QuillAudits

Audit Report
November, 2022

For

XT Fund

Table of Content

Table of Content

Table of Content

Executive Summary

Project Name	X11Fund Contracts Initial Audit Report
Overview	This is a Defi based project where a user can invest BUSD stable coins to earn BUSD reward as well he receives a NFT for investing, which he can further stake to earn x11 tokens as reward
Timeline	23th August,2022 - 10th November,2022
Method	Manual Review, Functional Testing, Automated Testing.
Scope of Audit	The scope of this audit was to analyze x11Fund codebase for quality, security, and correctness. *https://github.com/x11-finance/X11Fund-Smart-Contracts/tree/main/contracts*
Fixed In	*https://github.com/x11-finance/X11Fund-Smart-Contracts/tree/main/contracts* **Commit Hash:** b936e3c686aa5fd8a98d757f3def5c104467b6c5

High Medium

Low Informational

	High	Medium	Low	Informational
Open Issues	0	0	0	0
Acknowledged Issues	4	4	0	3
Partially Resolved Issues	0	0	0	0
Resolved Issues	5	2	5	20

Types of Severities

High

A high severity issue or vulnerability means that your smart contract can be exploited. Issues on this level are critical to the smart contract's performance or functionality, and we recommend these issues be fixed before moving to a live environment.

Medium

The issues marked as medium severity usually arise because of errors and deficiencies in the smart contract code. Issues on this level could potentially bring problems, and they should still be fixed.

Low

Low-level severity issues can cause minor impact and or are just warnings that can remain unfixed for now. It would be better to fix these issues at some point in the future.

Informational

These are severity issues that indicate an improvement request, a general question, a cosmetic or documentation error, or a request for information. There is low-to-no impact.

Types of Issues

Open

Security vulnerabilities identified that must be resolved and are currently unresolved.

Resolved

These are the issues identified in the initial audit and have been successfully fixed.

Acknowledged

Vulnerabilities which have been acknowledged but are yet to be resolved.

Partially Resolved

Considerable efforts have been invested to reduce the risk/impact of the security issue, but are not completely resolved.

Checked Vulnerabilities

- Re-entrancy
- Timestamp Dependence
- Gas Limit and Loops
- Exception Disorder
- Gasless Send
- Use of tx.origin
- Compiler version not fixed
- Address hardcoded
- Divide before multiply
- Integer overflow/underflow
- Dangerous strict equalities
- Tautology or contradiction
- Return values of low-level calls
- Missing Zero Address Validation
- Private modifier
- Revert/require functions
- Using block.timestamp
- Multiple Sends
- Using SHA3
- Using suicide
- Using throw
- Using inline assembly

Techniques and Methods

Throughout the audit of smart contract, care was taken to ensure:

- The overall quality of code.
- Use of best practices.
- Code documentation and comments match logic and expected behaviour.
- Token distribution and calculations are as per the intended behaviour mentioned in the whitepaper.
- Implementation of ERC-20 token standards.
- Efficient use of gas.
- Code is safe from re-entrancy and other vulnerabilities.

The following techniques, methods and tools were used to review all the smart contracts.

Structural Analysis

In this step, we have analysed the design patterns and structure of smart contracts. A thorough check was done to ensure the smart contract is structured in a way that will not result in future problems.

Static Analysis

Static analysis of smart contracts was done to identify contract vulnerabilities. In this step, a series of automated tools are used to test the security of smart contracts.

Code Review / Manual Analysis

Manual analysis or review of code was done to identify new vulnerabilities or verify the vulnerabilities found during the static analysis. Contracts were completely manually analysed, their logic was checked and compared with the one described in the whitepaper. Besides, the results of the automated analysis were manually verified.

Gas Consumption

In this step, we have checked the behaviour of smart contracts in production. Checks were done to know how much gas gets consumed and the possibilities of optimization of code to reduce gas consumption.

Tools and Platforms used for Audit

Remix IDE, Truffle, Truffle Team, Solhint, Mythril, Slither, Solidity statistic analysis.

Manual Testing

A. Contract - X11 contract

High Severity Issues

No issues found

Medium Severity Issues

A.1 Missing mint functionality in X11 token

Line	Function -

1	`// SPDX-License-Identifier: MIT`
	`pragma solidity >=0.4.22 <0.9.0;`
	
	`import "@openzeppelin/contracts/token/ERC20/ERC20.sol";`
	`import`
	`"@openzeppelin/contracts/token/ERC20/extensions/ERC20Burnable.`
	`sol";`
	`contract X11 is ERC20, ERC20Burnable {`
	`  constructor() ERC20("X11 Token", "X11") {`
	`  _mint(msg.sender, 1000000000*10e18);`
	`  }`
	`}`

Description

X11 Token contract does not have mint feature. Just mark it as acknowledged if this is the required behaviuor and the total supply is capped.

Remediation

mintable feature can be added using mintable extension of ERC20 implemented by openzepplin.

Status

Acknowledged

Auditors' Comment: X11 team acknowledged that this is the desired behavior, total supply is capped.

Low Severity Issues

No issues found

Informational Issues

No issues found

B. Contract - X721 Contract

High Severity Issues

No issues found

Medium Severity Issues

No issues found

Low Severity Issues

B.1 poolId and tokenId should also be indexed

Line	Function -
30	`event Minted(address indexed user, uint256 poolId, uint256 amount, uint256 tokenId);`

Description

ndexed fields help the backend listen to the events. Unique and important fields should be indexed for easier search on the backend.

Status

Resolved

Informational Issues

B.2 Inefficient implementation of _getBalancePool

Line	

169	function getBalanceInPool(uint256 _poolId) public view returns (uint256) {
	uint256 balanceInPool = 0;
	for (uint256 i = 0; i < tokens.length; i++) {
	if (ownerOf(tokens[i]) == msg.sender && tokensData[tokens[i]].poolId == _poolId) {
	balanceInPool += tokensData[tokens[i]].amount;
	}
	}
	
	return balanceInPool;
	}

Description

Loop is used in getBalanceInPool method to find the token which belong to caller. List of token owned by a user can be gathered by using ERC721Enumerable contract.

Remediation

getBalanceInPool can be implemented in more efficient way by using ERC721Enumerable features.

Status

Resolved

Auditors' Comment: X11 team has removed this functionality as it is not being used in the application.

B.3 Redundant Inheritance

Line	computeClaimableTokens()
15	`contract X721 is ERC721Enumerable, ERC721Burnable, ERC721URIStorage, Pausable, Ownable, AccessControl { // /* ERC721Burnable, ERC721URIStorage */ using Counters for Counters.Counter;`

Description

X721 Contract inherits both Ownable and AccessControl contracts. All the features of Ownable contract can be achieved using AccessControl by defining different roles.

Remediation

Contract Ownable can be removed from inheritance list.

Status

Resolved

B.4 Unnecessary Variable

Line	computeClaimableTokens()
15	`uint256[] tokens; _tokenIdCounter.increment(); uint256 newItemId = _tokenIdCounter.current(); tokensData[newItemId] = Metadata2(_poolId, _amount); tokens.push(newItemId);`

Description

Variable tokens is being used to store newItemId which is being incremented with each new minted X721. So each element of array tokens is equal to its own index. Length of the tokens will be equal to the length of array tokensData. So the tokens array does not carry any additional information. This aray can be removed and that will save unnecessary storage allocation and computation.

Remediation

Variable tokens can be removed from the codebase

Status

Resolved

B.5 method _setTokenURI is not being used

Line	computeClaimableTokens()
95	`_setTokenURI(newItemId, formatTokenURI(_poolId, _amount));`

Description

This method is setting the tokenURI but it is not being used anywhere. There is no implementation of getter method to read TokenURI. No need to set the tokenURI if it is not being used. This will consume extra gas.

Status

Resolved

B.6 Unused Import

Line	computeClaimableTokens()
4	`import "@openzeppelin/contracts/token/ERC721/ERC721.sol";`

Description

ERC721.sol imported but not used

Status

Resolved

B.7 Unused Import

Line	computeClaimableTokens()
15	`contract X721 is ERC721Enumerable, ERC721Burnable, ERC721URIStorage, Pausable, Ownable, AccessControl { // /* ERC721Burnable, ERC721URIStorage */` `using Counters for Counters.Counter;`

Description

_burn function is declared and implantation overrides the implementation of ERC721 and ERC721URIstorage. So Burn function can't be used. If intended behavior is not to burn method then ERC721Burnable should not be inherited.

Status

Resolved

Auditors' Comment: X11 team has updated the code

B.8 Unused code

Line	batchMint(), _batchTransfer

129	function formatTokenURI2(uint256 _poolId, uint256 _amount)
	public pure returns (string memory) {
	string memory json =
	Base64.encode(bytes(string(abi.encodePacked('{"name": "xUSD
	', _poolId, '", "description": "xUSD is a tokenized
	stablecoin that represents the value of USD. It is backed by a
	basket of stablecoins and can be redeemed for USD at any
	time.", "image": "https://xusd.finance/images/xUSD.png",
	"attributes": [{"trait_type": "Pool ID", "value": "', _poolId,
	'"}, {"trait_type": "Amount", "value": "', _amount, '"}]
	'))));
	string memory output =
	string(abi.encodePacked('data:application/json;base64,',
	json));
	return output;
	}

Description

formatTokenURI2 has been declared and implemented but not being used anywhere in the codebase.

Remediation

formatTokenURI2 can be removed from the codebase.

Status

Acknowledged

Auditors' Comment: Removed from the code base.

C. Contract - Fund

High Severity Issues

C.1 Vote calculation logic flaw

Line	Function -
303	function getVotes(uint256 _poolId) public view returns (uint256) { uint256 totalVotes = 0; for (uint256 i = 0; i < votes.length; i++) { if (votes[i].poolId == _poolId && votes[i].vote) { totalVotes++; } } return totalVotes; }

Description

Total votes count does not involve the weight of the vote. User A who stakes 10000 BUSD five times gets 5 votes when compared with user B who stakes 10000 BUSD in a single go and gets a single vote only.

Remediation

Status

Resolved

Auditors' Comment: All the Voting related code implementation has been removed from the codebase. This resolves the issue but there will be a change in business logic.

C.2 _castVote implementation flaw

Line	Function -

330	`function _castVote(uint256 _poolId, bool _vote) internal {`
	`  votes.push(Vote(msg.sender, _poolId, _vote));`
	`  canVote[msg.sender][_poolId].isTrue = false;`
	`}`

Description

At line 330 canVote corresponding to a wrong tokenId is set to false.

Remediation

Status

Resolved

Auditors' Comment: All the Voting related code implementation has been removed from the codebase. This resolves the issue but there will be a change in business logic.

C.3 DoS attack

Line

169

```
function emergencyWithdrawRewardsToAdmin(uint256 _poolId,
uint256 _busdStakesAmount) public onlyOwner {
   require(pools[_poolId].funded >= 0);

   // this loop will fail if the gas usage is more than block
gas limit. DoS attack
   for (uint256 i = 0; i < _busdStakesAmount; i++) {
     if (busdStakes[i].poolId == _poolId) {
       uint256 unclaimedReward = busdStakes[i].rewards -
busdStakes[i].claimedRewards;
       busdStakes[i].claimedRewards += unclaimedReward;
       busd.transferFrom(address(this), msg.sender,
unclaimedReward);
     }
   }
   pools[_poolId].funded = 0;
   emit EmergencyWithdrawn(_poolId, block.timestamp);
 }
function closePool(uint256 _poolId, uint256 _busdStakesAmount,
uint256 _totalStakedInPool) public onlyOwner {
   require(pools[_poolId].funded > 0, "Please fund the pool
first.");
   // this loop will fail if the gas usage is more than block
gas limit. DoS attack. Tokens will get stuck
   for (uint256 i = 0; i < _busdStakesAmount; i++) {
     if (busdStakes[i].poolId == _poolId) {
       busdStakes[i].tokenamount -= 2 *
busdStakes[i].tokenamount / 100;
       uint256 part = busdStakes[i].tokenamount * 10e8 /
_totalStakedInPool;
       uint256 cashout = pools[i].funded * part / 10e8;
       busd.transferFrom(address(this), busdStakes[i].from,
cashout);
     }
   }

   pools[_poolId].isActive = false;
 }
```

Description

The for loop in these methods will run through all the stakes in all the pools. This array can be huge and that can lead to DoS attack due to the block gas limit.

Status

Resolved

Auditors' Comment: The reason for DoS attack is the use of a loop inside the function. emergencyWithdrawRewardsToAdmin is rewritten. No check for if the pool is funded or not. closePool still has the same implementation but the X11 team conveyed that it will only be called by the admin, who will take care of gas limits.

C.4 transferFrom flaw

Line	Function -
398, 463	`busd.transferFrom(address(this), msg.sender, unclaimedReward);` `busd.transferFrom(address(this), busdStakes[i].from, cashout);`

Description

transferFrom is used by a third party to transfer balance on behalf of someone who has provided approval to that third party. Here transferFrom is being used to transfer from the contract itself which will fail as there is no approv

Remediation

Use simple transfer here

Status

Resolved

Auditors' Comment: All the Voting related code implementation has been removed from the codebase. This resolves the issue but there will be a change in business logic.

C.5 Close pool check

Line	Function -

411, 457	function fundPool(uint256 _poolId, uint256 _tokenamount) public onlyOwner {
	require(_tokenamount <= GetBUSDAllowance(), "Please approve tokens before transferring.");
	_fundPool(_poolId, _tokenamount);
	}
	function closePool(uint256 _poolId, uint256 _busdStakesAmount, uint256 _totalStakedInPool) public onlyOwner {
	require(pools[_poolId].funded > 0, "Please fund the pool first.");

Description

As per discussion on telegram, the FundPool method should only be callable when the pool is closed. Should add a check on contract and remove the check from closePool.

Status

Acknowledged

Auditors' Comment: X11 team conveyed that information given earlier was by mistake. There is no such a limitation. ClosePool() can be called after the pool is funded. However, these functions will be called only by the admin and the timing of calling will be decided by the admin. No condition from a business point of view has been included in smart contracts.

Medium Severity Issues

C.6 Update reward method should be called from withdrawBUSDRewardWithToken

Line	Function -

262	`function updateReward(uint256 _tokenId) public {`
	`  uint256 poolId = x721.getPoolId(_tokenId);`
	`  uint256 totalStakedInCurrentPool = stakedInThePool[poolId];`
	`  address stakerAdr = x721.ownerOf(_tokenId); // not used`
	`  uint256 stakedAmount = x721.peggedAmount(_tokenId); // not used`
	`  BUSDStake storage stake = tokenToStake[_tokenId];`
	
	`  uint256 part = stake.tokenamount * 10e8 / totalStakedInCurrentPool;`
	`  stake.rewards = pools[poolId].funded * part / 10e8;`
	
	`  emit RewardUpdated(poolId, _tokenId);`
	`}`

Description

to BUSD reward with token a user will have to make two transactions if these two methods are separate

Remediation

Combine these methods and add a view method that calculates the user's current rewards free of charge.

Status

Resolved

Low Severity Issues

C.7 tokenId and poolId should also be indexed

Line	

99	event Staked(address indexed user, uint256 tokenId, uint256 amount, uint256 poolIndex, uint256 indexInPool, uint256 timestamp);
	event StakedInit(address indexed user, uint256 amount, uint256 poolIndex, uint256 indexInPool, uint256 timestamp);
	event Unstaked(address indexed user, uint256 amount, uint256 poolIndex, uint256 timestamp);
	event UnstakedInit(address indexed user, uint256 amount, uint256 poolIndex, uint256 timestamp);
	event EmergencyWithdrawn(uint256 poolIndex, uint256 timestamp);
	event RevenueWithdrawn(uint256 poolIndex, uint256 timestamp, address to, uint256 tokenId);
	event PoolFunded(uint256 poolId, uint256 tokenamount);
	event PoolCreated(uint256 poolId, string name, string description, string companies, string setStarts, string setEnds);
	event RewardUpdated(uint256 poolId, uint256 tokenId);
	event RewardsUpdated();

Description

Indexed fields help the backend listen to the events. Unique and important fields should be indexed for easier search on the backend. Also here poolId is used and also poolIndex for a pool's Id which is confusing.

Status

Resolved

Informational Issues

C.8 Contract design limitation

Line	

Description

Contract is designed to only support BUSD and cannot support a new stablecoin after deployment, contracts are not upgradeable as well

Status

Acknowledged

Auditors' Comment: X11 team conveyed that "This is by design, and there will be no support for other stablecoins in v1."

C.9 Unnecessary variables

Line	

18	`struct Stake {`
	`  address from;`
	`  uint256 tokenamount;`
	`  uint256 since;`
	`  uint256 poolId;// no need of this as Stake is already mapped from poolId`
	`  uint256 rewards;`
	`  uint256 claimedRewards;`
	`}`

Description

No need of poolId as Stake is already mapped from poolId. poolId occupies an unnecessary storage slot and hence costs gas.

Status

Resolved

C.10 Unused variables

Line	

23, 32, 64	`uint256 rewards;`
	
	`uint256 since; //unused`
	`uint256 since;//unused`

Description

rewards in Stake struct, since in BUSDStake and CanVote are not used anywhere. These variables occupy unused storage slots and hence cost gas.

Status

Resolved

C.11 Unnecessary variables that can be maintained on backend

Line	

77,79,80, 90	`uint256 totalInitStakes = 0;`
	
	`mapping (uint256 => uint256) stakersInThePool;`
	
	`mapping (uint256 => uint256) stakedInThePool;`
	
	`uint256 totalInvestments;`

Description

No need to maintain this on contract as it costs storage and can be maintained on backend instead by listening to events.

Status

Acknowledged

Auditors' Comment: X11 team conveyed that "We have no backend, Smart Contract is a backend..

C.12 Unnecessary require checks

Line	
192	`function claimInitStakeFromPool(uint256 _poolId, uint256 _idInPool) public returns(bool) {` `  require(_idInPool >= 0, "You're not in this pool!"); // no need of this check _idInPool is a uint`

Description

No need to check as _idInPool is a unit and will always >= 0

Status

Acknowledged

Auditors' Comment: X11 team conveyed that "We have no backend, Smart Contract is a backend..

C.13 Unused variables inside methods

Line	
265, 266, 285, 286, 287	

```
function updateReward(uint256 _tokenId) public {
   uint256 poolId = x721.getPoolId(_tokenId);
   uint256 totalStakedInCurrentPool = stakedInThePool[poolId];
   address stakerAdr = x721.ownerOf(_tokenId); // not used
   uint256 stakedAmount = x721.peggedAmount(_tokenId); // not
used
   BUSDStake storage stake = tokenToStake[_tokenId];

   uint256 part = stake.tokenamount * 10e8 /
totalStakedInCurrentPool;
   stake.rewards = pools[poolId].funded * part / 10e8;

   emit RewardUpdated(poolId, _tokenId);
 }
function withdrawBUSDRewardWithToken(uint256 _tokenId) public
nonReentrant {
   uint256 poolId = x721.getPoolId(_tokenId);
   require(pools[poolId].funded >= 0);
   require(x721.ownerOf(_tokenId) == msg.sender);

   uint256 totalStakedInCurrentPool =
stakedInThePool[poolId];// not used
   address stakerAdr = x721.ownerOf(_tokenId);// not used
   uint256 stakedAmount = x721.peggedAmount(_tokenId);// not
used
   BUSDStake storage stake = tokenToStake[_tokenId];

   uint256 unclaimedReward = stake.rewards -
stake.claimedRewards;
   stake.claimedRewards += unclaimedReward;

   busd.transfer(stake.from, unclaimedReward);

   emit RevenueWithdrawn(poolId, block.timestamp, msg.sender,
_tokenId);
 }
```


Description

unused variables inside methods. Unnecessary computations

Status

Resolved

C.14 Unused Code

Line	Function -

303	`struct Company{`
	`  string index;`
	`  string name;`
	`}`

Description

Structure Company is declared but not being used anywhere.

Remediation

Status

Resolved

C.15 Unused Function

Line	Function -

154	`function getPoolInfoAdmin(uint _i) public view onlyOwner`
	`returns (Pool memory) {`
	`  return pools[_i];`
	`}`

Description

No use of getPoolInfoAdmin method as getPoolInfo already defined.

Remediation

Status

Resolved

D. Contract - ERC721Staking

High Severity Issues

D.1 _unstake logic flaw

Line

160

```
function _unstake(address _user, uint256 _tokenId) internal {
      require(tokenOwner[_tokenId] == _user, "User must own
the token.");
      Stake storage _stake = stakes[_user];

      uint256 lastIndex = _stake.tokenIds.length - 1;
      uint256 lastIndexKey = _stake.tokenIds[lastIndex];
//unused

      if (_stake.tokenIds.length > 0) {
          _stake.tokenIds.pop();
      }

      if(_stake.balance == 0) {
          delete stakes[_user];
      }
      delete tokenOwner[_tokenId];

      stakedToken.safeTransferFrom(address(this), _user,
 tokenId);
_tokenId);
      totalStaked--;

      emit Unstaked(_user, _tokenId);

  }
```

Description

Logical flaw. Last tokenId is being popped from the tokenIds array even if the tokenId being unstaked is some other index on the array. Also tokenIds is being popped but the since variable corresponding to that tokenId is not popped.

Status

Resolved

Auditors' Comment: X11 team has Resolved the issue. Code has been updated in the next version.

D.2 _unstake balance flaw

Line

160

```
function _unstake(address _user, uint256 _tokenId) internal {
        require(tokenOwner[_tokenId] == _user, "User must own
the token.");
        Stake storage __stake = stakes[_user];

        uint256 lastIndex = __stake.tokenIds.length - 1;
        uint256 lastIndexKey = __stake.tokenIds[lastIndex];
//unused

        if (__stake.tokenIds.length > 0) {
            __stake.tokenIds.pop();
        }

        if(__stake.balance == 0) {
            delete stakes[_user];
        }

        delete tokenOwner[_tokenId];

        stakedToken.safeTransferFrom(address(this), _user,
_tokenId);
        totalStaked--;

        emit Unstaked(_user, _tokenId);

    }
```

Description

_stake.balance is never subtracted when a token is unstaked.

Status

Acknowledged

Auditors' Comment: X11 team has Resolved the issue. Code has been updated in the next version.

D.3 updateReward logic flaw

Line

188

```
function updateReward(address _user) public {
        Stake storage __stake = stakes[_user];
        uint256[] storage ids = __stake.tokenIds;

        for (uint256 j = 0; j < ids.length; j++) {
            uint256 stakedDays = ((block.timestamp -
uint(__stake.since[j]))) / stakingTime;
            uint256 tier = getInvestmentTier(ids[j]);
            if (j == 0) {
                __stake.rewards = 0; //
            }
            uint256 tokenRewards =
stakedToken.peggedAmount(ids[j]) * stakedDays
                * tier * 10e18 * 10e7 / (x11RateToUSD * 100);
            __stake.rewards += tokenRewards;
        }
    }
```

Description

tokenRewards calculated will be flawed as stakedToken.peggedAmount already has 10**18 in it which is not considered in this calculation.

Status

Acknowledged

Auditors' Comment: X11 team conveyed that "StakedToken.peggedAmount doesn't have a 10**18 multiplier. You can see the test code for reference."

D.4 getInvestmentTier logic flaw

Line	

160	function getInvestmentTier(uint256 _tokenId) internal view returns (uint256) {
	uint256 peggedAmount = stakedToken.peggedAmount(_tokenId);
	if (peggedAmount < 5000) {
	return uint256(5) * uint256(10e8) / uint256(365);
	} else if (peggedAmount >= 5000 && peggedAmount < 15000) {
	return uint256(8) * uint256(10e8) / uint256(365);
	} else if (peggedAmount >= 15000 && peggedAmount < 30000) {
	return uint256(10) * uint256(10e8) / uint256(365);
	} else if (peggedAmount >= 30000 && peggedAmount < 50000) {
	return uint256(15) * uint256(10e8) / uint256(365);
	} else if (peggedAmount >= 50000 && peggedAmount < 70000) {
	return uint256(20) * uint256(10e8) / uint256(365);
	} else if (peggedAmount >= 70000) {
	return uint256(30) * uint256(10e8) / uint256(365);
	}
	return uint256(0);
	}

Description

Logical flaw. Pegged amount has 10**18 multiplied into itself so here always the logic will go into the last else if.

Status

Acknowledged

Auditors' Comment: X11 team conveyed that "StakedToken.peggedAmount doesn't have a 10**18 multiplier. You can see the test code for reference."

Medium Severity Issues

D.5 cannot stop staking

Line	Function -

67	`function initStaking() public onlyOwner {`
	`require(!initialised, "Already initialized");`
	`stakingStartTime = block.timestamp;`
	`initialised = true;`
	`}`

Description

can only set the initialised to true cannot set it back to false

Status

Acknowledged

Auditors' Comment: X11 team conveyed that "This is the desired behavior."

D.6 Unstake one leads to claimReward for all

Line	Function -

67	`function unstake(uint256 _tokenId) public nonReentrant {`
	`claimReward(msg.sender); // No need of passing msg.sender here`
	`_unstake(msg.sender, _tokenId);`
	`}`

Description

Unstaking one tokenId leads to claiming rewards for all the tokenIds. Is this the required behavior here? If this is the required behavior please mark this as acknowledged.

Status

Acknowledged

D.7 Use of for loop in updateReward method

Line	

214	`for (uint256 j = 0; j < ids.length; j++) {`
	`uint256 stakedDays = ((block.timestamp -`
	`uint(__stake.since[j]))) / stakingTime;`
	`uint256 tier = getInvestmentTier(ids[j]);`
	`if (j == 0) {`
	`__stake.rewards = 0; //`
	`}`
	`uint256 tokenRewards =`
	`stakedToken.peggedAmount(ids[j]) * stakedDays`
	`* tier * 10e18 * 10e7 / (x11RateToUSD * 100);`
	`// these rewards will be too much, why no division`
	`__stake.rewards += tokenRewards;`
	`}`

Description

For loop can lead to DoS attack due to block gas limit if ids array gets too long

Status

Resolved

D.8 setRateToUSD should be set using oracle service

Line	

214	`function setRateToUSD(uint256 _rate) public onlyOwner {`
	`x11RateToUSD = _rate;`
	`}`

Description

This price should not be set by owner instead should be set using chainlink price oracles.

Remediation

Combine these methods and add a view method that calculates the user's current rewards free of charge.

Status

Acknowledged

Low Severity Issues

D.9 tokenId should also be indexed

Line	

46,47,49	`event Staked(address indexed user, uint256 amount, uint256 tokenId);`
	`event Unstaked(address indexed user, uint256 tokenId);`
	`event EmergencyUnstake(address indexed user, uint256 tokenId);`

Description

Indexed fields help the backend listen to the events. Unique and important fields should be indexed for easier search on the backend.

Status

Resolved

D.10 Unnecessary require in emergencyUnstake method

Line	

146	`function emergencyUnstake(uint256 _tokenId) public {`
	`require(`
	`tokenOwner[_tokenId] == msg.sender,`
	`"nft._unstake: Sender must have staked tokenID"`
	`);`
	`_unstake(msg.sender, _tokenId);`
	`emit EmergencyUnstake(msg.sender, _tokenId);`
	`}`

Description

the require in this method is already available in _unstake method.

Status

Resolved

D.11 unnecessary parameter in claimReward method

Line	
240	`function claimReward(address _user) public returns (uint256) {`

Description

This function will work even without passing the user parameter when called both as externally and internally

Remediation

Combine these methods and add a view method that calculates the user's current rewards free of charge.

Status

Resolved

Informational Issues

D.12 Unused storage variables

Line	

26	`struct Stake {`
	`  uint256[] tokenIds;`
	`  uint256[] since;`
	`  uint256 balance;`
	`  uint256 rewards;`
	`  uint256 claimedRewards;`
	`  address from;// unused`
	`}`

Description

from variable is unused

Status

Resolved

D.13 Unnecessary variables

Line	
18	`mapping (uint256 => address) tokenOwner;`

Description

No need of this as already set in xUSD token contract. Unnecessary storage and computation

Status

Resolved

D.14 Unnecessary variables that can be maintained on backend

Line	

30,31,38	`uint256[] stakedTokens;`
	`address[] users;`
	`uint256 public totalStaked;`

Description

No need to maintain this on contract as it costs storage and can be maintained on backend instead by listening to events.

Status

Resolved

D.15 Unused Inheritance

Line	
30,31,38	`contract ERC721Staking is ERC721Holder, ReentrancyGuard, Ownable, Pausable {`

Description

Pausable not used anywhere

Status

Resolved

D.16 Methods can be set from public to external

Line	
192	`function setTokensClaimable(bool _isEnabled) public onlyOwner`

Description

should be set to external to save gas

Status

Resolved

D.17 Unused Imports

Line	
4	`import "@openzeppelin/contracts/token/ERC721/IERC721.sol";`

Description

IERC721 interface imported but not used

Status

Resolved

D.18 Unused variables

Line	

4, 165	`uint256 constant token = 10e18;`
	
	`uint256 lastIndexKey = __stake.tokenIds[lastIndex];`

Description

Not used anywhere

Status

Resolved

D.19 Confusing variable name

Line	
40	`uint256 constant stakingTime = 1 days;`

Description

stakingTime name is confusing. Constants should be declared in capital letters.

Status

Resolved

Functional Testing

Contract: Fund

- ✓ should deploy
- ✓ should add a pool
- ✓ shouldn't add a pool with non-deployer
- ✓ should allow the user to add an init stake
- ✓ should not allow to add init stake twice
- ✓ should allow the user to add an init stake in another pool
- ✓ shouldn't allow to withdraw init stake until a month has passed
- ✓ should allow to withdraw init stake after a month has passed
- ✓ should allow the user to add a BUSD stake
- ✓ should allow the user to add a BUSD stake in another pool
- ✓ shouldn't allow to add a BUSD stake before init stake
- ✓ shouldn't allow non-admin to start voting
- ✓ should allow members to vote
- ✓ shouldn't allow non-members to vote
- ✓ should fund the pool
- ✓ shouldn't allow members to vote when the voting is closed
- ✓ should calculate, update and distribute the rewards
- ✓ shouldn't allow the user to add a BUSD stake in the amount of less than 1K

Contract: X11

- ✓ has the correct name
- ✓ has the correct symbol
- ✓ has the correct decimals
- ✓ has the correct total supply
- ✓ has the correct balance of the owner
- ✓ has the correct balance of another account
- ✓ should transfer the correct amount of tokens
- ✓ should not transfer tokens without approval
- ✓ should approve tokens even if the balance is zero
- ✓ should not transfer from tokens without approval
- ✓ should not transfer from approved tokens
- ✓ should return the balance of token owner
- ✓ should transfer right token
- ✓ should give accounts[1] authority to spend account[0]'s token

Contract: ERC721Staking

- ✓ should deploy
- ✓ should set the correct rate
- ✓ shouldn't accept the stake before staking is started
- ✓ should accept the stake
- ✓ shouldn't allow to withdraw stake and rewards until tokens are set claimable
- ✓ should allow to emergency withdraw stake without carrying about the rewards
- ✓ should not allow to emergency unstake twice
- ✓ should allow to withdraw stake and rewards after tokens are set claimable
- ✓ shouldn't allow to withdraw stake twice
- ✓ should allow one person to stake more than once
- ✓ should not accept the stake without approval
- ✓ should not allow non-owner to withdraw stake
- ✓ should distribute rewards

Contract: X721

- ✓ has the correct name
- ✓ has the correct symbol
- ✓ is possible to mint tokens for the minter role
- ✓ is not possible to mint tokens for non-minter role
- ✓ can retrive balance of tokens of a user and can trace the owner
- ✓ can transfer tokens between users

Automated Tests

No major issues were found. Some false positive errors were reported by the tools. All the other issues have been categorized above according to their level of severity.

Closing Summary

In this report, we have considered the security of the X11 Fund contracts. We performed our audit according to the procedure described above.

Some issues of High, Medium, Low and informational severity were found, Some suggestions and best practices are also provided in order to improve the code quality and security posture.

Disclaimer

QuillAudits smart contract audit is not a security warranty, investment advice, or an endorsement of the X11 Fund Platform. This audit does not provide a security or correctness guarantee of the audited smart contracts.

The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them. Securing smart contracts is a multistep process. One audit cannot be considered enough. We recommend that the X11 Fund Team put in place a bug bounty program to encourage further analysis of the smart contract by other third parties.

About QuillAudits

QuillAudits is a secure smart contracts audit platform designed by QuillHash Technologies. We are a team of dedicated blockchain security experts and smart contract auditors determined to ensure that Smart Contract-based Web3 projects can avail the latest and best security solutions to operate in a trustworthy and risk-free ecosystem.

700+
Audits Completed

$15B
Secured

700K
Lines of Code Audited

Follow Our Journey

Audit Report
November, 2022

For

XY Fund

QuillAudits

Canada, India, Singapore, United Kingdom

audits.quillhash.com

audits@quillhash.com