



CyberCrime Shield
cybercrimeshield.org

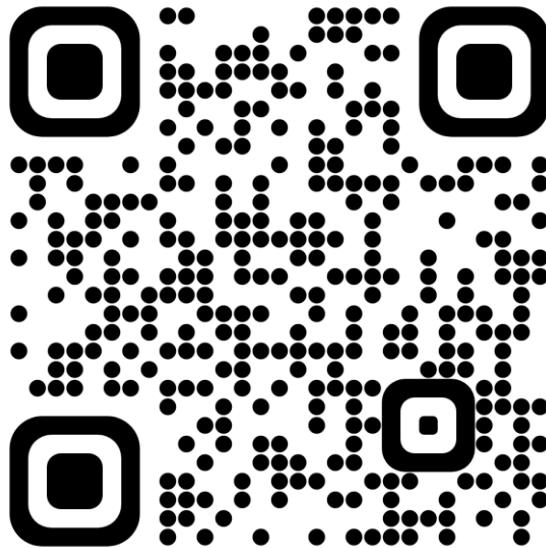
Smart Contract Audit Report

Fantom Fortune

<https://www.fantomfortune.com/>

AUDIT TYPE: **PUBLIC**

YOU CAN CHECK THE VALIDITY USING THE QR CODE OR LINK:



<https://cybercrimeshield.org/secure/fantomfortune>

Report ID: A0125928

July 2, 2022



TABLE OF CONTENTS

SMART CONTRACT	3
INTRODUCTION	4
AUDIT METHODOLOGY	5
ISSUES DISCOVERED	6
AUDIT SUMMARY	6
CONCLUSION	7
SOURCE CODE	7-22



CyberCrime Shield

cybercrimeshield.org

SMART CONTRACT

<https://ftmscan.com/address/0x60a230C87fB70b9664b610511A0544ccE61053fa#code>

Mirror: <https://cybercrimeshield.org/secure/uploads/fantomfortune.sol>

Compiler Version: v0.8.14+commit.80d49f37

License: MIT

CRC32: DBC8E4C8

MD5: B20E91F6DF87817623847F9AF1DECAB1

SHA-1: 1F6EFE3A488BBABD2C53D319710B44A248D233E4



INTRODUCTION

Blockchain platforms, such as Nakamoto's Bitcoin, enable the trade of crypto currencies between mutually mistrusting parties.

To eliminate the need for trust, Nakamoto designed a peer-to-peer network that enables its peers to agree on the trading transactions.

Smart contracts have shown to be applicable in many domains including financial industry, public sector and cross-industry.

The increased adoption of smart contracts demands strong security guarantees. Unfortunately, it is challenging to create smart contracts that are free of security bugs.

Consequently, critical vulnerabilities in smart contracts are discovered and exploited every few months.

In turn, these exploits have led to losses reaching billions worth of USD in the past few years.

It is apparent that effective security checks for smart contracts are strictly needed.

Our company provides comprehensive, independent smart contract auditing.

We help stakeholders confirm the quality and security of their smart contracts using our standardized audit process.

The scope of this audit was to analyze and document the Fantom Fortune contract.

This document is not financial advice, you perform all financial actions on your own responsibility.



AUDIT METHODOLOGY

1. Design Patterns

We inspect the structure of the smart contract, including both manual and automated analysis.

2. Static Analysis

The static analysis is performed using a series of automated tools, purposefully designed to test the security of the contract.

All the issues found by tools were manually checked (rejected or confirmed).

3. Manual Analysis

Contract reviewing to identify common vulnerabilities. Comparing of requirements and implementation. Reviewing of a smart contract for compliance with specified customer requirements. Checking for energy optimization and self-documentation. Running tests of the properties of the smart contract in test net.



ISSUES DISCOVERED

Issues are listed from most critical to least critical. Severity is determined by an assessment of the risk of exploitation or otherwise unsafe behavior.

Severity Levels

Critical - Funds may be allocated incorrectly, lost or otherwise result in a significant loss.

Medium - Affects the ability of the contract to operate.

Low - Minimal impact on operational ability.

Informational - No impact on the contract.

AUDIT SUMMARY

The summary result of the audit performed is presented in the table below

Findings list:

LEVEL	AMOUNT
Critical	0
Medium	0
Low	0
Informational	0



CONCLUSION

























- The contract has a clear structure and is easy to read
- Gas usage is optimal
- Contract is fully Ethereum & Fantom completable
- No backdoors or overflows are present in the contract



SOURCE CODE

```
1. /**
2.  *Submitted for verification at FtmScan.com on 2022-07-01
3.  */
4.
5. // SPDX-License-Identifier: MIT
6.
7.
8. /*
9.
10.
11. [REDACTED]
12. [REDACTED]
13. [REDACTED]
```



```
14.        
15.        
16.        
17.
18.
19. */

20.
21.
22. pragma solidity >=0.4.22 <0.9.0;
23.
24. // Interface Price Feed
25. interface AggregatorV3Interface {
26.     function decimals() external view returns (uint8);
27.
28.     function description() external view returns (string memory);
29.
30.     function version() external view returns (uint256);
31.
32.     function getRoundData(uint80 _roundId)
33.         external
34.         view
35.         returns (
36.             uint80 roundId,
37.             int256 answer,
38.             uint256 startedAt,
39.             uint256 updatedAt,
40.             uint80 answeredInRound
41.         );
42.
43.     function latestRoundData()
44.         external
45.         view
46.         returns (
47.             uint80 roundId,
48.             int256 answer,
49.             uint256 startedAt,
50.             uint256 updatedAt,
51.             uint80 answeredInRound
52.         );
53. }
54.
```




```
55. // Main Contract
56. contract FtmFortune {
57.     using SafeMath for uint256;
58.     AggregatorV3Interface public priceFeedBnb;
59.     address payable public PLATFORM_WALLET;
60.     address payable public INSURANCE_WALLET;
61.     address payable public DEPLOYER;
62.
63.     uint256 public constant MIN_AMOUNT = 20 ether;
64.     uint256[3] public REF_DEP_PERCENTS = [500, 300, 100];
65.     uint256[3] public REF_WID_PERCENTS = [150, 50, 10];
66.     uint256 public constant DEPOSIT_FEE = 800; // 8% deposit fee
67.     uint256 public constant WITHDRAW_FEE = 800; // 8% withdraw fee
68.     uint256 public constant P_WITHDRAW_FEE = 6250; // 5% withdraw fee for owner
69.     uint256 public constant I_WITHDRAW_FEE = 3750; // 3% withdraw fee for
    insurance
70.     uint256 public constant PERCENT_STEP = 20; // 0.2% daily increment
71.     uint256 public constant WITHDRAW_TAX_PERCENT = 3500; // emergency withdraw
    tax 35%
72.     uint256 public constant P_WITHDRAW_TAX_PERCENT = 500; // emergency withdraw
    tax 5% for owner
73.     uint256 public constant I_WITHDRAW_TAX_PERCENT = 500; // emergency withdraw
    tax 5% for insurance
74.     uint256 public constant MAX_HOLD_PERCENT = 200; // 2% hold bonus
75.     uint256 public constant PERCENTS_DIVIDER = 10000;
76.     uint256 public TIME_STEP = 1 days;
77.
78.     uint256 public startTime;
79.     uint256 public totalStaked;
80.     uint256 public totalWithdrawn;
81.     uint256 public totalRefBonus;
82.     uint256 public insuranceTriggerBalance;
83.     uint256 public totalUsers;
84.
85.     bool public launched;
86.
87.     struct Plan {
88.         uint256 time;
89.         uint256 percent;
90.     }
91.
92.     Plan[] internal plans;
93.
94.     struct Deposit {
95.         uint8 plan;
```



```
96.         uint256 percent;
97.         uint256 amount;
98.         uint256 profit;
99.         uint256 start;
100.        uint256 finish;
101.    }
102.
103.    struct User {
104.        Deposit[] deposits;
105.        uint256 checkpoint;
106.        uint256 holdBonusCheckpoint;
107.        address referrer;
108.        uint256[3] levels;
109.        uint256 bonus;
110.        uint256 debt;
111.        uint256 totalBonus;
112.        uint256 totalWithdrawn;
113.    }
114.
115.    mapping(address => User) internal users;
116.    mapping(uint256 => uint256) public INSURANCE_MAXBALANCE;
117.
118.    modifier onlyDeployer() {
119.        require(msg.sender == DEPLOYER, "NOT AN OWNER");
120.        _;
121.    }
122.
123.    event Newbie(address user);
124.    event NewDeposit(
125.        address indexed user,
126.        uint8 plan,
127.        uint256 percent,
128.        uint256 amount,
129.        uint256 profit,
130.        uint256 start,
131.        uint256 finish
132.    );
133.    event Withdrawn(address indexed user, uint256 amount);
134.    event REINVEST(address indexed user, uint256 amount);
135.    event RefBonus(
136.        address indexed referrer,
137.        address indexed referral,
138.        uint256 indexed level,
139.        uint256 amount
140.    );
```



```
141.     event FeePaid(address indexed user, uint256 totalAmount);
142.
143.     constructor(
144.         address payable _platform,
145.         address payable _insurance,
146.         uint256 _time
147.     ) {
148.         startTime = _time;
149.         DEPLOYER = payable(msg.sender);
150.         PLATFORM_WALLET = _platform;
151.         INSURANCE_WALLET = _insurance;
152.         priceFeedBnb = AggregatorV3Interface(
153.             // mainnet
154.             0xf4766552D15AE4d256Ad41B6cf2933482B0680dc
155.             // testnet
156.             // 0xe04676B9A9A2973BCb0D1478b5E1E9098BBB7f3D
157.         );
158.
159.         plans.push(Plan(10, 1258));
160.         plans.push(Plan(16, 1143));
161.         plans.push(Plan(21, 1236));
162.         plans.push(Plan(26, 1274));
163.         plans.push(Plan(10, 2418));
164.         plans.push(Plan(16, 2056));
165.         plans.push(Plan(21, 1808));
166.         plans.push(Plan(26, 1586));
167.     }
168.
169.     receive() external payable {}
170.
171.     function invest(address referrer, uint8 plan) public payable {
172.         require(launched, "wait for the launch");
173.         require(!isContract(msg.sender));
174.         require(msg.value >= MIN_AMOUNT, "less than min Limit");
175.         deposit(msg.sender, referrer, plan, msg.value);
176.     }
177.
178.     function deposit(
179.         address userAddress,
180.         address referrer,
181.         uint8 plan,
182.         uint256 amount
183.     ) internal {
184.         require(plan < 8, "Invalid plan");
185.         User storage user = users[userAddress];
```



```
186.
187.         uint256 fee = amount.mul(DEPOSIT_FEE).div(PERCENTS_DIVIDER);
188.         PLATFORM_WALLET.transfer(fee);
189.         emit FeePaid(userAddress, fee);
190.
191.         if (user.referrer == address(0)) {
192.             if (
193.                 (users[referrer].deposits.length == 0 ||
194.                  referrer == userAddress)
195.             ) {
196.                 referrer = DEPLOYER;
197.             }
198.
199.             user.referrer = referrer;
200.
201.             address upline = user.referrer;
202.             for (uint256 i = 0; i < REF_DEP_PERCENTS.length; i++) {
203.                 if (upline != address(0)) {
204.                     users[upline].levels[i] =
205.                     users[upline].levels[i].add(1);
206.                     upline = users[upline].referrer;
207.                 } else break;
208.             }
209.
210.             if (user.referrer != address(0)) {
211.                 address upline = user.referrer;
212.                 for (uint256 i = 0; i < REF_DEP_PERCENTS.length; i++) {
213.                     if (upline != address(0)) {
214.                         uint256 refAmount = amount.mul(REF_DEP_PERCENTS[i]).div(
215.                         PERCENTS_DIVIDER
216.                         );
217.                         users[upline].bonus =
218.                         users[upline].bonus.add(refAmount);
219.                         users[upline].totalBonus = users[upline].totalBonus.add(
220.                         refAmount
221.                         );
222.                         totalRefBonus = totalRefBonus.add(refAmount);
223.                         emit RefBonus(upline, userAddress, i, refAmount);
224.                         upline = users[upline].referrer;
225.                     } else break;
226.                 }
227.
228.                 if (user.deposits.length == 0) {
```



CyberCrime Shield

cybercrimeshield.org

```
229.         totalUsers = totalUsers.add(1);
230.         user.checkpoint = block.timestamp;
231.         user.holdBonusCheckpoint = block.timestamp;
232.         emit Newbie(userAddress);
233.     }
234.
235.     (uint256 percent, uint256 profit, uint256 finish) = getResult(
236.         plan,
237.         amount
238.     );
239.     user.deposits.push(
240.         Deposit(plan, percent, amount, profit, block.timestamp, finish)
241.     );
242.
243.     totalStaked = totalStaked.add(amount);
244.     emit NewDeposit(
245.         userAddress,
246.         plan,
247.         percent,
248.         amount,
249.         profit,
250.         block.timestamp,
251.         finish
252.     );
253. }
254.
255. function withdraw() public {
256.     User storage user = users[msg.sender];
257.     require(
258.         block.timestamp >= user.checkpoint.add(TIME_STEP),
259.         "wait for next withdraw"
260.     );
261.
262.     uint256 totalAmount = getUserDividends(msg.sender);
263.     uint256 referralBonus = getUserReferralBonus(msg.sender);
264.     if (referralBonus > 0) {
265.         user.bonus = 0;
266.         totalAmount = totalAmount.add(referralBonus);
267.     }
268.     if (user.debt > 0) {
269.         totalAmount = totalAmount.add(user.debt);
270.         user.debt = 0;
271.     }
272.     require(totalAmount > 0, "User has no dividends");
273.     uint256 fee = totalAmount.mul(WITHDRAW_FEE).div(PERCENTS_DIVIDER);
```



```
274. PLATFORM_WALLET.transfer(fee.mul(P_WITHDRAW_FEE).div(PERCENTS_DIVIDE
    R));
275. INSURANCE_WALLET.transfer(
276.     fee.mul(I_WITHDRAW_FEE).div(PERCENTS_DIVIDER)
277. );
278. totalAmount = totalAmount.sub(fee);
279.
280. uint256 contractBalance = address(this).balance;
281. if (totalAmount > contractBalance) {
282.     user.debt = user.debt.add(totalAmount.sub(contractBalance));
283.     totalAmount = contractBalance;
284. }
285.
286. user.checkpoint = block.timestamp;
287. user.holdBonusCheckpoint = block.timestamp;
288. user.totalWithdrawn = user.totalWithdrawn.add(totalAmount);
289. totalWithdrawn = totalWithdrawn.add(totalAmount);
290.
291. payable(msg.sender).transfer(totalAmount);
292.
293. if (user.referrer != address(0) && contractBalance >=
    totalAmount.div(20)) {
294.     address upline = user.referrer;
295.     for (uint256 i = 0; i < REF_WID_PERCENTS.length; i++) {
296.         if (upline != address(0)) {
297.             uint256 refAmount = totalAmount
298.                 .mul(REF_WID_PERCENTS[i])
299.                 .div(PERCENTS_DIVIDER);
300.             users[upline].bonus =
301.                 users[upline].bonus.add(refAmount);
302.             users[upline].totalBonus = users[upline].totalBonus.add(
303.                 refAmount
304.             );
305.             totalRefBonus = totalRefBonus.add(refAmount);
306.             emit RefBonus(upline, msg.sender, i, refAmount);
307.             upline = users[upline].referrer;
308.         } else break;
309.     }
310.
311.     emit Withdrawn(msg.sender, totalAmount);
312. }
313.
314. function emergencyWithdraw(uint256 index) public {
315.     User storage user = users[msg.sender];
```



CyberCrime Shield

cybercrimeshield.org

```
316.         uint8 plan = user.deposits[index].plan;
317.         require(plan == 6 || plan == 7, "invlaid package");
318.         require(isDepositActive(msg.sender, index), "deposit not active");
319.         uint256 depositAmount = user.deposits[index].amount;
320.         uint256 forceWithdrawTax = (depositAmount * WITHDRAW_TAX_PERCENT) /
321.             PERCENTS_DIVIDER;
322.         uint256 pWithdrawTax = (depositAmount * P_WITHDRAW_TAX_PERCENT) /
323.             PERCENTS_DIVIDER;
324.         uint256 iWithdrawTax = (depositAmount * I_WITHDRAW_TAX_PERCENT) /
325.             PERCENTS_DIVIDER;
326.         PLATFORM_WALLET.transfer(pWithdrawTax);
327.         INSURANCE_WALLET.transfer(iWithdrawTax);
328.
329.         uint256 totalAmount = depositAmount - forceWithdrawTax;
330.
331.         uint256 contractBalance = address(this).balance;
332.         if (totalAmount > contractBalance) {
333.             user.debt = user.debt.add(totalAmount.sub(contractBalance));
334.             totalAmount = contractBalance;
335.         }
336.         user.totalWithdrawn += totalAmount;
337.         user.deposits[index].finish = block.timestamp;
338.         user.deposits[index].profit = 0;
339.         totalWithdrawn += totalAmount;
340.
341.         payable(msg.sender).transfer(totalAmount);
342.
343.         emit Withdrawn(msg.sender, totalAmount);
344.     }
345.
346.     function launch() external onlyDeployer {
347.         require(!launched, "Already launched");
348.         launched = true;
349.         startTime = block.timestamp;
350.     }
351.
352.     function changeDeployer(address payable _new) external onlyDeployer {
353.         require(!isContract(_new), "Can't be a contract");
354.         DEPLOYER = _new;
355.     }
356.
357.     function changePlatform(address payable _new) external onlyDeployer {
358.         require(!isContract(_new), "Can't be a contract");
359.         PLATFORM_WALLET = _new;
360.     }
```



```
361.
362.     function getContractBalance() public view returns (uint256) {
363.         return address(this).balance;
364.     }
365.
366.     function getPlanInfo(uint8 plan)
367.         public
368.         view
369.         returns (uint256 time, uint256 percent)
370.     {
371.         time = plans[plan].time;
372.         percent = plans[plan].percent;
373.     }
374.
375.     function getPercent(uint8 plan) public view returns (uint256) {
376.         if (block.timestamp > startTime) {
377.             return
378.                 plans[plan].percent.add(
379.                     PERCENT_STEP.mul(block.timestamp.sub(startTime)).div(
380.                         TIME_STEP
381.                     )
382.                 );
383.         } else {
384.             return plans[plan].percent;
385.         }
386.     }
387.
388.     function getResult(uint8 plan, uint256 amount)
389.         public
390.         view
391.         returns (
392.             uint256 percent,
393.             uint256 profit,
394.             uint256 finish
395.         )
396.     {
397.         percent = getPercent(plan);
398.
399.         if (plan < 4) {
400.             profit = amount.mul(percent).mul(plans[plan].time).div(100);
401.         } else if (plan < 8) {
402.             profit = amount.mul(percent);
403.             for (uint256 i = 1; i < plans[plan].time; i++) {
404.                 uint256 newProfit =
profit.mul(percent).div(PERCENTS_DIVIDER);
```




```
405.         profit = profit.add(newProfit);
406.     }
407.     profit = profit.div(100);
408. }
409.
410.     finish = block.timestamp.add(plans[plan].time.mul(TIME_STEP));
411. }
412.
413. // to get real time price of Bnb
414. function getLatestPriceBnb() public view returns (uint256) {
415.     (, int256 price, , , ) = priceFeedBnb.latestRoundData();
416.     return uint256(price);
417. }
418.
419. function getUserDividends(address userAddress)
420.     public
421.     view
422.     returns (uint256)
423. {
424.     User storage user = users[userAddress];
425.
426.     uint256 totalAmount;
427.     uint256 holdBonus = getUserHoldBonusPercent(userAddress);
428.
429.     for (uint256 i = 0; i < user.deposits.length; i++) {
430.         if (user.checkpoint < user.deposits[i].finish) {
431.             if (user.deposits[i].plan < 4) {
432.                 uint256 share = user
433.                     .deposits[i]
434.                     .amount
435.                     .mul(user.deposits[i].percent.add(holdBonus))
436.                     .div(PERCENTS_DIVIDER);
437.                 uint256 from = user.deposits[i].start > user.checkpoint
438.                     ? user.deposits[i].start
439.                     : user.checkpoint;
440.                 uint256 to = user.deposits[i].finish < block.timestamp
441.                     ? user.deposits[i].finish
442.                     : block.timestamp;
443.                 if (from < to) {
444.                     totalAmount = totalAmount.add(
445.                         share.mul(to.sub(from)).div(TIME_STEP)
446.                     );
447.                 }
448.             } else if (block.timestamp > user.deposits[i].finish) {
449.                 totalAmount = totalAmount.add(
```



```
450.             user.deposits[i].profit.div(100)
451.             );
452.         }
453.     }
454. }
455.
456.     return totalAmount;
457. }
458.
459. function getUserHoldBonusPercent(address userAddress)
460.     public
461.     view
462.     returns (uint256)
463.     {
464.         User storage user = users[userAddress];
465.
466.         uint256 timeMultiplier = block
467.             .timestamp
468.             .sub(user.holdBonusCheckpoint)
469.             .div(TIME_STEP);
470.         timeMultiplier = timeMultiplier.mul(20); // +0.2% per day
471.         if (timeMultiplier > MAX_HOLD_PERCENT) {
472.             timeMultiplier = MAX_HOLD_PERCENT;
473.         }
474.         return timeMultiplier;
475.     }
476.
477. function getUserCheckpoint(address userAddress)
478.     public
479.     view
480.     returns (uint256)
481.     {
482.         return users[userAddress].checkpoint;
483.     }
484.
485. function getUserHoldBonusCheckpoint(address userAddress)
486.     public
487.     view
488.     returns (uint256)
489.     {
490.         return users[userAddress].holdBonusCheckpoint;
491.     }
492.
493. function getUserReferrer(address userAddress)
494.     public
```



```
495.         view
496.         returns (address)
497.     {
498.         return users[userAddress].referrer;
499.     }
500.
501.     function getUserDownlineCount(address userAddress)
502.     public
503.     view
504.     returns (
505.         uint256 level1,
506.         uint256 level2,
507.         uint256 level3
508.     )
509.     {
510.         level1 = users[userAddress].levels[0];
511.         level2 = users[userAddress].levels[1];
512.         level3 = users[userAddress].levels[2];
513.     }
514.
515.     function getUserReferralBonus(address userAddress)
516.     public
517.     view
518.     returns (uint256)
519.     {
520.         return users[userAddress].bonus;
521.     }
522.
523.     function getUserReferralTotalBonus(address userAddress)
524.     public
525.     view
526.     returns (uint256)
527.     {
528.         return users[userAddress].totalBonus;
529.     }
530.
531.     function getUserReferralWithdrawn(address userAddress)
532.     public
533.     view
534.     returns (uint256)
535.     {
536.         return users[userAddress].totalBonus.sub(users[userAddress].bonus);
537.     }
538.
```



CyberCrime Shield

cybercrimeshield.org

```
539.     function getUserDebt(address userAddress) public view returns (uint256)
540.     {
541.         return users[userAddress].debt;
542.     }
543.     function getUserAvailable(address userAddress)
544.         public
545.         view
546.         returns (uint256)
547.     {
548.         return
549.             getUserReferralBonus(userAddress)
550.             .add(getUserDividends(userAddress))
551.             .add(getUserDebt(userAddress));
552.     }
553.
554.     function getUserAmountOfDeposits(address userAddress)
555.         public
556.         view
557.         returns (uint256)
558.     {
559.         return users[userAddress].deposits.length;
560.     }
561.
562.     function getUserTotalDeposits(address userAddress)
563.         public
564.         view
565.         returns (uint256 amount)
566.     {
567.         for (uint256 i = 0; i < users[userAddress].deposits.length; i++) {
568.             amount = amount.add(users[userAddress].deposits[i].amount);
569.         }
570.     }
571.
572.     function getUserDepositInfo(address userAddress, uint256 index)
573.         public
574.         view
575.         returns (
576.             uint8 plan,
577.             uint256 percent,
578.             uint256 amount,
579.             uint256 profit,
580.             uint256 start,
581.             uint256 finish
582.         )
```



CyberCrime Shield

cybercrimeshield.org

```
583.     {
584.         User storage user = users[userAddress];
585.
586.         plan = user.deposits[index].plan;
587.         percent = user.deposits[index].percent;
588.         amount = user.deposits[index].amount;
589.         profit = user.deposits[index].profit;
590.         start = user.deposits[index].start;
591.         finish = user.deposits[index].finish;
592.     }
593.
594.     function getUserTotalWithdrawn(address userAddress)
595.         public
596.         view
597.         returns (uint256)
598.     {
599.         return users[userAddress].totalWithdrawn;
600.     }
601.
602.     function isDepositActive(address userAddress, uint256 index)
603.         public
604.         view
605.         returns (bool)
606.     {
607.         User storage user = users[userAddress];
608.
609.         return (user.deposits[index].finish > block.timestamp);
610.     }
611.
612.     function isContract(address addr) internal view returns (bool) {
613.         uint256 size;
614.         assembly {
615.             size := extcodesize(addr)
616.         }
617.         return size > 0;
618.     }
619. }
620.
621. library SafeMath {
622.     function add(uint256 a, uint256 b) internal pure returns (uint256) {
623.         uint256 c = a + b;
624.         require(c >= a, "SafeMath: addition overflow");
625.
626.         return c;
627.     }
```



```
628.
629.     function sub(uint256 a, uint256 b) internal pure returns (uint256) {
630.         require(b <= a, "SafeMath: subtraction overflow");
631.         uint256 c = a - b;
632.
633.         return c;
634.     }
635.
636.     function mul(uint256 a, uint256 b) internal pure returns (uint256) {
637.         if (a == 0) {
638.             return 0;
639.         }
640.
641.         uint256 c = a * b;
642.         require(c / a == b, "SafeMath: multiplication overflow");
643.
644.         return c;
645.     }
646.
647.     function div(uint256 a, uint256 b) internal pure returns (uint256) {
648.         require(b > 0, "SafeMath: division by zero");
649.         uint256 c = a / b;
650.
651.         return c;
652.     }
653. }
```