easy_wasm

> 新生赛一上来就出wasm逆向，真的好吗。。
>
> 参考链接：
>
> https://xz.aliyun.com/t/5170
> https://www.anquanke.com/post/id/179556
> https://xz.aliyun.com/t/2854

附件下载解压，得到的是这三个文件：

名称	修改日期	类型	大小
index.html	2021/9/29 21:12	Firefox HTML D...	1 KB
main.wasm	2021/9/30 11:01	WASM 文件	2,259 KB
wasm_exec.js	2021/9/29 18:06	JavaScript 源文件	15 KB

对wasm文件进行重构，得到可逆向的程序文件

逆向出.c文件（可读性较差）

对于wasm文件，采用wabt来逆向，得到一份C文件：

```
wasm2c main.wasm -o main.c
```

这是摘要：

```
/* Automically generated by wasm2c */
#include <math.h>
#include <string.h>

#include "main.h"
#define UNLIKELY(x) __builtin_expect(!!(x), 0)
#define LIKELY(x) __builtin_expect(!!(x), 1)

#define TRAP(x) (wasm_rt_trap(WASM_RT_TRAP_##x), 0)

#define FUNC_PROLOGUE                                            \
  if (++wasm_rt_call_stack_depth > WASM_RT_MAX_CALL_STACK_DEPTH) \
    TRAP(EXHAUSTION)

#define FUNC_EPILOGUE --wasm_rt_call_stack_depth

#define UNREACHABLE TRAP(UNREACHABLE)

#define CALL_INDIRECT(table, t, ft, x, ...)          \
```

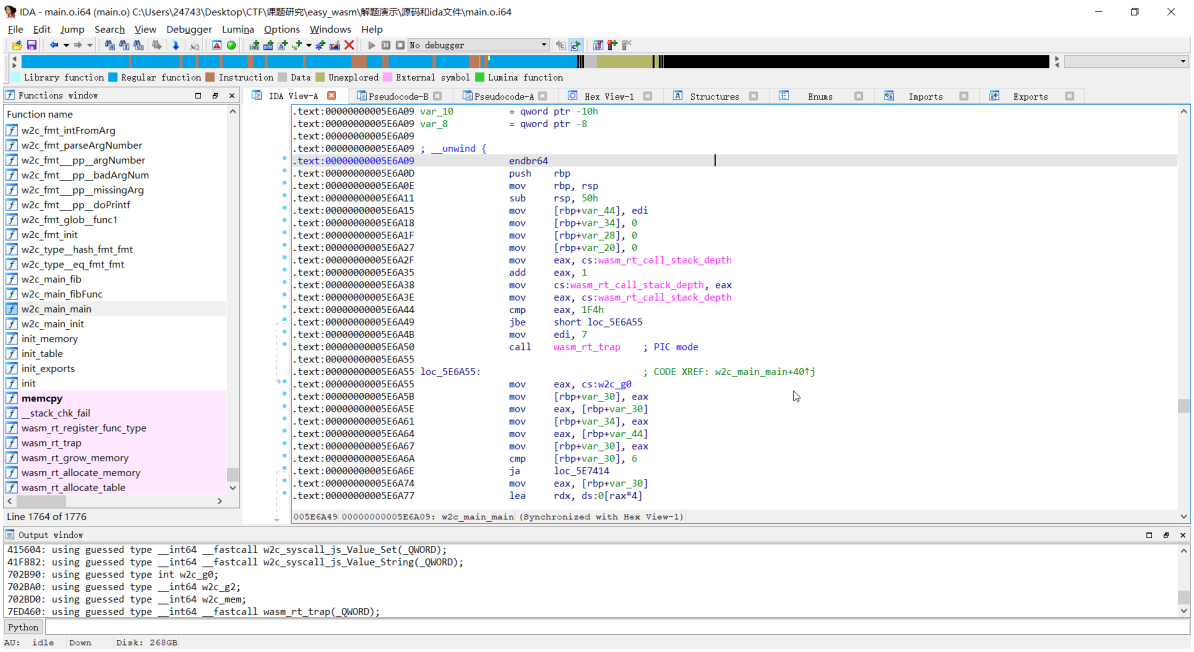
```
  (LIKELY((x) < table.size && table.data[x].func &&  \
          table.data[x].func_type == func_types[ft]) \
       ? ((t)table.data[x].func)(__VA_ARGS__)        \
       : TRAP(CALL_INDIRECT))

#if WASM_RT_MEMCHECK_SIGNAL_HANDLER
#define MEMCHECK(mem, a, t)
#else
#define MEMCHECK(mem, a, t)  \
  if (UNLIKELY((a) + sizeof(t) > mem->size)) TRAP(OOB)
#endif

#define DEFINE_LOAD(name, t1, t2, t3)                         \
  static inline t3 name(wasm_rt_memory_t* mem, u64 addr) {   \
    MEMCHECK(mem, addr, t1);                                  \
    t1 result;                                                \
    __builtin_memcpy(&result, &mem->data[addr], sizeof(t1)); \
    return (t3)(t2)result;                                    \
  }

#define DEFINE_STORE(name, t1, t2)                                   \
  static inline void name(wasm_rt_memory_t* mem, u64 addr, t2 value) { \
    MEMCHECK(mem, addr, t1);                                         \
    t1 wrapped = (t1)value;                                          \
    __builtin_memcpy(&mem->data[addr], &wrapped, sizeof(t1));          \
  }

DEFINE_LOAD(i32_load, u32, u32, u32);
DEFINE_LOAD(i64_load, u64, u64, u64);
DEFINE_LOAD(f32_load, f32, f32, f32);
DEFINE_LOAD(f64_load, f64, f64, f64);
DEFINE_LOAD(i32_load8_s, s8, s32, u32);
DEFINE_LOAD(i64_load8_s, s8, s64, u64);
DEFINE_LOAD(i32_load8_u, u8, u32, u32);
DEFINE_LOAD(i64_load8_u, u8, u64, u64);
DEFINE_LOAD(i32_load16_s, s16, s32, u32);
DEFINE_LOAD(i64_load16_s, s16, s64, u64);
DEFINE_LOAD(i32_load16_u, u16, u32, u32);
DEFINE_LOAD(i64_load16_u, u16, u64, u64);
DEFINE_LOAD(i64_load32_s, s32, s64, u64);
DEFINE_LOAD(i64_load32_u, u32, u64, u64);
DEFINE_STORE(i32_store, u32, u32);
DEFINE_STORE(i64_store, u64, u64);
DEFINE_STORE(f32_store, f32, f32);
DEFINE_STORE(f64_store, f64, f64);
DEFINE_STORE(i32_store8, u8, u32);
DEFINE_STORE(i32_store16, u16, u32);
DEFINE_STORE(i64_store8, u8, u64);
DEFINE_STORE(i64_store16, u16, u64);
DEFINE_STORE(i64_store32, u32, u64);
```

对逆向出的.c文件，编译成.o文件，使其可以用IDA反汇编出C代码

可以看到，这份逆向出的代码，可读性非常差，于是考虑编译成.o链接文件，便可以在IDA里面直接分析：

```
gcc -c main.c -o main.o
```

得到的.o文件，便可以直接用IDA进行反汇编分析：



对程序结构进行分析

定位wasm的函数

首先，定位`w2c_main_main`主函数：

```
__int64 __fastcall w2c_main_main(unsigned int a1)
{
  if ( ++wasm_rt_call_stack_depth > 0x1F4u )
    wasm_rt_trap(7LL);
  if ( a1 <= 6 )
    __asm { jmp     rax }
  wasm_rt_trap(5LL);
  --wasm_rt_call_stack_depth;
  return 1LL;
}
```

还有`w2c_main_fib`和`w2c_main_fibFunc`两个函数，由于`w2c_main_fibFunc`使用了函数`w2c_mainfib`，所以猜测这里处理事件的函数为`w2c_main_fib`：

```
__int64 __fastcall w2c_main_fibFunc(unsigned int a1)
{
  int v1; // eax
  __int64 v2; // rax
  int v3; // eax
  __int64 v4; // rax
  int v5; // eax
```

```
__int64 v6; // rax
int v7; // eax
int v8; // eax
__int64 v9; // rax
int v10; // eax
unsigned int v13; // [rsp+14h] [rbp-2Ch]
unsigned int v14; // [rsp+14h] [rbp-2Ch]
unsigned int v15; // [rsp+14h] [rbp-2Ch]
unsigned int v16; // [rsp+14h] [rbp-2Ch]
unsigned int v17; // [rsp+14h] [rbp-2Ch]
unsigned int v18; // [rsp+14h] [rbp-2Ch]
__int64 v19; // [rsp+20h] [rbp-20h]
__int64 v20; // [rsp+28h] [rbp-18h]
__int64 v21; // [rsp+28h] [rbp-18h]

if ( ++wasm_rt_call_stack_depth > 0x1F4u )
  wasm_rt_trap(7LL);
v13 = w2c_g0;
while ( a1 <= 8 )
{
  if ( v13 <= (unsigned int)i32_load(&w2c_mem, (unsigned int)w2c_g2 + 16LL) )
  {
    w2c_g0 = v13 - 8;
    i64_store(&w2c_mem, v13 - 8, 0x16C80000LL);
    v1 = w2c_runtime_morestack_noctxt(0LL);
    v13 = w2c_g0;
    if ( v1 )
      goto LABEL_16;
  }
  v14 = v13 - 40;
  w2c_g0 = v14;
  v2 = i64_load(&w2c_mem, 0x15DD58LL);
  i64_store(&w2c_mem, v14, v2);
  i64_store(&w2c_mem, v14 + 8LL, &loc_342F7);
  i64_store(&w2c_mem, v14 + 16LL, 5LL);
  w2c_g0 = v14 - 8;
  i64_store(&w2c_mem, v14 - 8, 0x16C80002LL);
  v3 = w2c_syscall_js_Value_Get(0LL);
  v15 = w2c_g0;
  if ( v3 )
    goto LABEL_16;
  v4 = i64_load(&w2c_mem, (unsigned int)w2c_g0 + 24LL);
  i64_store(&w2c_mem, v15, v4);
  w2c_g0 = v15 - 8;
  i64_store(&w2c_mem, v15 - 8, 0x16C80003LL);
  v5 = w2c_syscall_js_Value_String(0LL);
  v16 = w2c_g0;
  if ( v5 )
    goto LABEL_16;
  v20 = i64_load(&w2c_mem, (unsigned int)w2c_g0 + 16LL);
  v6 = i64_load(&w2c_mem, v16 + 8LL);
  i64_store(&w2c_mem, v16, v6);
  i64_store(&w2c_mem, v16 + 8LL, v20);
  w2c_g0 = v16 - 8;
  i64_store(&w2c_mem, v16 - 8, 0x16C80004LL);
  v7 = w2c_strconv_Atoi(0LL);
  v13 = w2c_g0;
  if ( v7 )
```

```
      goto LABEL_16;
    v21 = i64_load(&w2c_mem, (unsigned int)w2c_g0 + 24LL);
    v19 = i64_load(&w2c_mem, v13 + 16LL);
    if ( v21 )
    {
      i64_store(&w2c_mem, v13 + 80, 0LL);
      i64_store(&w2c_mem, v13 + 80 + 8LL, 0LL);
      v17 = v13 + 48;
      w2c_g0 = v17;
      i64_store(&w2c_mem, v17, v19);
      w2c_g0 = v17 - 8;
      i64_store(&w2c_mem, v17 - 8, 0x16C80007LL);
      v8 = w2c_main_fib(0);
      v18 = w2c_g0;
      if ( v8 )
        goto LABEL_16;
      v9 = i64_load(&w2c_mem, 0x15DD40LL);
      i64_store(&w2c_mem, v18, v9);
      i64_store(&w2c_mem, v18 + 8LL, 0x34C24LL);
      i64_store(&w2c_mem, v18 + 16LL, 9LL);
      i64_store(&w2c_mem, v18 + 24LL, 0x154E0LL);
      i64_store(&w2c_mem, v18 + 32LL, 0x49B60LL);
      w2c_g0 = v18 - 8;
      i64_store(&w2c_mem, v18 - 8, 0x16C80008LL);
      v10 = w2c_syscall_js_Value_Set(0LL);
      v13 = w2c_g0;
      if ( v10 )
        goto LABEL_16;
      a1 = 5;
    }
    else
    {
      a1 = 6;
    }
  }
  wasm_rt_trap(5LL);
LABEL_16:
  --wasm_rt_call_stack_depth;
  return 1LL;
}
```

```
__int64 __fastcall w2c_main_fib(unsigned int a1)
{
  int v1; // eax
  __int64 v2; // rax
  int v3; // eax
  int v4; // eax
  int v5; // eax
  __int64 v6; // rax
  __int64 v7; // rax
  int v8; // eax
  __int64 v9; // rax
  int v10; // eax
  __int64 v11; // rax
  __int64 v12; // rax
  int v13; // eax
  unsigned int v16; // [rsp+18h] [rbp-88h]
```

```
  unsigned int v17; // [rsp+18h] [rbp-88h]
  unsigned int v18; // [rsp+18h] [rbp-88h]
  unsigned int v19; // [rsp+18h] [rbp-88h]
  unsigned int v20; // [rsp+18h] [rbp-88h]
  int v21; // [rsp+18h] [rbp-88h]
  unsigned int v22; // [rsp+1Ch] [rbp-84h]
  __int64 v23; // [rsp+30h] [rbp-70h]
  __int64 v24; // [rsp+30h] [rbp-70h]
  __int64 v25; // [rsp+30h] [rbp-70h]
  __int64 v26; // [rsp+30h] [rbp-70h]
  __int64 v27; // [rsp+30h] [rbp-70h]
  __int64 v28; // [rsp+30h] [rbp-70h]
  __int64 v29; // [rsp+30h] [rbp-70h]
  __int64 v30; // [rsp+38h] [rbp-68h]
  __int64 v31; // [rsp+38h] [rbp-68h]
  __int64 v32; // [rsp+38h] [rbp-68h]
  int v33; // [rsp+38h] [rbp-68h]
  unsigned __int64 v34; // [rsp+40h] [rbp-60h]
  __int64 v35; // [rsp+48h] [rbp-58h]
  unsigned __int64 v36; // [rsp+50h] [rbp-50h]
  int v37; // [rsp+58h] [rbp-48h]
  int v38; // [rsp+68h] [rbp-38h]
  __int64 v39; // [rsp+68h] [rbp-38h]
  __int64 v40; // [rsp+70h] [rbp-30h]
  __int64 v41; // [rsp+70h] [rbp-30h]
  unsigned __int64 v42; // [rsp+70h] [rbp-30h]
  int v43; // [rsp+78h] [rbp-28h]

  LODWORD(v36) = 0;
  v37 = 0;
  if ( ++wasm_rt_call_stack_depth > 0x1F4u )
    wasm_rt_trap(7LL);
  v16 = w2c_g0;
  while ( 1 )
  {
    while ( 1 )
    {
      while ( 1 )                                   // 循环次数为v4数组的长度
      {
        while ( 1 )                                 // a
                                                    //
        {
          while ( 1 )
          {
            while ( 1 )
            {
              if ( a1 > 0x2C )
              {
                wasm_rt_trap(5LL);
                goto LABEL_27;
              }
              if ( v16 <= (unsigned int)i32_load(&w2c_mem, (unsigned int)w2c_g2
+ 16LL) + 192 )
              {
                w2c_g0 = v16 - 8;
                i64_store(&w2c_mem, v16 - 8, 382140416LL);
                v1 = w2c_runtime_morestack_noctxt(0LL);
                v16 = w2c_g0;
```

```
      if ( v1 )
        goto LABEL_27;
    }
    v17 = v16 - 320;
    w2c_g0 = v17;
    v2 = i64_load(&w2c_mem, v17 + 328LL);
    i64_store(&w2c_mem, v17, v2);
    i64_store(&w2c_mem, v17 + 8LL, 10LL);
    w2c_g0 = v17 - 8;
    i64_store(&w2c_mem, v17 - 8, 382140420LL);
    v3 = w2c_strconv_FormatInt(0LL);
    v18 = w2c_g0;
    if ( v3 )
      goto LABEL_27;
    v23 = i64_load(&w2c_mem, (unsigned int)w2c_g0 + 24LL);
    i64_store(&w2c_mem, v18 + 112LL, v23);
    v30 = i64_load(&w2c_mem, v18 + 16LL);
    i64_store(&w2c_mem, v18 + 184LL, v30);
    i64_store(&w2c_mem, v18, v18 + 136LL);
    i64_store(&w2c_mem, v18 + 8LL, v30);
    i64_store(&w2c_mem, v18 + 16LL, v23);
    w2c_g0 = v18 - 8;
    i64_store(&w2c_mem, v18 - 8, 0x16C70006LL);
    v4 = w2c_runtime_stringtoslicebyte(0);// 转换为byte数组, v4
    v16 = w2c_g0;
    if ( v4 )
      goto LABEL_27;
    v24 = i64_load(&w2c_mem, (unsigned int)w2c_g0 + 32LL);
    v31 = i64_load(&w2c_mem, v16 + 24LL);
    if ( v24 )
      break;
    a1 = 43;
  }
  i64_store(&w2c_mem, v16 + 104LL, v24);
  i64_store(&w2c_mem, v16 + 176LL, v31);
  v25 = i64_load8_u(&w2c_mem, (unsigned int)v31) - 48;// v25 = v4[0] -
48
  i64_store32(&w2c_mem, v16 + 80LL, v25);
  i64_store32(&w2c_mem, v16, v25);
  w2c_g0 = v16 - 8;
  i64_store(&w2c_mem, v16 - 8, 0x16C70008LL);
  v5 = w2c_runtime_convT32(0LL);
  v19 = w2c_g0;
  if ( v5 )
    goto LABEL_27;
  v26 = i64_load(&w2c_mem, (unsigned int)w2c_g0 + 8LL);
  i64_store(&w2c_mem, v19 + 192, 0LL);
  i64_store(&w2c_mem, v19 + 192 + 8LL, 0LL);
  i64_store(&w2c_mem, v19 + 192, &loc_155E0);
  i64_store(&w2c_mem, v19 + 192 + 8LL, v26);
  v27 = i64_load(&w2c_mem, 1328488LL);
  i64_store(&w2c_mem, v19, 307520LL);
  i64_store(&w2c_mem, v19 + 8LL, v27);
  i64_store(&w2c_mem, v19 + 16LL, &loc_3403D);
  i64_store(&w2c_mem, v19 + 24LL, 3LL);
  i64_store(&w2c_mem, v19 + 32LL, v19 + 192LL);
  i64_store(&w2c_mem, v19 + 40LL, 1LL);
  i64_store(&w2c_mem, v19 + 48LL, 1LL);
```

```
            w2c_g0 = v19 - 8;
            i64_store(&w2c_mem, v19 - 8, 0x16C7000BLL);
            w2c_fmt_Fprintf(0LL);
            v20 = w2c_g0;
            i64_load32_u(&w2c_mem, (unsigned int)w2c_g0 + 80LL);
            v28 = i64_load(&w2c_mem, v20 + 96LL) + 1;
            v40 = 12 * i64_load32_u(&w2c_mem, v20 + 80LL);// v40 = v25 * 12
            v38 = i64_load(&w2c_mem, v20 + 176LL);
            v43 = i64_load(&w2c_mem, v20 + 96LL);
            v32 = i64_load8_u(&w2c_mem, (unsigned int)(v43 + v38)) + v40 - 48;//
v32 = v40 + v4[1] - 48
            i64_store(&w2c_mem, v20 + 96LL, v28);
            i64_store32(&w2c_mem, v20 + 80LL, v32);
            v6 = i64_load(&w2c_mem, v20 + 184LL);
            i64_store(&w2c_mem, v20, v6);
            v7 = i64_load(&w2c_mem, v20 + 112LL);
            i64_store(&w2c_mem, v20 + 8LL, v7);
            w2c_g0 = v20 - 8;
            i64_store(&w2c_mem, v20 - 8, 0x16C7000FLL);
            v8 = w2c_runtime_countrunes(0LL);
            v16 = w2c_g0;
            if ( v8 )
              goto LABEL_27;
            v41 = i64_load(&w2c_mem, (unsigned int)w2c_g0 + 96LL);
            if ( v41 < i64_load(&w2c_mem, v16 + 16LL) )
              break;                                // 循环次数为v4数组的长度
            a1 = 17;
          }
          v42 = i64_load(&w2c_mem, v16 + 96LL);
          if ( v42 < i64_load(&w2c_mem, v16 + 0x68LL) )
            break;
          a1 = 41;
        }
        v9 = i64_load(&w2c_mem, 0x15DEE8LL);
        i64_store(&w2c_mem, v16, v9);
        i64_store(&w2c_mem, v16 + 8LL, 0x342ACLL);
        i64_store(&w2c_mem, v16 + 16LL, 5LL);
        w2c_g0 = v16 - 8;
        i64_store(&w2c_mem, v16 - 8, 0x16C70012LL);
        v10 = w2c_syscall_js_Value_Get(0LL);
        v16 = w2c_g0;
        if ( v10 )
          goto LABEL_27;
        v29 = i64_load(&w2c_mem, (unsigned int)w2c_g0 + 24LL);
        i64_store32(&w2c_mem, v16 + 84, 0LL);
        v11 = i64_load(&w2c_mem, v16 + 328LL);
        i64_store8(&w2c_mem, v16 + 87, v11);
        v34 = (unsigned int)i64_load(&w2c_mem, v16 + 328LL);
        i64_store8(&w2c_mem, v16 + 86, v34 >> 8);// 对输入数据进行切片
        i64_store8(&w2c_mem, v16 + 85, v34 >> 16);
        i64_store8(&w2c_mem, v16 + 84, v34 >> 24);
        v35 = i64_load32_u(&w2c_mem, (unsigned int)(4 * v36 + v37)) ^ 0xFFFFFF;
        v36 = (unsigned __int8)(i64_load8_u(&w2c_mem, v16 + 85) ^ v35);
        v37 = i64_load(&w2c_mem, 1273040LL);
        if ( v36 < i64_load(&w2c_mem, 1273048LL) )
          break;
        a1 = 39;
      }
```

```
      v12 = w2c_runtime_wasmTruncS(fabs((double)~(_DWORD)v35));
      v16 = w2c_g0;
      v33 = v12;
      i64_store(&w2c_mem, (unsigned int)w2c_g0 + 128LL, v12);
      if ( (unsigned int)i64_load32_u(&w2c_mem, v16 + 80LL) == 0x6C1540EELL )
        break;
      a1 = 24;
    }
    if ( v33 == 1963701148LL )
      break;
    a1 = 24;
  }
  i64_store(&w2c_mem, v16 + 208, 0LL);
  i64_store(&w2c_mem, v16 + 208 + 8LL, 0LL);
  i64_store(&w2c_mem, v16 + 208, 87264LL);
  i64_store(&w2c_mem, v16 + 208 + 8LL, 301904LL);
  i64_store(&w2c_mem, v16, v29);
  i64_store(&w2c_mem, v16 + 8LL, v16 + 208LL);
  i64_store(&w2c_mem, v16 + 16LL, 1LL);
  i64_store(&w2c_mem, v16 + 24LL, 1LL);
  w2c_g0 = v16 - 8;
  i64_store(&w2c_mem, v16 - 8, 382140441LL);
  v13 = w2c_syscall_js_Value_Invoke(0LL);
  v21 = w2c_g0;
  if ( !v13 )
  {
    v39 = i64_load(&w2c_mem, (unsigned int)w2c_g0 + 128LL);
    i64_store32(&w2c_mem, (unsigned int)(v21 + 336), v39);
    w2c_g0 = v21 + 328;
    v22 = 0;
    goto LABEL_28;
  }
LABEL_27:
  v22 = 1;
LABEL_28:
  --wasm_rt_call_stack_depth;
  return v22;
}
```

找到编码算法的特征，并逆向解出数据

在`w2c_main_fib`函数中，看到了这个语句：

```
if ( v33 == 1963701148LL )
```

根据算法特征，可以分析出这个是CRC32算法。于是结合代码逻辑，编写逆向解密脚本：

```
num = ''
f = 0x6C1540EE
while 1:
    f += 48
    num += str(f%12)[0:1]
    f -= f%12 + 48
    f /= 12
    if f < 10:
      num += str(f)[0:1]
      break
print(num[::-1])
# 427346092
```

搭建本地环境，运行wasm文件

利用Python搭建本地http环境

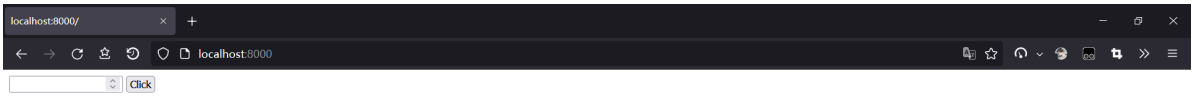
在存放index.html和main.wasm的文件夹下，命令行执行（前提是已安装Python3）：

```
python -m http.server 8000
```

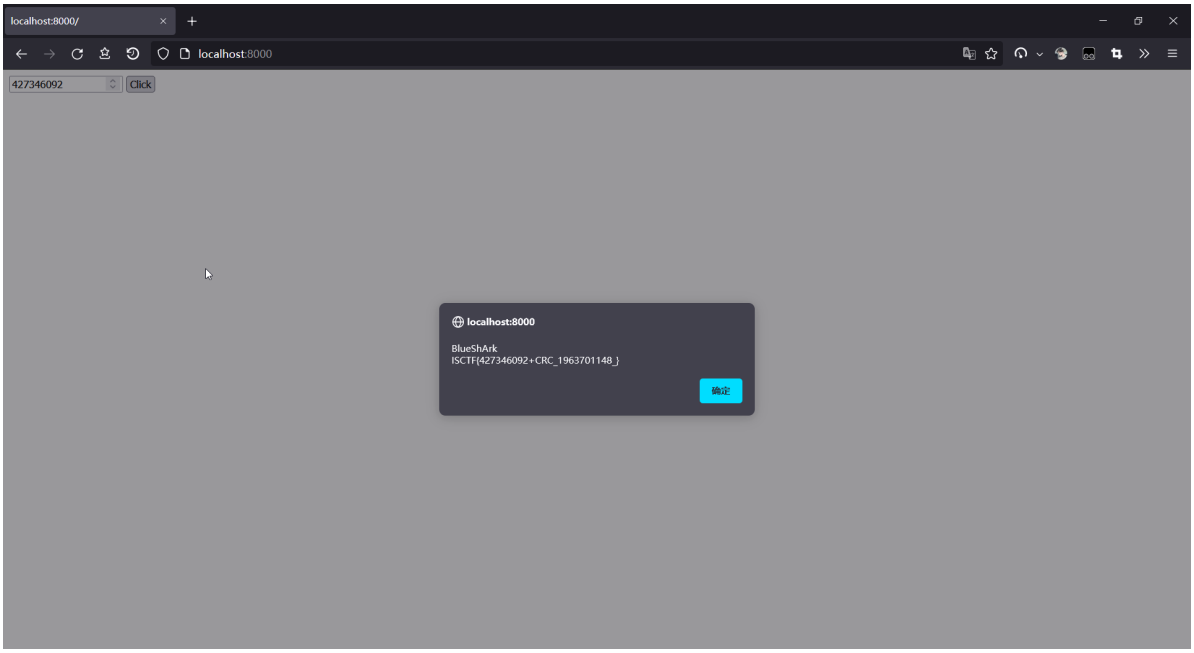
这样子就可以在本地的8000端口打开本地环境，如果端口冲突的话，可以考虑换一个端口。

本地验证数据正确性

打开：localhost:8000：



输入`427346092`验证计算结果是否正确：



就可以成功得到flag:~~ISCTF{427346092+CRC_1963701148_}~~