OneETH合约问题列表

SECBIT

Dec 18, 2018

问题列表

风险类型	描述	风险级别	问题状态
实现漏洞	1.1 `isHuman()` 修饰函数可以被绕过	中	已修复
代码优化	1.2 `withdraw()` 函数中重复判断合约调用者身份	信息	已修复
代码优化	1.3 `withdrawAll()` 函数中存在代码冗余	信息	已修复
实现漏洞	1.4 龙虎榜内选手名次变动时，榜单末位可能置零，存在多发奖励的风险	高	已修复
逻辑漏洞	1.5 高级经理人分红结算方式与游戏规则不符	低	已修复
逻辑漏洞	1.6 经理的判定方式与游戏规则不符	低	已修复
实现漏洞	1.7 每轮游戏结束时，高级经理人未分配奖金留在合约中无法提取	中	已修复
逻辑漏洞	1.8 当玩家推荐三个经理后直推30ETH无法成为高级经理人	中	已修复
实现漏洞	1.9 `recordDynamicProfits()`函数中，当系统为推荐人时只获取一层奖金，其余奖金没有流向也无法提取	高	已修复
实现漏洞	1.10 未确定推荐奖金记录时间超过12小时后，时间记录可能被覆盖，系统无法获得对应的超时奖金	中	已修复
实现漏洞	1.11 `getUnProfits()` 函数中，推荐人应得的未确定推荐奖金可能计算不正确	中	已修复
实现漏洞	1.12 未确定推荐奖金的分配由被推荐玩家参与游戏被动触发，游戏结束后存在未分配的未确定推荐奖金无法提取	中	已修复
代码优化	1.13 玩家入场时计算静态收益判断是否有玩家出局，此过程存在不必要计算	信息	已修复
逻辑漏洞	1.14 第201个玩家入场时的静态收益分配与游戏规则不符	低	已修复
实现漏洞	2.1 `tryGetRefererByName()` 函数可以绑定任意推荐人关系	中	已修复
实现漏洞	3.1 `winGame()` 函数中当`data.winNumber-i = left` 且 `data.winNumber+i < right` 时，程序进入死循环	高	已修复

风险类型	描述	风险级别	问题状态
代码优化	3.2 `winGame()` 函数中left与right值可能被重置，产生大量无效的计算	中	已修复
代码优化	3.3 `winGame()` 函数中存在大量重复计算与 sload 调用，产生额外的gas消耗	低	已修复
代码优化	A.1 代码中提示信息存在语法错误或表达不当	信息	已修复

合约问题

OneETH.sol

1.1 `isHuman()` 修饰函数可以被绕过

- 风险级别：**中**
- 问题类型：实现漏洞
- 问题描述：

  合约中通过判断msg.sender的codesize是否为0来判断msg.sender是否为合约，codesize为0则认为msg.sender为普通账户。但是存在一种特殊情况，合约部署时会调用构造函数，构造函数调用完毕前，代码没有部署到区块链上，合约地址对应的codesize为0，所以如果攻击者在合约的构造函数中调用游戏合约，则能绕过该限制。
- 影响结果：

  防止合约调用合约失败。
- 规避方案：

  建议通过判断tx.origin是否等于msg.sender达到效果。

```
modifier isHuman() {
  require(tx.origin == msg.sender);
  _;
}
```

- 问题状态：

  已按照修改建议修改。

1.2 `withdraw()` 函数中重复判断合约调用者身份

- 风险级别：**信息**
- 问题类型：代码优化
- 问题描述：

  `withdraw()` 函数中 `require(!isContract(msg.sender), "you aren't human");` 与 `isHuman()` 修饰函数具有相同功能，重复判断。
- 影响结果：

  重复判断，增加gas消耗。
- 规避方案：

  删除 `require(!isContract(msg.sender), "you aren't human");` 语句或者 `isHuman()` 修饰函数。

- 问题状态：

  已按照修改建议修改。

1.3 `withdrawAll()` 函数中存在代码冗余

- 风险级别：**信息**
- 问题类型：代码优化
- 问题描述：

  函数先计算`guess.getProfits()`判断用户收益，然后执行`guess.withdrawProfits()`函数进行提款操作，后者包含了前者的计算内容。并且当用户存在未提款的当天收益时，`guess.getProfits()`函数计算返回大于0的收益，然而在`guess.withdrawProfits()`函数中不存在可以提取的收益，因而产生两次函数计算的gas消耗。
- 影响结果：

  代码冗余，增加gas消耗。
- 规避方案：

  建议删除冗余代码，优化gas消耗。
- 问题状态：

  已按照修改建议修改。

1.4 龙虎榜内选手名次变动时，榜单末位可能置零，存在多发奖励的风险

- 风险级别：高
- 问题类型：实现漏洞
- 问题描述：

  `tryUpdateDT()` 函数用于处理龙虎榜的变动，当有入榜选手名次变动时，榜单末位可能置零，此后当有新的直推数进入榜单时，对应的并列玩家数会置为1。由于龙虎榜发放金额以参与游戏的总ETH数按比例发放，并列玩家不准确会造成大量超额的奖金发放，影响玩家的利益。例如，某时刻龙虎榜前五名为(201,1)，(200,1)，(100,14)，(99,20)，(98,25)。 当榜单第二名玩家直推一个ETH与第一名并列，榜单更新过程中，第五名会更新为(0,0)。榜单更新为(201,2)，(100,14)，(99,20)，(98,25)，(0,0)。此后，当有新的玩家进入榜单时（直推数不限），合约统计的并列玩家数为1，实际上可能存在很多并列的玩家。由于存在前几名胶着更替名次的情况，第五名会不断被更新为(0,0)，新入榜的直推数对应的并列玩家为1名。同样的，当并列玩家发生名次变更，榜单第五名也可能被挤出榜单，该直推数重新进入榜单时并列玩家数会被置为1。 按照游戏规则，龙虎奖第五名获得奖金占当天参与游戏总金额的0.0015倍(后期为0.004倍)。假设游戏终止时共10000个ETH入场，以第五名实际并列人数为20，统计并列人数为1计算，游戏多支出的龙虎奖奖金为(20-1) * 0.004 * 10000 = 760ETH。

  产生该漏洞的代码为：

```
if (upgradeFromBackwards == true ) {
    if (i < 4 && rInfos_[rid].dts[day][i + 1].amount != 0) {
        rInfos_[rid].dts[day][i + 1].amount = rInfos_[rid].dts[day][i +
1].amount.sub(1);
```

```
            // No user at this position, set length to zero.
            if (rInfos_[rid].dts[day][i + 1].amount == 0) {
                rInfos_[rid].dts[day][i + 1].children = 0;

                // Move backwards forward.
                for (uint k = i + 1; k < 4; k++) {
                    rInfos_[rid].dts[day][k].amount = rInfos_[rid].dts[day][k
+ 1].amount;
                    rInfos_[rid].dts[day][k].children = rInfos_[rid].dts[day]
[k + 1].children;
                }

                // Set fifth info to 0, 0.
                rInfos_[rid].dts[day][4].amount = 0;
                rInfos_[rid].dts[day][4].children = 0;
            }
        }
    }
```

- 影响结果:

  龙虎榜奖金计算不正确，导致多发金额，严重影响游戏生态与玩家利益。

- 规避方案:

  增加存储榜单的数组长度，保证不存在榜单第五名置零，以及入榜直推数重新统计的问题。

- 问题状态:

  已修改龙虎榜奖金分配规则，问题已修复。

1.5 高级经理人分红结算方式与游戏规则不符

- 风险级别：**低**
- 问题类型：逻辑漏洞
- 问题描述:

  游戏规则中，高级经理人分红是每10000ETH结算一次，代码实现中表达的是每10000个参与人次进行一次结算。由于合约中的参与游戏的金额可以通过`changeBetNum_()`函数更改，因此游戏规则与代码实现可能出现不符合的情况。

```
if (rInfos_[rid].seniorManagers.length == 0 && rInfos_[rid].eSerial %
10000 == 0) {
    foundationProfits =
foundationProfits.add(rInfos_[rid].eJoin.mul(100));
}
```

- 影响结果:

  可能使玩家对游戏规则产生误解。

- 规避方案:

  建议修改代码或者游戏规则，达到一致。

- 问题状态：

  已确认修改游戏规则中的描述，问题已修复。

1.6 经理的判定方式与游戏规则不符

- 风险级别：**低**
- 问题类型：逻辑漏洞
- 问题描述：

  游戏规则中，每个玩家直推30个ETH可以成为经理。代码实现中，用户是否达到经理的要求在`recordDynamicProfits()`函数中通过`plyrRnds_[rid][tmpRPid].directChildren.length`判定，表达的是玩家直推的游戏人次，并不是直推的ETH数。由于合约中的参与游戏的金额可以通过`changeBetNum_()`函数更改，因此游戏规则与代码实现可能出现不符合的情况。

```
if (floor == 1 || plyrRnds_[rid][tmpRPid].directChildren.length >=
childrenClaim) {
    // Updates determine profits of ancestor.
    plyrRnds_[rid][tmpRPid].dynamicDProfits = plyrRnds_[rid]
[tmpRPid].dynamicDProfits.add(profits);
} else {
    // Try to insert it into undetermined profits.
    foundationProfits = foundationProfits.add(tryInsertUnDProfits(rid,
tmpRPid, floor, time));
}
```

- 影响结果：

  可能使玩家对游戏规则产生误解。

- 规避方案：

  建议修改代码或者游戏规则，达到一致。

- 问题状态：

  已确认修改游戏规则中的描述，问题已修复。

1.7 每轮游戏结束时，高级经理人未分配奖金留在合约中无法提取

- 风险级别：**中**
- 问题类型：实现漏洞
- 问题描述：

  游戏规则中，每10000个ETH对高级经理人分配一次奖金，金额为100ETH。当参与游戏的ETH数量不为10000的整数倍时，多出金额的1%会留在合约中无法提取。

- 影响结果：

  存在金额在游戏中无法提取，影响游戏团队与玩家的利益。

- 规避方案：

建议在每轮游戏结束时，对未分配的高级经理人奖金进行分配。

- 问题状态：

  已在代码中增加`calUndeservedSeniorP()`函数用于处理未分配奖金，问题已修复。

1.8 当玩家推荐三个经理后直推30ETH无法成为高级经理人

- 风险级别：**中**
- 问题类型：逻辑漏洞
- 问题描述：

  游戏规则中，当一个玩家同时满足推荐直推三个经理以及直推30ETH时成为高级经理人。代码实现中，只当玩家成为经理时，判断其推荐人是否符合成为高级经理人的条件。若玩家直推三个经理时没有直推30ETH，当该玩家直推30ETH时，无法成为高级经理人。

  `updatePGInfo()`函数中，问题代码如下：

```
uint referChildren = plyrRnds_[rid][refererPid].directChildren.length;
if (referChildren == 30) {
    // Gets ancestor of referer.
    uint ancestorPid = players_[refererPid].aff;
    if (ancestorPid != 0) {
        plyrRnds_[rid][ancestorPid].ownManagers++;
        // Updates ancestor role to senior manager if he has 3 managers.
        if (plyrRnds_[rid][ancestorPid].ownManagers == 3 &&
            plyrRnds_[rid][ancestorPid].directChildren.length >= 30
        ) {
            plyrRnds_[rid][ancestorPid].timeBeSenManager = time;
            rInfos_[rid].seniorManagers.push(ancestorPid);

            emit LogSeniorManager(rid, ancestorPid, time);
        }
    }
}
```

- 影响结果：

  与游戏规则不符，影响玩家利益。

- 规避方案：

  建议在玩家成为经理时，同样判断其是否达到成为高级经理人的条件。

- 问题状态：

  已按照修改建议修复。

1.9 `recordDynamicProfits()`函数中，当系统为推荐人时只获取一层奖金，其余奖金没有流向也无法提取

- 风险级别：**高**
- 问题类型：实现漏洞

- 问题描述：

  当系统为推荐人时，由于在其所在推荐层级之上没有推荐人，系统应获取直至50层的奖金，游戏实现中只获取了一层奖金。例如，每当有玩家参与游戏，当系统所在推荐层级为1,2,3,4,5层时，未领取奖金分别为参与游戏金额的20%,15%,12%,10%,9%，这部分奖金在游戏中没有流向也无法提取。

  问题代码如下：

```
while(floor <= 50) {
    tmpRPid = players_[pid].aff;
    profits = getPercentageInTenThousandths(floor).mul(eJoin).div(10000);
    if (tmpRPid == 0) {
        // System always can get profits in 50 floor.
        return foundationProfits.add(profits);
    }
    ...
}
```

- 影响结果：

  存在金额在游戏中无法提取，严重影响游戏团队的利益。

- 规避方案：

  建议修改代码，当系统为推荐人时，统计该层级之上所有的奖金。

- 问题状态：

  已按照修改建议修复。

1.10 未确定推荐奖金记录时间超过12小时后，时间记录可能被覆盖，系统无法获得对应的超时奖金

- 风险级别：**中**
- 问题类型：实现漏洞
- 问题描述：

  当玩家参与游戏时，其推荐人若不满足领取对应层级的推荐奖金的条件，其金额会当做未确定推荐奖金存放在数组中。当用户达到领取该层级推荐奖金时，读取数组中的金额并奖励给该用户。合约中，`plyrRnds_[rid][pid].oldIndex[]`和`plyrRnds_[rid][pid].newIndex[]`分别记录用户在特定层级被统计的未确定推荐奖金对应时间段的起点与终点。每次更新数组时，若oldIndex对应的奖金超过12小时未领取，则奖金发放给系统，oldIndex数组中的值顺延直至oldIndex对应时间在12小时内。

  合约中使用长度为12的两个数组记录用户未分配奖金的时间与金额，以下描述中以`time[i]`与`amount[i]`简化表示。假设游戏过程中，某玩家在每个小时里都有其推荐的玩家入场游戏，且玩家始终不符合领取推荐奖金的条件。若游戏开始时对应小时数为hr。12小时后，`time[]`与`amount[]`数组中的值为(hr,n0),(hr+1,n1),...,(hr+11,n11)，此时`oldIndex`为0，`newIndex`为11，`hour`对应值为hr+12，`newLIndex`计算结果为0。代码实现中，`time[0]`与`amount[0]`首先被更新为(hr+12,n0+1)，即`oldIndex`对应推荐奖金的时间点被更新。此后的12个小时内，由

于hour-time[oldIndex]<=12始终成立，系统无法获得超时的奖金，且随着newIndex更新，time[]数组中其他值也会被新的时间覆盖。

由于仅当oldIndex <= newIndex时，系统会判断是否得到超时奖金。当newIndex更新至0后，无论oldIndex下标对应时间点是否超时，都无法被统计，newIndex会累加记录到oldIndex-1。此后当记录新的时间点时，newLIndex与oldIndex相等。newLIndex对应下标的时间首先进行了更新，即oldIndex下标对应时间点被更新，亦不符合系统获得超时奖金的条件。此后12个小时内，oldIndex都不会更新。

tryInsertUnDProfits()函数中出现问题的代码如下：

```
// Adds 1 when hour same with newest time in time array.
if (hour == plyrRnds_[rid][pid].undProfitsTime[floorIndex][newIndex]) {
    plyrRnds_[rid][pid].undProfitsAmount[floorIndex][newIndex] =
plyrRnds_[rid][pid].undProfitsAmount[floorIndex][newIndex].add(1);
    return 0;
}

// Try to insert a new item in new hour.
uint newLIndex = newIndex + 1;
// Index out of bound(Limit length of array to 12).
if (newLIndex > 11) {
    // Starts from first element.
    newLIndex = 0;
}

// Inserts/Updates new hour info.
plyrRnds_[rid][pid].undProfitsTime[floorIndex][newLIndex] = hour;
plyrRnds_[rid][pid].undProfitsAmount[floorIndex][newLIndex] =
plyrRnds_[rid][pid].undProfitsAmount[floorIndex][newLIndex].add(1);
// Try to delete old time whose distance with current time is bigger than
12.
for (; oldIndex <= newIndex; oldIndex++) {
    if (hour.sub(plyrRnds_[rid][pid].undProfitsTime[floorIndex]
[oldIndex]) <= 12) {
        break;
    }

    // Records discard profits to foundation.
    foundationProfits = foundationProfits.add(plyrRnds_[rid]
[pid].undProfitsAmount[floorIndex][oldIndex]);
}

// Updates index.
plyrRnds_[rid][pid].newIndex[floorIndex] = newLIndex;
if (oldIndex != plyrRnds_[rid][pid].oldIndex[floorIndex]) {
    plyrRnds_[rid][pid].oldIndex[floorIndex] = oldIndex;
}
```

- 影响结果：

  统计玩家未确定推荐奖金记录的时间数组被覆盖，导致系统无法获得应得的超时奖金。

- 规避方案：

  建议统计未确定推荐奖金的数组长度增加至14，可以保证newIndex从13更新至0时，oldIndex一定不为0。建议newIndex下标更新时，数组更新的语句改为plyrRnds_[rid][pid].undProfitsAmount[floorIndex][newLIndex] = 1。建议通过oldIndex遍历奖金记录时，考虑newIndex更新到0的情况。当newIndex < oldIndex时，oldIndex可以遍历至数组下标最大值，然后从0至newIndex。

- 问题状态：

  已按照修改建议修复。

1.11 getUnProfits() 函数中，推荐人应得的未确定推荐奖金可能计算不正确

- 风险级别：**中**
- 问题类型：实现漏洞
- 问题描述：

  合约中通过长度为12的数组统计玩家的未确定推荐奖金，当数组更新下标达到最大值时，newIndex会更新至0。此后oldIndex仅当oldIndex<=newIndex时遍历更新，最大只能遍历到newIndex+1。然而newIndex也更新到newIndex+1。即存在oldIndex与newIndex相等的情况。

  玩家直推ETH数对应层级的推荐奖金通过getUnProfits()函数计算，若计算未确定推荐奖金时oldIndex与newIndex相等，函数中只统计oldIndex下标所对应的时间点的推荐奖金，数组中其余时间点的推荐奖金都没有进行统计，玩家利益受到损害。

  getUnProfits() 函数中问题相关代码如下：

```
if (plyrRnds_[rid][pid].newIndex[floor] < plyrRnds_[rid]
[pid].oldIndex[floor]) {}
    endIndex = 11;

    for (uint i = 0; i <= plyrRnds_[rid][pid].newIndex[floor]; i++) {
        if (ended || timeInH.sub(plyrRnds_[rid]
[pid].undProfitsTime[floor][i]) > 12) {
            foundationProfits = foundationProfits.add(plyrRnds_[rid]
[pid].undProfitsAmount[floor][i]);
            continue;
        }

        dProfits = dProfits.add(plyrRnds_[rid]
[pid].undProfitsAmount[floor][i]);
    }
}
```

```
for (i = plyrRnds_[rid][pid].oldIndex[floor]; i <= endIndex; i++) {
    if (ended || timeInH.sub(plyrRnds_[rid][pid].undProfitsTime[floor]
[i]) > 12) {
        foundationProfits = foundationProfits.add(plyrRnds_[rid]
[pid].undProfitsAmount[floor][i]);
        continue;
    }

    dProfits = dProfits.add(plyrRnds_[rid][pid].undProfitsAmount[floor]
[i]);
}
```

- 影响结果:

  当oldIndex与newIndex相等时，用户只提取到一个时间点记录的未确定推荐奖金，影响玩家的利益。

- 规避方案:

  建议修改tryInsertUnDProfits()函数中oldIndex的更新算法，当newIndex < oldIndex时，oldIndex遍历至数组下标最大值，然后从0至newIndex。确保oldIndex表示的是玩家可以提取的未确定推荐奖金的最早记录。

- 问题状态:

  代码中超时奖金的计算与更新部分已修改，问题已修复。

1.12 未确定推荐奖金的分配由被推荐玩家参与游戏被动触发，游戏结束后存在未分配的未确定推荐奖金无法提取

- 风险级别：**中**
- 问题类型：实现漏洞
- 问题描述:

  游戏合约中，每当玩家参与游戏时，若推荐人不符合领取推荐奖金的条件时，未确定推荐奖金通过tryInsertUnDProfits()函数记录，同时触发超时奖金的统计。直推ETH数更新时，推荐人应得的未确定推荐奖金通过getUnProfits() 函数统计。若游戏结束时，推荐人没有达到领取对应层级推荐奖金的直推ETH数，合约中统计的未确定推荐奖金将无法提取。每轮游戏结束时，存在部分未确定推荐奖金无法提取。

- 影响结果:

  每轮游戏结束时，存在部分未确定推荐奖金无法提取，影响游戏团队与玩家的利益。

- 规避方案:

  建议增加UndDynamicProfit[rid][hour]数组，统计每个时间段所有玩家的未确定推荐奖金的变化，每当未确定推荐奖金更新时进行相应更新。系统的超时奖金可以直接通过该数组进行统计与提取，从而简化其他函数中的相关操作。

- 问题状态:

  已按照修改建议修复。

1.13 玩家入场时计算静态收益判断是否有玩家出局，此过程存在不必要计算

- 风险级别：**信息**
- 问题类型：代码优化
- 问题描述：

  游戏中每当玩家入场时，计算静态收益时首先调用`countStaticDivies()`函数计算是否有玩家出局，更新`enterASeq_[]`数组与`outFlag`变量。实际游戏中，由于用户的静态收益具有累加性，且出局时都只有A段收益，因此通过用户进入A段后某一时刻的静态收益，就可以计算出之后任一时刻该玩家的静态收益。此外，由于静态收益的计算在每轮游戏中都具有同样的规律，游戏过程中计算的`enterASeq_[]`数组可以在后面轮数的游戏中重复使用。只需在后面一轮游戏参与人次超过前面时，对`enterASeq_[]`数组进行更新。同样的，`staticAssist_[]`数组也存在重复计算的问题。
- 影响结果：

  玩家入场时进行没有必要的计算，增加gas消耗。
- 规避方案：

  建议通过状态变量存储游戏中参与的最多人次，在每轮游戏结束时更新，`betNum`变化时置0。游戏中，仅在当前序号大于最多人次时更新`enterASeq_[]`数组。某序号对应的`outFlag`也可以通过`enterASeq_[]`数组判断得到。建议使用状态变量记录玩家成为待出局玩家时的静态收益，`eSerial.sub(enterASeq_[outFlag+100]).mul(eJoin).div(1000)`即为待出局玩家在后面阶段的静态收益，两者相加即可得到更新后的静态收益。
- 问题状态：

  已按照修改建议对代码进行优化。

1.14 第201个玩家入场时的静态收益分配与游戏规则不符

- 风险级别：**低**
- 问题类型：逻辑漏洞
- 问题描述：

  游戏规则中，人数不足202时，采用后面的给前面人平均分配计算。第201个玩家入场时，前面每个人应该得到`0.3*eJoin/200`的奖金。代码实现中，每人应得按照A段与C段计算,实际为`0.1*eJoin/100`。
- 影响结果：

  玩家静态收益计算与游戏规则不符，影响玩家的利益。
- 规避方案：

  建议修改代码或者游戏规则，达到一致。
- 问题状态：

  已确认修改游戏规则中的描述，问题已修复。

PlayerBook.sol

2.1 `tryGetRefererByName()` 函数可以绑定任意推荐人关系

- 风险级别：**中**
- 问题类型：实现漏洞
- 问题描述：

  `tryGetRefererByName()` 函数可见性为public，当参数设置为游戏中未注册地址时，会调用`registerReferer()`函数对新玩家进行注册。攻击者在调用`tryGetRefererByName()`函数时，参数设为任意未注册地址与自己的昵称，可以将任意地址注册为自己的推荐玩家。
- 影响结果：

  推荐人关系可以任意添加，与实际游戏逻辑不符。游戏中任意推荐人关系可以被绑定，且`tryGetRefererByName()`调用会产生大量无效的游戏用户，影响`TotalPlayers`参数对实际参与游戏人数统计的准确性。
- 规避方案：

  游戏逻辑上，推荐人关系的绑定应由被推荐人主动发起。建议在`registerReferer()`函数中将被推荐人限定为`tx.origin`，即在函数中增加判定`require(player==tx.origin,'Referer must be added by player himself/herself.')`。主合约中`tryGetRefererByName()` 函数只在新用户发送ETH参加游戏时调用，可以保证被绑定的用户是新注册的玩家，有效阻止攻击者随意添加推荐人关系。
- 问题状态：

  已按照修改建议修复。

GuessContract.sol

3.1 `winGame()` 函数中当`data.winNumber-i = left` 且 `data.winNumber+i < right` 时，程序进入死循环

- 风险级别：**高**
- 问题类型：实现漏洞
- 问题描述：

  `winGame()`函数中，当data.winNumber-i = left 且 data.winNumber+i < right 时，i被赋值为`min(right.sub(data.winNumber),data.winNumber.sub(left))`，由于后面的`continue`语句，i的值没有变化并重新进入while循环。

  产生该漏洞的代码为：

```
if ((data.winNumber.sub(i)) < left) {
    if (data.recordToUser[data.winNumber.sub(i)].length != 0) {
        // Add users to winners.
        data.winnerRange[awardLevel] = i;
        data.numOfWinners[awardLevel] =
data.numOfWinners[awardLevel].add(data.recordToUser[data.winNumber.sub(i)
].length);
        hasWinner = true;
    }
// if right and left bound neither has a user.
} else if ((data.winNumber.add(i)) < right) {
    i = min(right.sub(data.winNumber),data.winNumber.sub(left));
    continue;
}
```

- 影响结果:

  由于每轮游戏结束后，用户猜测的数值以及data.winNumber都无法改变，该漏洞将导致`winGame()`函数执行失败而无法开奖，间接导致所有对`getProfits()`函数的调用失败。

- 规避方案:

  data.winNumber左侧存在中奖者区间的最大值为left-1，建议将更新i数值的语句改为`i = min(right.sub(data.winNumber),data.winNumber.sub(left)+1);`。

- 问题状态:

  已按照修改建议修复。

3.2 `winGame()` 函数中left与right值可能被重置，产生大量无效的计算

- 风险级别：**中**
- 问题类型：代码优化
- 问题描述:

  `winGame()`函数中，首先通过循环计算出可能存在赢家的猜测数字区间left和right。由于变量i初始值为1，且当data.winNumber±i被100整除时会重新计算left或者right，先前计算的结果可能会被重置，从而造成大量的额外计算。当data.winNumber $\equiv \pm 1 \pmod{100}$时，left与right都会被重置。此外，若data.winNumber一侧没有猜测者，即存在left = data.winNumber,或者right = data.winNumber的情况，开始阶段计算的left和right数值也会被重置。

  例如，前面几轮游戏玩家数都在10000左右，玩家猜测的数值在5000-20000的区间，新的一轮游戏特别火爆，人数增长至40001人。`winGame()`函数中，首先会计算出left=20000, right=40001(大于40001的区间对应data.spaceA[]元素全为0)。在后面计算中，由于40001-1被100整除，且39900-39999范围内没有猜测者，left被重新赋值为39900。此时,left=39900, i=1, right=40001。然后i从2-98循环，直到i=99,40001+99被100整除，right被更新为40200。此时,left=39900, i=99, right=40200,满足更新i的条件`(data.winNumber.sub(i)) >= left`以及`(data.winNumber.add(i)) < right`,i被赋值为101。然后，data.winNumber-101被100整除，left被赋值为39800，满足更新i的条件，i被赋值为199。再一次循环中，right被赋值

为40300，满足更新i的条件，i被赋值为201。如此往复，i的赋值变化为1->2->...->99->101->199->201->299->301->399->401->499->...->19899->19901->19999->20001->...。以上的过程中，存在大量无效的计算，从而消耗大量额外的gas。

产生该问题的代码如下：

```
i = 1;
bool hasWinner = false;
// Left
if (data.winNumber > i) {
    if ((data.winNumber.sub(i)).mod(100) == 0 &&
        (data.spaceA[(data.winNumber.sub(i)).div(100).sub(1)] ) == 0
    ) {
        left = ((data.winNumber.sub(i)).div(100).sub(1)).mul(100);
    }
}
// Right
if ((data.winNumber.add(i)).mod(100) == 0 &&
    (data.spaceA[(data.winNumber.add(i)).div(100)] ) == 0
) {
    right = ((data.winNumber.add(i)).div(100).add(1)).mul(100);
}
...
if (data.winNumber > i) {
    if ((data.winNumber.sub(i)) < left) {
    ...
    } else if ((data.winNumber.add(i)) < right) {
        i = min(right.sub(data.winNumber),data.winNumber.sub(left));
        continue;
    }
    ...
}
```

- 影响结果：

  某些情况下，造成大量无效的计算，从而消耗大量额外的gas。

- 规避方案：

  建议当left与right值更新时，将其值与原先的值比较后更新，i的初始值设置为min(right.sub(data.winNumber),data.winNumber.sub(left)+1)。当data.winNumber $\equiv \pm 1 \pmod{100}$或者data.winNumber的一侧没有猜测者时，代码需要做特别处理，避免left与right被重置，减少计算循环次数与gas消耗。

- 问题状态：

  `winGame()`函数已修改，上述问题已修复。

3.3 `winGame()` 函数中存在大量重复计算与 sload 调用，产生额外的gas消耗

- 风险级别：**低**
- 问题类型：代码优化

- 问题描述:

  `winGame()` 函数中data.winNumber被多次读取，产生大量sload指令,每次sload指令需要消耗300gas。若将其存储在局部变量中，通过mload指令，每次读取只需消耗3gas。此外，left与right的计算中，每一步计算都包含data.winNumber.div(100)，同样存在大量重复计算。在计算left与right的过程中，若未寻找到存在猜测者的区间，left与right保持初始值data.winNumber，后面会重复对left与right进行计算。

  产生该问题的代码如下:

```
left = data.winNumber;
right = data.winNumber;
for (i = 1; data.winNumber.div(100) >= i; i++) {
    if ((data.spaceA[data.winNumber.div(100).sub(i)]) != 0) {
        left = (data.winNumber.div(100).sub(i).add(1)).mul(100);
        break;
    }
}
for (i = 1; data.winNumber.div(100).add(i) < 1000; i++) {
    if ((data.spaceA[data.winNumber.div(100).add(i)]) != 0) {
        right = (data.winNumber.div(100).add(i)).mul(100);
        break;
    }
}
```

- 影响结果:

  代码中存在大量重复的计算与 sload 调用，会产生额外的gas消耗。

- 规避方案:

  建议首先创建局部变量存储data.winNumber的值，在后续计算中调用。在计算left与right值时，首先计算data.winNumber.div(100)的值，存入局部变量，然后代入后续的计算中。建议将left与right的初始值设为循环计算结果的边界值，若寻找到存在猜测者的区间，left与right范围会相应缩小。如此可以避免未寻找到存在猜测者的区间时left与right保持初始值data.winNumber，引入额外计算。此外，当left计算出有效值时，right的计算无需循环1000次，也可做适当优化。建议修改代码如下:

```
uint winN = data.winNumber;
...
...
uint N = winN.div(100);
uint N_r = 1000;
left = 0;
for (i = 1; N >= i; i++) {
    if ((data.spaceA[N.sub(i)]) != 0) {
        left = (N.sub(i).add(1)).mul(100);
        N_r = min(N.add(i),1000);
        break;
    }
}
```

```
right = N_r.mul(100);
for (i = 1; N.add(i) < N_r; i++) {
    if ((data.spaceA[N.add(i)]) != 0) {
        right = (N.add(i)).mul(100);
        break;
    }
}
```

- 问题状态:

  `winGame()`函数已修改，上述问题已修复。

A.1 代码中提示信息存在语法错误或表达不当

- 风险级别：**信息**
- 问题类型：代码优化
- 问题描述:

  代码中提示信息存在语法错误或表达不当，具体错误如下:

  - OneETH.sol#L101: `require(activated_ == true, "its not ready yet.");`
  - OneETH.sol#L204: `require(ok, "your referer haven't register");`
  - OneETH.sol#L210: `require(plyrRnds_[rID_][pid].bets.length <= maxBuy_, "too much buy, 200 max");`
  - OneETH.sol#L421: `require(activated_ == false, "have active before");`
  - OneETH.sol#L422: `require(time >= now, "start time shouldn't smaller than now");`
  - GuessContract.sol#L149,152: `revert("player has already win the guess.");`
  - GuessContract.sol#L372: `require(pid != 0, "Pid con't be 0.");`
  - PlayerBook.sol#L166: `require(bName != systemName, "you can register system name");`
- 影响结果:

  提示信息存在错误，影响消息传递以及外界对游戏合约的整体印象。
- 规避方案:

  提及的提示信息的建议修改如下:

  - OneETH.sol#L101: `require(activated_ == true, "it's not ready yet.");`
  - OneETH.sol#L204: `require(ok, "your referer hasn't registered yet");`
  - OneETH.sol#L210: `require(plyrRnds_[rID_][pid].bets.length <= maxBuy_, "too much bought, 200 max");`
  - OneETH.sol#L421: `require(activated_ == false, "the game has already been activated before");`

    - OneETH.sol#L422: `require(time >= now, "start time shouldn't be smaller than now");`
    - GuessContract.sol#L149,152: `revert("player has already won the guess.");`
    - GuessContract.sol#L372: `require(pid != 0, "Pid can't be 0.");`
    - PlayerBook.sol#L166: `require(bName != systemName, "you can't register system name");`

- 问题状态:
  已按照修改建议修复。