
000000000041CF73	48:AD	lodsq	
000000000041CF75	48:85C0	test rax,rax	
000000000041CF78	✓ 74 14	je upx_pro_max.41CF8E	
000000000041CF7A	51	push rcx	
000000000041CF7B	52	push rdx	
000000000041CF7C	41:50	push r8	
000000000041CF7E	48:83EC 28	sub rsp,28	
000000000041CF82	FFD0	call rax	
000000000041CF84	48:83C4 28	add rsp,28	
000000000041CF88	41:58	pop r8	
000000000041CF8A	5A	pop rdx	
000000000041CF8B	59	pop rcx	
000000000041CF8C	^ EB E5	jmp upx_pro_max.41CF73	
000000000041CF8E	5E	pop rsi	
000000000041CF8F	C3	ret	
000000000041CF90	B8 CF410000	mov eax,41CF	
000000000041CF95	0000	add byte ptr ds:[rax],al	
000000000041CF97	00C0	add al,al	
000000000041CF99	CF	iretd	
000000000041CF9A	41:0000	add byte ptr ds:[r8],al	
000000000041CF9D	0000	add byte ptr ds:[rax],al	
000000000041CF9F	00FC	add ah,bh	
000000000041CFA1	8540 00	test dword ptr ds:[rax],eax	
000000000041CFA4	0000	add byte ptr ds:[rax],al	
000000000041CFA6	0000	add byte ptr ds:[rax],al	
000000000041CFA8	C0CF 41	ror bh,41	

5,通过f8我们可以依靠jmp跳到正确有效的函数入口,此时就可以使用scylla脱壳了

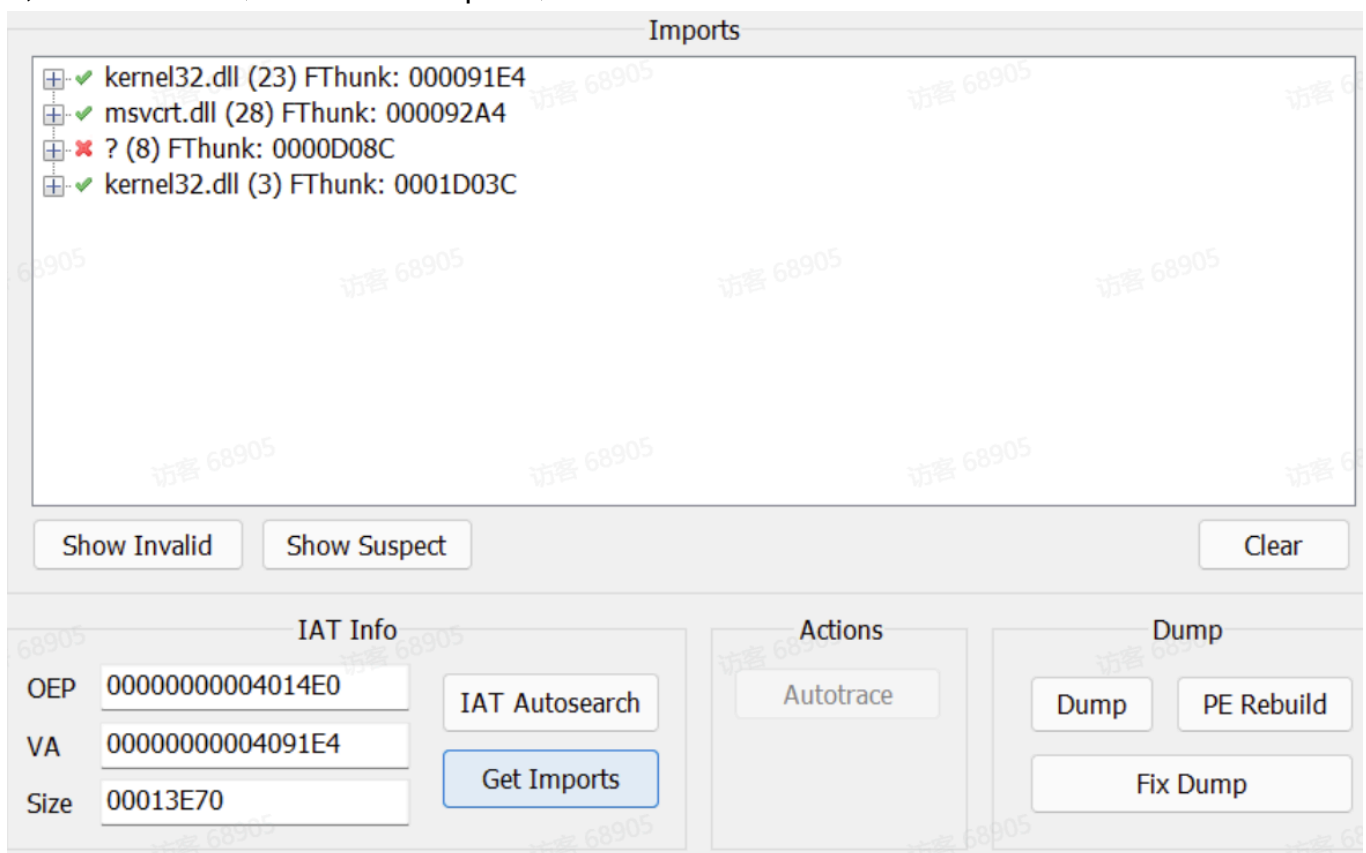
00000000004014E0	48:83EC 28	sub rsp,28	
00000000004014E4	48:8B05 F5420000	mov rax,qword ptr ds:[4057E0]	00000000004057E0:L"藏@"
00000000004014EB	C700 00000000	mov dword ptr ds:[rax],0	
00000000004014F1	E8 CA080000	call upx_pro_max.401DC0	
00000000004014F6	E8 85FCFFFF	call upx_pro_max.401180	
00000000004014FB	90	nop	
00000000004014FC	90	nop	
00000000004014FD	48:83C4 28	add rsp,28	
0000000000401501	C3	ret	
0000000000401502	0F1F40 00	nop dword ptr ds:[rax],eax	
0000000000401506	662E:0F1F8400 00000000	nop word ptr ds:[rax+rax],ax	
0000000000401510	48:83EC 28	sub rsp,28	
0000000000401514	E8 EF1C0000	call <JMP.&_onexit>	
0000000000401519	48:85C0	test rax,rax	
000000000040151C	0F94C0	sete al	
000000000040151F	0FB6C0	movzx eax,al	
0000000000401522	F7D8	neg eax	
0000000000401524	48:83C4 28	add rsp,28	
0000000000401528	C3	ret	
0000000000401529	90	nop	
000000000040152A	90	nop	
000000000040152B	90	nop	
000000000040152C	90	nop	
000000000040152D	90	nop	
000000000040152E	90	nop	
000000000040152F	90	nop	
0000000000401530	48:8D0D 09000000	lea rcx,qword ptr ds:[401540]	
0000000000401537	F9 D4FFFFFF	imn upx_pro_max.401510	



6,点击Dump, 然后保存好文件, 再点击IAT Autosearch,



7,看到IAT已找到，再点击Get Imports，



8,点击Fix Dump,选择刚才Dump下来的文件就可以了,现在就可以ida分析了,


```

1 _int64 sub_401806()
2 {
3     unsigned __int64 v1; // rbx
4     unsigned __int64 v2; // rbx
5     _DWORD v3[44]; // [rsp+40h] [rbp-40h] BYREF
6     _QWORD v4[4]; // [rsp+F0h] [rbp+70h] BYREF
7     _QWORD v5[4]; // [rsp+110h] [rbp+90h] BYREF
8     _QWORD v6[4]; // [rsp+130h] [rbp+B0h] BYREF
9     _QWORD v7[4]; // [rsp+150h] [rbp+D0h] BYREF
10    _QWORD v8[4]; // [rsp+170h] [rbp+F0h] BYREF
11    _QWORD v9[4]; // [rsp+190h] [rbp+110h] BYREF
12    _BYTE v10[48]; // [rsp+1B0h] [rbp+130h] BYREF
13    char v11[52]; // [rsp+1E0h] [rbp+160h] BYREF
14    int j; // [rsp+214h] [rbp+194h]
15    int i; // [rsp+218h] [rbp+198h]
16    int n6; // [rsp+21Ch] [rbp+19Ch]
17
18    sub_401D80();
19    sub_4031B8(asc_405000);
20    sub_4031B8(asc_4050A0);
21    sub_4031B8(asc_405140);
22    sub_4031B8(asc_4051E0);
23    sub_4031B8(asc_405280);
24    qword_40926C(500);
25    sub_4031B8(aAtThisPointShe);
26    sub_4031B8(aIHopeYouEnjoye);
27    sub_4031B8(aPlzInputYourFl);
28    memset(v9, 0, sizeof(v9));
29    memset(v8, 0, sizeof(v8));
30    memset(v7, 0, sizeof(v7));
31    memset(v6, 0, sizeof(v6));
32    memset(v5, 0, sizeof(v5));
33    memset(v4, 0, sizeof(v4));
34    v3[42] = 0;
35    v3[0] = 34;
36    v3[1] = 17;
37    v3[2] = 23;
38    v3[3] = 33;
39    v3[4] = 22;
40    v3[5] = 17;
41    v3[6] = 60;
42    v3[7] = 61;
43    v3[8] = 36;

```

00000BCD sub_401806:40 (4019CD)

9,这里就很简单了,前面去壳是真几把麻烦,这里扔给gpt再让他进行异或运算就可以,脚本如下

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <stdbool.h>
#include <windows.h>
#include <winternl.h>

int main() {
    int enc[] = {0x22, 0x11, 0x17, 0x21, 0x16, 0x11, 0x3C, 0x23, 0x65, 0x5C,
                 0x2E, 0x74, 0x7C, 0x7D, 0x6D, 0x72, 0x6C, 0x0E, 0x36, 0x34,
                 0x64, 0x42, 0x57, 0x4E, 0x3B, 0x24, 0x36, 0x3A, 0x2C, 0x6D,
                 0x43, 0x13, 0x7A, 0x68, 0x11, 0x3D, 0x24, 0x10, 0x2E, 0x52,
                 0x5D, 0x29};

    for (int i = 41; i > 0; i--) {
        enc[i - 1] = enc[i - 1] ^ enc[i];
    }

    for (int j = 0; j < 42; j++) {
        printf("%c", enc[j] ^ j ^ '});
    }

    return 0;
}

```


10,(hy告诉了我了个简单方法,不用断点调试,f9运行到cmd卡住,此时直接dump就出来了)