DRAGON QUEST

`(x|y)\d+` 的变量都只有读的引用, 程序运行期间必为0, 写个脚本查交叉引用, 去除花指令(IDA 7.0及以上):

```python
import ida_xref
import ida_idaapi
from ida_bytes import get_bytes, patch_bytes

def do_patch(ea):
    if(get_bytes(ea, 1) == "\x8B"): # mov eax-edi, dword
        reg = (ord(get_bytes(ea + 1, 1)) & 0b00111000) >> 3
        patch_bytes(ea, chr(0xB8 + reg) + "\x00\x00\x00\x00\x90\x90")
    elif(get_bytes(ea, 2) == "\x44\x8B"): # mov r8d-r15d, dword
        reg = (ord(get_bytes(ea + 2, 1)) & 0b00111000) >> 3
        patch_bytes(ea + 1, chr(0xB8 + reg) + "\x00\x00\x00\x00\x90\x90")

for addr in xrange(0x610318, 0x6105AC, 4):
    ref = ida_xref.get_first_dref_to(addr)
    print(hex(addr).center(20,"-"))
    while(ref != ida_idaapi.BADADDR):
        do_patch(ref)
        print("patch at " + hex(ref))
        ref = ida_xref.get_next_dref_to(addr, ref)
    print("-"*20)
```

solver:

```python
secret = [100, 214, 266, 369, 417, 527, 622, 733, 847, 942, 1054, 1106, 1222
, 1336, 1441, 1540, 1589, 1686, 1796, 1891, 1996, 2112, 2165, 2260, 2336, 24
12, 2498, 2575]
n = 0
flag = ""
for i in xrange(0, len(secret)):
    ch = secret[i] - n
    n += ch
    flag += chr(ch)
print(flag)
```

flag: `flag{dr4g0n_or_p4tric1an_it5_LLVM}`

EASYVM

bytecode:

```
05010B mov r1, rB // input
130303 xor r3, r3
130000 xor r0, r0
130404 xor r4, r4

28 enter loop
0C0033 add r0, 33
140020 mod r0, 20
050901 mov r9, r1
110900 add_pch r9, r0
0B0A09 ldr_ch rA, [r9]
01040A mov r4, rA
1B0504 push r5, r4 // push as int
0C0301 add r3, 1
240320 cmp r3, 20
28 jl
--------------
130000 xor r0, r0
070805 lea_int r8, r5
0E08E0 add_pint r8, E0
070208 lea_int r2, r8
090A02 ldr_int rA, r2
01000A mov r0, rA
1800E0 and r0, E0
1E0005 shr r0, 5
010400 mov r4, r0
130303 xor r3, r3

28 enter loop
090A02 ldr_int rA, r2
01000A mov r0, rA
18001F and r0, 1F
200003 shl r0, 3
1B0500 push r5, r0
070805 lea_int r8, r5
0E08E0 add_pint r8, E0 // -0x20
070208 lea_int r2, r8
090A02 ldr_int rA, r2
01000A mov r0, rA
1800E0 and r0, E0
1E0005 shr r0, 5
1D050A pop r5, rA
```

```
0D0A00 add rA, r0
1B050A push r5, rA
0C0301 add r3, 1
24031F cmp r3, 1F
28 jl

090A02 ldr_int rA, r2
01000A mov r0, rA
18001F and r0, 1F
200003 shl r0, 3
0D0004 add r0, r4
1B0500 push r5, r0
----------------------------
130303 xor r3, r3
03040D mov32 r4, rD // 0xEFBEADDE
28 enter loop
070805 lea_int r8, r5
0E08E0 add_pint r8, E0
070208 lea_int r2, r8
090A02 ldr_int rA, r2
01000A mov r0, rA
1B0500 push r5, r0
010004 mov r0, r4
0D0003 add r0, r3
1D050A pop r5, rA
130A00 xor rA, r0
1B050A push r5, rA
220408 ror r4, 8
0C0301 add r3, 1
240320 cmp r3, 1F
28 jl
----------------------------
130303 xor r3, r3
130404 xor r4, r4
05010C mov r1, rC // secret
28 enter loop
050901 mov r9, r1
110903 add r9, r3
0B0A09 ldr_ch rA, [r9]
01000A mov r0, rA
1B0500 push r5, r4
070805 lea_int r8, r5
0E08DF add_pint r8, DF
090A08 ldr_int rA, r8
1D0500 pop r5, r0
1B0500 push r5, r0
27000A cmpeq r0, rA
170407 or r4, r7
```

```
0C0301 add r3, 1
240320 cmp r3, 1F
28 jl
----------------------------
2A final
```

solver:

```
secret = [0x75, 0x85, 0xD1, 0x39, 0x0B, 0x29, 0xCD, 0x77, 0x6D, 0x9F, 0x73,
0x23, 0x61, 0x8B, 0x4D, 0x45, 0x9D, 0x8F, 0x5B, 0x11, 0xC1, 0xC9, 0xE5, 0xCF
, 0x45, 0xE5, 0xB1, 0xB3, 0x41, 0xD9, 0xCF, 0xCF]
key = [0xDE, 0xAD, 0xBE, 0xEF]

x = [secret[i] ^ ((key[i % 4] + i) & 0xFF) for i in xrange(0, 32)]
# --------------------------
y = [0 for i in xrange(0, 32)]
def fn(a, b):
    return ((a << 5) & 0xE0) | ((b >> 3) & 0x1F)
for i in xrange(1, 32):
    y[i] = fn(x[i-1], x[i])
y[0] = fn(x[31], x[0])
# --------------------------
z = [0 for i in xrange(0, 32)]
k = 0
for i in xrange(0, 32):
    k += 0x33
    z[k % 0x20] = y[i]

flag = ""
for c in z:
    flag += chr(c)
print flag
```

flag: `xman{ae791f19bdf77357ff10bb6b0e}`

EASYWASM

The WebAssembly Binary Toolkit

wasm2c: `wasm2c easywasm.wasm -o easywasm.c`

solver:

```
from hashlib import md5

t = open("data", "rb").read().split("\x00")

flag = ""
for i in xrange(0, 32):
    ci = 0
    for cc in xrange(0x20, 0x7F):
        ch = chr(cc)
        x = md5("23333333333333333333333333333333" + ch).digest()
        x = x.encode("hex").lower()
        x = md5(x).digest()
        x = x.encode("hex").lower()
        if(x == t[i]):
            ci = cc
            break
    assert(ci != 0)
    flag += chr(ci)
print(flag)
```

flag: `xman{99754106633f94d350db34d548}`

SWAG

solver:

```
import numpy

B = \
[
0x106, 0x245, 0x09C, 0x1E2, 0x224, 0x27A,
0x112, 0x0AE, 0x323, 0x3C4, 0x370, 0x0DC,
0x387, 0x01E, 0x0B6, 0x3D8, 0x35D, 0x13A,
0x2B9, 0x162, 0x083, 0x225, 0x057, 0x18C,
0x109, 0x21B, 0x319, 0x0EE, 0x2C1, 0x1D5,
0x23A, 0x19A, 0x145, 0x25E, 0x32A, 0x1D6
]

C = \
[
0x4E36B, 0x362D6, 0x3D5F1, 0x63C4C, 0x66AF7, 0x418B7,
0x4BE2E, 0x35571, 0x3DA7F, 0x60D4A, 0x6423A, 0x3FC18,
0x3A3B6, 0x2FBEE, 0x38F5B, 0x509E4, 0x57DAE, 0x37D25,
0x2E69A, 0x28B2A, 0x363B1, 0x41DAE, 0x49FA8, 0x2D536,
0x3B440, 0x28D5B, 0x3AF48, 0x51F80, 0x59294, 0x30E5F,
0x47CF0, 0x34F47, 0x33520, 0x547A8, 0x581E0, 0x3E875
]

b = numpy.mat(B).reshape(6,6)
c = numpy.mat(C).reshape(6,6)
a = c * b.I

f = a.T.reshape(1,36).tolist()[0]
flag = ""
for v in f:
    flag += chr(int(round(v)))
print(flag)
```

flag: `Flag{mayF0rcebew1thY0Ucbd862jknyzioor26ff}`