Write-up_picoCTF_2021

Chall: Some Assembly Required 4

You can find out the source of challenge and script here(script, wasm and decompiled code)

The key steps to solve

1. Get file web assembly(.wasm) from this challenge webpage
2. Install two packets including **wabt** toolkit and **binaryen**
3. Using following command to decompile the web assembly to pseudo-code format:

```
$ wasm-decompile ZoRd23o0wd > output
```

4. Analyzing the **check_flag()** function of file which contains the pseudo code, I recognize that there are three stages in processing the entered flag:

- Stage 1:

```
export function check_flag():int {
  var a:int = g_a;
  var b:int = 16;
  var c:int = a - b;
  g_a = c;
  var d:int = 0;
  c[3]:int = d;
  loop L_b {
    var e:ubyte_ptr = c[3]:int;
    var f:int = e[1072];
    var g:int = 24;
    var h:int = f << g;
    var i:int = h >> g;
    if (eqz(i)) goto B_a;    // flag[i] == 0 then break Stage 1
    var j:int = 0;
    var k:int = c[3]:int;
    var l:int = k[1072]:ubyte;
```

```
    var m:int = 24;
    var n:int = l << m;
    var o:int = n >> m;
    var p:int = 20;
    var q:int = o ^ p;    //flag[i] ^= 20
    k[1072]:byte = q;
    var r:int = c[3]:int;
    var s:int = r;
    var t:int = j;
    var u:int = s > t;
    var v:int = 1;
    var w:int = u & v;
    if (eqz(w)) goto B_c;  //if i <= 0 then jump
    var x:int = c[3]:int;
    var y:int = 1;
    var z:ubyte_ptr = x - y;
    var aa:int = z[1072];
    var ba:int = 24;
    var ca:int = aa << ba;
    var da:int = ca >> ba;
    var ea:int = c[3]:int;
    var fa:int = ea[1072]:ubyte;
    var ga:int = 24;
    var ha:int = fa << ga;
    var ia:int = ha >> ga;
    var ja:int = ia ^ da;  //else flag[i]^=flag[i - 1]
    ea[1072]:byte = ja;
    label B_c:
    var ka:int = 2;
    var la:int = c[3]:int;
    var ma:int = la;
    var na:int = ka;
    var oa:int = ma > na;
    var pa:int = 1;
    var qa:int = oa & pa;
    if (eqz(qa)) goto B_d;  //if i <=2 the jump
    var ra:int = c[3]:int;
    var sa:int = 3;
    var ta:ubyte_ptr = ra - sa;
    var ua:int = ta[1072];
    var va:int = 24;
    var wa:int = ua << va;
    var xa:int = wa >> va;
    var ya:int = c[3]:int;
    var za:int = ya[1072]:ubyte;
    var ab:int = 24;
    var bb:int = za << ab;
    var cb:int = bb >> ab;
    var db:int = cb ^ xa;  //else flag[i]^=flag[i - 3]
    ya[1072]:byte = db;
    label B_d:
```

```
70      var eb:int = c[3]:int;
71      var fb:int = 10;
72      var gb:int = eb % fb;
73      var hb:int = c[3]:int;
74      var ib:int = hb[1072]:ubyte;
75      var jb:int = 24;
76      var kb:int = ib << jb;
77      var lb:int = kb >> jb;
78      var mb:int = lb ^ gb;  // flag[i] ^=(i % 10)
79      hb[1072]:byte = mb;
80      var nb:int = c[3]:int;
81      var ob:int = 2;
82      var pb:int = nb % ob;
83      if (pb) goto B_f;       //if i %2 != 0 the jump
84      var qb:int = c[3]:int;
85      var rb:int = qb[1072]:ubyte;
86      var sb:int = 24;
87      var tb:int = rb << sb;
88      var ub:int = tb >> sb;
89      var vb:int = 9;
90      var wb:int = ub ^ vb;  //else flag[i]^=9
91      qb[1072]:byte = wb;
92      goto B_e;
93      label B_f:
94      var xb:int = c[3]:int;
95      var yb:int = xb[1072]:ubyte;
96      var zb:int = 24;
97      var ac:int = yb << zb;
98      var bc:int = ac >> zb;
99      var cc:int = 8;
100     var dc:int = bc ^ cc;  //i%2 != 0: flag[i]^=8
101     xb[1072]:byte = dc;
102     label B_e:
103     var ec:int = c[3]:int;
104     var fc:int = 3;
105     var gc:int = ec % fc;
106     if (gc) goto B_h;
107     var hc:int = c[3]:int;
108     var ic:int = hc[1072]:ubyte;
109     var jc:int = 24;
110     var kc:int = ic << jc;
111     var lc:int = kc >> jc;
112     var mc:int = 7;
113     var nc:int = lc ^ mc;  //i%3 = 0: flag[i]^= 7
114     hc[1072]:byte = nc;
115     goto B_g;
116     label B_h:
117     var oc:int = 1;
118     var pc:int = c[3]:int;
119     var qc:int = 3;
120     var rc:int = pc % qc;
```

```
    var sc:int = rc;
    var tc:int = oc;
    var uc:int = sc == tc;
    var vc:int = 1;
    var wc:int = uc & vc;
    if (eqz(wc)) goto B_j;
    var xc:int = c[3]:int;
    var yc:int = xc[1072]:ubyte;
    var zc:int = 24;
    var ad:int = yc << zc;
    var bd:int = ad >> zc;
    var cd:int = 6;
    var dd:int = bd ^ cd;//i %3 =1: flag[i]^= 6
    xc[1072]:byte = dd;
    goto B_i;
    label B_j:
    var ed:int = c[3]:int;
    var fd:int = ed[1072]:ubyte;
    var gd:int = 24;
    var hd:int = fd << gd;
    var id:int = hd >> gd;
    var jd:int = 5;
    var kd:int = id ^ jd;  //i %3 =2: flag[i] ^=5
    ed[1072]:byte = kd;
    label B_i:
    label B_g:
    var ld:int = c[3]:int;
    var md:int = 1;
    var nd:int = ld + md;
    c[3]:int = nd;
    continue L_b;
  }
```

- Stage 2: After evaluating the element's value of flag, ***check_flag()*** function will swap the ***flag[i]*** and ***flag[i + 1].***

```
label B_a:
  var od:int = 0;
  c[1]:int = od;
  loop L_l {
    var pd:int = c[1]:int;  //name c[1] is j
    var qd:int = c[3]:int;  //c[3]: i - now, i is len(flag array)
    var rd:int = pd;
    var sd:int = qd;
    var td:int = rd < sd;
    var ud:int = 1;
```

```
    var vd:int = td & ud;
    if (eqz(vd)) goto B_k;  //i <= j break
    var wd:int = c[1]:int;
    var xd:int = 2;
    var yd:int = wd % xd;
    if (yd) goto B_m;      //if j %2 != 0 then continue loop
    var zd:int = c[1]:int;
    var ae:int = 1;
    var be:int = zd + ae;
    var ce:int = c[3]:int;
    var de:int = be;
    var ee:int = ce;
    var fe:int = de < ee;
    var ge:int = 1;
    var he:int = fe & ge;
    if (eqz(he)) goto B_m;  //if j + 1 >= i then break loop
    var ie:ubyte_ptr = c[1]:int;
    var je:int = ie[1072];  //flag[j]
    c[11]:byte = je;        //tmp = flag[j]
    var ke:int = c[1]:int;
    var le:int = 1;
    var me:ubyte_ptr = ke + le;  //j + 1
    var ne:int = me[1072];
    var oe:byte_ptr = c[1]:int;
    oe[1072] = ne;      //flag[j] = flag[j + 1]
    var pe:int = c[11]:ubyte;
    var qe:int = c[1]:int;
    var re:int = 1;
    var se:byte_ptr = qe + re;
    se[1072] = pe;      //flag[j + 1] = tmp
    label B_m:
    var te:int = c[1]:int;
    var ue:int = 1;
    var ve:int = te + ue;
    c[1]:int = ve;
    continue L_l;
  }
```

- Stage3: Finally, the **check_flag** function will compare **resulted flag array** to **target array** which is available in decompiled code of this challenge:

```
data d_a(offset: 1024) =
  "\18j|a\118i7F[#\06fJV:\0d\1c\12/dd\11Vu\0fn\1b\068\07E\10/o?\13\02+\09"
  "^\00\00";
```

```
//check flag with target
  label B_k:
  var we:int = 0;
  var xe:int = 1072;
  var ye:int = 1024;
  var ze:int = strcmp(ye, xe);
  var af:int = ze;
  var bf:int = we;
  var cf:int = af != bf;
  var df:int = -1;
  var ef:int = cf ^ df;
  var ff:int = 1;
  var gf:int = ef & ff;
  var hf:int = 16;
  var if:int = c + hf;
  g_a = if;
  return gf;
```