Ace Poker

Smart Contract Security Audit

Prepared by ShellBoxes

July 24th, 2023 - July 26th, 2023

Shellboxes.com

contact@shellboxes.com

Document Properties

Client	Ace Poker
Version	1.0
Classification	Public

Scope

Contract Name	Contract Address
AcePoker	0xf70C5A7923D4F471C019912E4fA87d6B964A3B0d
VestingContract	0x67271F27221DDdAdD9115Fdcb87C0E6f08bB2700

Re-Audit

Contract Name	Contract Address
AcePoker	0x7BBB3Bd1b6C38bd6aE0cF53b4e97d933685b25C3
VestingContract	0xAaF7B06258187385fc64C2b16d70E75F30523bD6

Contacts

COMPANY	EMAIL
ShellBoxes	contact@shellboxes.com

Contents

1 Introduction

Ace Poker engaged ShellBoxes to conduct a security assessment on the Ace Poker beginning on July 24th, 2023 and ending July 26th, 2023. In this report, we detail our methodical approach to evaluate potential security issues associated with the implementation of smart contracts, by exposing possible semantic discrepancies between the smart contract code and design document, and by recommending additional ideas to optimize the existing code. Our findings indicate that the current version of smart contracts can still be enhanced further due to the presence of many security and performance concerns.

This document summarizes the findings of our audit.

1.1 About Ace Poker

ACE Poker - A New Poker Game Integrating Web3 Technology. The leading online poker site with cryptocurrency deposit and withdrawal options, and a profit-sharing program that gives 50% of profits to holders.

Issuer	Ace Poker
Website	`https://acepoker.io`
Type	Solidity Smart Contract
Documentation	Ace Poker Documentation
Audit Method	Whitebox

1.2 Approach & Methodology

ShellBoxes used a combination of manual and automated security testing to achieve a balance between efficiency, timeliness, practicability, and correctness within the audit's scope. While manual testing is advised for identifying problems in logic, procedure, and implementation, automated testing techniques help to expand the coverage of smart contracts and can quickly detect code that does not comply with security best practices.

1.2.1 Risk Methodology

Vulnerabilities or bugs identified by ShellBoxes are ranked using a risk assessment technique that considers both the LIKELIHOOD and IMPACT of a security incident. This framework is effective at conveying the features and consequences of technological vulnerabilities.

Its quantitative paradigm enables repeatable and precise measurement, while also revealing the underlying susceptibility characteristics that were used to calculate the Risk scores. A risk level will be assigned to each vulnerability on a scale of 5 to 1, with 5 indicating the greatest possibility or impact.

- Likelihood quantifies the probability of a certain vulnerability being discovered and exploited in the untamed.
- Impact quantifies the technical and economic costs of a successful attack.
- Severity indicates the risk's overall criticality.

Probability and impact are classified into three categories: H, M, and L, which correspond to high, medium, and low, respectively. Severity is determined by probability and impact and is categorized into four levels, namely Critical, High, Medium, and Low.

Impact				
	High	Critical	High	Medium
	Medium	High	Medium	Low
	Low	Medium	Low	Low
		High	Medium	Low

Likelihood

2 Findings Overview

2.1 Disclaimer

This audit report evaluates the source code and logic of the contract for security and functionality. However, it does not guarantee that the contract owner will insert the same values of tokenomics. The decision to modify tokenomics or contract behavior lies solely with the contract owner. Users should exercise their own judgment and due diligence before engaging with the contract. This report is not an endorsement or investment advice, and any risks associated with using the contract are the sole responsibility of the user or entity.

2.2 Summary

The following is a synopsis of our conclusions from our analysis of the Ace Poker implementation. During the first part of our audit, we examine the smart contract source code and run the codebase via a static code analyzer. The objective here is to find known coding problems statically and then manually check (reject or confirm) issues highlighted by the tool. Additionally, we check business logics, system processes, and DeFi-related components manually to identify potential hazards and/or defects.

2.3 Key Findings

In general, these smart contracts are well-designed and constructed, but their implementation might be improved by addressing the discovered flaws, which include , 2 low-severity vulnerabilities.

Vulnerabilities	Severity	Status
SHB.1. Owner Can Renounce Ownership	LOW	Fixed
SHB.2. Missing Value Verification	LOW	Fixed

3 Finding Details

SHB.1 Owner Can Renounce Ownership

- Severity : LOW
- Likelihood : 1
- Status : Fixed
- Impact : 2

Description:

Typically, the contract's owner is the account that deploys the contract. As a result, the owner is able to perform certain privileged activities on his behalf. The renounceOwnership function is used in smart contracts to renounce ownership. Otherwise, if the contract's ownership has not been transferred previously, it will never have an Owner, which is risky.

Files Affected:

SHB.1.1: AcePoker.sol

```
8  contract AcePoker is ERC20, Ownable {
```

SHB.1.2: VestingContract.sol

```
11  contract VestingContract is Ownable,Initializable {
```

Recommendation:

It is advised that the Owner cannot call renounceOwnership without first transferring ownership to a different address. Additionally, if a multi-signature wallet is utilized, executing the renounceOwnership method for two or more users should be confirmed. Alternatively, the Renounce Ownership functionality can be disabled by overriding it.

Updates

The team has added the renounceOwnership function, if called the transaction will be reverted.

SHB.1.3: AcePoker.sol,VestingContract.sol

```
    function renounceOwnership() public override onlyOwner {
            revert("can't renounceOwnership");
        }
```

SHB.2 Missing Value Verification

- Severity : LOW
- Status : Fixed
- Likelihood : 1
- Impact : 2

Description:

Certain functions lack a safety check in the values, the values of the arguments should be verified to allow only the ones that align with the contract's logic. In the createVestingSchedule function, the contract should verify if _tgePerc is less than 100%.

Files Affected:

SHB.2.1: VestingContract.sol

```
53  function createVestingSchedule(
54   address _account,
55   uint256 _totalAmount,
56   uint256 _tgePerc, //[+] no verification 31a tge percent
57   uint256 _cliff,
58   uint256 _vestingPrc,
59   uint256 _duration
60        ) external onlyOwner {
61    require(_account!= address(0), "Address Zero");
```

```
62            require(vestingSchedules[_account].totalAmount == 0, "Vesting
                  ↪  already exists");
63            require(
64     _totalAmount > 0 , "Invalid Vesting Amount");
65            require(_vestingPrc <= DENOMINATOR , "Invalid Vesting
                  ↪ Percentage");
66    vestingSchedules[_account] = VestingSchedule(_totalAmount,_tgePerc,
         ↪ _cliff,_vestingPrc,_duration,0);
67     }
```

Recommendation:

Consider verifying createVestingSchedule to be less than the DENOMINATOR.

Updates

The team added the verification using the following line:

SHB.2.2: VestingContract.sol

```
require(_tgePerc <= DENOMINATOR , "Invalid Vesting Percentage");
```

4 Best Practices

BP.1 Remove Unused Variables/Functions

Description:

In the AcePoker.sol, we remark the definition of the variable MAX while the MAX variable does not appear to pose a direct security threat in this particular context, it is still considered a best practice to remove any unused variables from the codebase to enhance code readability and maintainability. Same for the withdrawETH,withdrawToken, receive functions since all liquidity is minted to the vesting contract we don't have any tokens left in the contract to withdraw nor withdraw ETH since no users will send ETH to the contract.

Files Affected:

BP.1.1: AcePoker.sol

```
10 uint256 private constant MAX = ~uint256(0);
```

BP.1.2: AcePoker.sol

```
22 function withdrawETH() external onlyOwner {
23         (bool success, )=address(owner()).call{value: address(this).
               ↪ balance}("");
24         require(success, "Failed in withdrawal");
25     }
26     function withdrawToken(address token) external onlyOwner{
27         require(address(this) != token, "Not allowed");
28         IERC20(token).safeTransfer(owner(), IERC20(token).balanceOf(
               ↪ address(this)));
29     }
30     receive() external payable {}
```

Status - Fixed

BP.2 Remove The Hardhat Console In Production

Description:

In the VestingContract.sol , we remark the import of the console from the hardhat library, since the contract is deployed, we are already in the production phase. Remove the hardhat console import before deploying the contract in production.

Files Affected:

BP.2.1: VestingContract.sol

```
6    import "hardhat/console.sol"
```

Status - Fixed

BP.3 Add Released Verification in the getReleasableAmount function

Description:

In the getReleasableAmount function, consider adding the verification of vestingSchedule.totalAmount == vestingSchedule.released.

Files Affected:

BP.3.1: VestingContract.sol

```
85  if (currentTime < startDate || vestingSchedule.totalAmount == 0 ) {
```

Status - Fixed

5 Tests

Results:

→ AcePoker & VestingContract Tests (14 passing)

✓ should verify vesting contract instance

✓ should create team vesting schedules for all categories by the admin wallet (73ms)

✓ should create airdrop & bounties vesting schedules for all categories by the admin wallet

✓ should create community vesting schedules for all categories by the admin wallet

✓ should create strategic partnerships vesting schedules for all categories by the admin wallet (56ms)

✓ should create liquidity vesting schedules for all categories by the admin wallet

✓ should create marketing vesting schedules for all categories by the admin wallet

✓ should allow the owner wallet to send tokens to launchpad contracts (67ms)

✓ should get a zero releasable amount when the current time is before the vesting start date

✓ should get the releasable amount when the current time is equal to the vesting start date (e.g., liquidity accounts) (85ms)

- ✓ should get a zero releasable amount if the vesting schedule does not exist
- ✓ should get a zero releasable amount when the current time is before the vesting cliff
- ✓ should get the releasable amount when the current time is at or after the vesting cliff (e.g., team account) (87ms)
- ✓ should get the releasable amount when the current time is at or after three vesting periods (e.g., marketing account) (92ms)

Coverage:

The code coverage results were obtained by running `npx hardhat coverage` in the `ace-poker` project. We found the following results :

- Statements Coverage : 93.94%
- Branches Coverage : 54.76%
- Functions Coverage : 80%
- Lines Coverage : 95.83%

6 Conclusion

We examined the design and implementation of Ace Poker in this audit and found several issues of various severities. We advise Ace Poker team to implement the recommendations contained in all 2 of our findings to further enhance the code's security. It is of utmost priority to start by addressing the most severe exploit discovered by the auditors then followed by the remaining exploits, and finally we will be conducting a re-audit following the implementation of the remediation plan contained in this report.

We would much appreciate any constructive feedback or suggestions regarding our methodology, audit findings, or potential scope gaps in this report.

7 Scope Files

7.1 Audit

Files	MD5 Hash
contracts/VestingContract.sol	1632442a140f168b3bca45a2ef585df8
contracts/AcePoker.sol	85b29f8a21949acdb1683cbeee8de3f4

7.2 Re-Audit

Files	MD5 Hash
contracts/VestingContract.sol	8e7b6bc5bd1b18e4b8f761f877aa9c79
contracts/AcePoker.sol	77db0a1bd6f4040320c94bd45628dad9

8 Disclaimer

Shellboxes reports should not be construed as "endorsements" or "disapprovals" of particular teams or projects. These reports do not reflect the economics or value of any "product" or "asset" produced by any team or project that engages Shellboxes to do a security evaluation, nor should they be regarded as such. Shellboxes Reports do not provide any warranty or guarantee regarding the absolute bug-free nature of the examined technology, nor do they provide any indication of the technology's proprietors, business model, business or legal compliance. Shellboxes Reports should not be used in any way to decide whether to invest in or take part in a certain project. These reports don't offer any kind of investing advice and shouldn't be used that way. Shellboxes Reports are the result of a thorough auditing process designed to assist our clients in improving the quality of their code while lowering the significant risk posed by blockchain technology. According to Shellboxes, each business and person is in charge of their own due diligence and ongoing security. Shellboxes does not guarantee the security or functionality of the technology we agree to research; instead, our purpose is to assist in limiting the attack vectors and the high degree of variation associated with using new and evolving technologies.

For a Contract Audit, contact us at contact@shellboxes.com