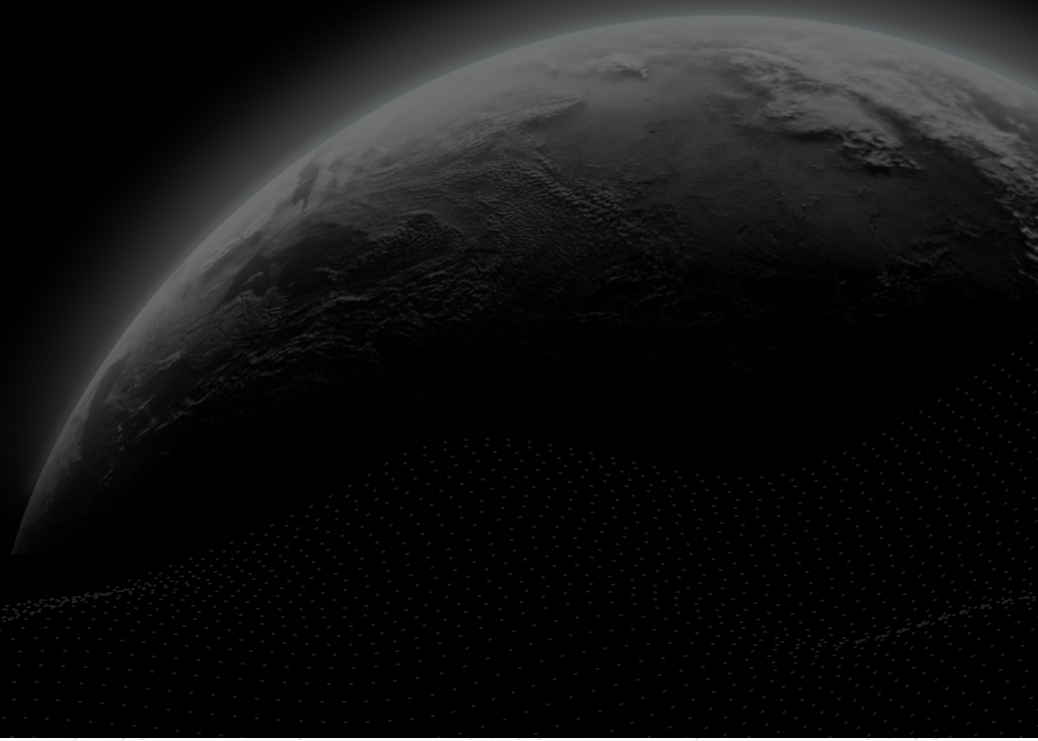
CERTIK

Security Assessment

Draft (Internal Use Only)

Bartrr

CertiK Verified on Sept 26th, 2022



CertiK Verified on Sept 26th, 2022

Bartrr

The security assessment was prepared by CertiK, the leader in Web3.0 security.

Executive Summary

TYPES	ECOSYSTEM	METHODS
Staking	Ethereum	Manual Review, Static Analysis
LANGUAGE	**TIMELINE**	**KEY COMPONENTS**
Solidity	Delivered on 09/26/2022	N/A

CODEBASE

https://github.com/bartrr-xyz/bartrr-contracts

...View All

Vulnerability Summary

11	10	1	0	0	0	0
Total Findings	Resolved	Mitigated	Partially Resolved	Acknowledged	Declined	Unresolved

3	Critical	3 Resolved	Critical risks are those that impact the safe functioning of a platform and must be addressed before launch. Users should not invest in any project with outstanding critical risks.
2	Major	1 Resolved, 1 Mitigated	Major risks can include centralization issues and logical errors. Under specific circumstances, these major risks can lead to loss of funds and/or control of the project.
0	Medium		Medium risks may not pose a direct risk to users' funds, but they can affect the overall functioning of a platform.
4	Minor	4 Resolved	Minor risks can be any of the above, but on a smaller scale. They generally do not compromise the overall integrity of the project, but they may be less efficient than other solutions.
2	Informational	2 Resolved	Informational errors are often recommendations to improve the style of the code or certain operations to fall within industry best practices. They usually do not affect the overall functioning of the code.

TABLE OF CONTENTS | BARTRR

CODEBASE | BARTRR

Repository

https://github.com/bartrr-xyz/bartrr-contracts

AUDIT SCOPE | BARTRR

5 files audited ● 5 files with Resolved findings

ID	File	SHA256 Checksum
● BBB	projects/Bartrr/BartrrBase.sol	08a85b03e41b12f0a926963a898a67d825a85c29832359b38c0cfe0ac204f42d
● CWB	projects/Bartrr/ConditionalWager.sol	6b6e6ac7191a74c1659000a60328b94180ebef2e4c8adef180b16c253372669d
● FWB	projects/Bartrr/FixedWager.sol	b9b5bb798da47a75b06245f4c03cbef61127af7cac43c0d718aaf1bf59c1515b
● RIF	projects/Bartrr/RoundIdFetcher.sol	d27fff605fde4472fb352264cb0358afdc30b6538a379f0431b2f8f0c24327f6
● TLB	projects/Bartrr/TockenLockup.sol	2525db7a0acf5c3b5d7a8f3522495ef11c4df92e3beba098024ad6d5c2b31947

APPROACH & METHODS | BARTRR

This report has been prepared for Bartrr to discover issues and vulnerabilities in the source code of the Bartrr project as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Manual Review and Static Analysis techniques.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

The security assessment resulted in findings that ranged from critical to informational. We recommend addressing these findings to ensure a high level of security standards and industry practices. We suggest recommendations that could better serve the project from the security perspective:

- Testing the smart contracts against both common and uncommon attack vectors;
- Enhance general coding practices for better structures of source codes;
- Add enough unit tests to cover the possible use cases;
- Provide more comments per each function for readability, especially contracts that are verified in public;
- Provide more transparency on privileged activities once the protocol is live.

FINDINGS | BARTRR

11	3	2	0	4	2
Total Findings	Critical	Major	Medium	Minor	Informational

This report has been prepared to discover issues and vulnerabilities for Bartrr. Through this audit, we have uncovered 11 issues ranging from different severity levels. Utilizing Static Analysis techniques to complement rigorous manual code reviews, we discovered the following findings:

ID	Title	Category	Severity	Status
ABC-01	Users Can Cheat And Never Lose A Game By Front Running	Logical Issue	Critical	Resolved
ABC-02	Possible Drain Of All Funds	Logical Issue	Critical	Resolved
BBB-01	**Centralization Risks In BartrrBase.Sol**	**Centralization / Privilege**	**Major**	**Mitigated**
BCP-01	Possibility To Lock Other Users Funds	Logical Issue	Major	Resolved
BCP-02	Potential Reentrancy Attack	Control Flow	Minor	Resolved
BCP-03	Missing Emit Events	Coding Style	Minor	Resolved
BKP-01	Cancelled Wagers Can Be Filled	Logical Issue	Minor	Resolved
TLA-01	Missing Upper Bound For Lock Duration	Volatile Code	Minor	Resolved
TLB-01	Reentrancy Attack	Logical Issue	Critical	Resolved
BBB-03	Missing Zero Address Validation	Volatile Code	Informational	Resolved
BCP-04	Different Solidity Versions	Language Specific	Informational	Resolved

ABC-01 | USERS CAN CHEAT AND NEVER LOSE A GAME BY FRONT RUNNING

Category	Severity	Location	Status
Logical Issue	Critical	projects/Bartrr/Audit_2/ConditionalWager.sol (Audit2): 327; projects/Bartrr/Audit_2/FixedWager.sol (Audit2): 324~325	Resolved

Description

When the function oracleMalfunction() is called, lost wagers can be refunded. This case only applies when the winner has not redeemed their wager.

For completeness, notice that the function `redeem()` will not revert if the winner has not redeemed their winnings. Therefore, if the function oracleMalfunction() is called, malicious users would create wagers with a refundable token as a wager token so that if they lost the wager they can receive their wager back by calling the function `redeem()` to get refunded.

Recommendation

If the wager has already finished, then it must be redeemed by the winner. We recommend implementing a logic that the function `redeem()` will automatically decide and transfer the funds to the winner when anyone calls it.

Alleviation

[Bartrr] - A timestamp has been added to note when a token is marked as refundable. Any wager that on that token that has not already ended must be refunded. If the wager has already ended, then it must be redeemed.

[CertiK] - The `refundableTimestamp` mapping can be found on line 37 in file BartrrBase.sol. The changes can be found in the `redeem()` function starting on line 290 in `FixedWager.sol` and line 295 in `ConditionalWager.sol`. The changes can be seen at this commit 79388211666ec97e60ffcebb45a3881535e8a503.

ABC-02 | POSSIBLE DRAIN OF ALL FUNDS

Category	Severity	Location	Status
Logical Issue	Critical	projects/Bartrr/Audit_2/ConditionalWager.sol (Audit2): 268; projects/Bartrr/Audit_2/FixedWager.sol (Audit2): 265~266	Resolved

Description

Suppose userA creates a wager that has not been filled. Now assume that userA receives a refund by calling the function redeem(). Notice at the end of the call the value of wagers[_wagerId].isClosed is False because the wager is not filled which implies there is no userB to set the value of wagers[_wagerId].isClosed to True. Now with the value of wagers[_wagerId].isClosed set to False, userA can call `cancelWager()` to obtain a second refund. This exploit will allow a malicious hacker to drain the contract from all of its funds. We summarize an attack flow below.

Attack Flow #1:

1. Assume the function oracleMalfunction is called for at least one wagerToken, and this contract contains 1000 ETH.
2. Create a wager, the amount user A wagered is 1000 ETH, and set the address of user B to an attacker-controlled address.
3. Do not fill this wager.
4. Call the function redeem() to get 1000 ETH back.
5. Call the function cancelWager() to get another 1000 ETH.

Recommendation

1. Make sure no one creates a wager on refundable tokens in the function createWager(), then this function should contain the following require statement

```
require(refundableTimestamp[_wagerToken].refundable <=
refundableTimestamp[_wagerToken].nonrefundable, "Token not allowed to be wagered
on");
```

1. Make sure no one can cancel a wager after getting refunded, then the function cancelWager() should contain the following require statement

```
require(!refundUserA[_wagerId] && !refundUserB[_wagerId], "Wager has been partially
refunded");
```

Alleviation

[Bartrr] Issue acknowledged. Changes have been reflected in the latest revision. No one can create a wager on refundable tokens and refundWager() can only be called if the wager isFilled.

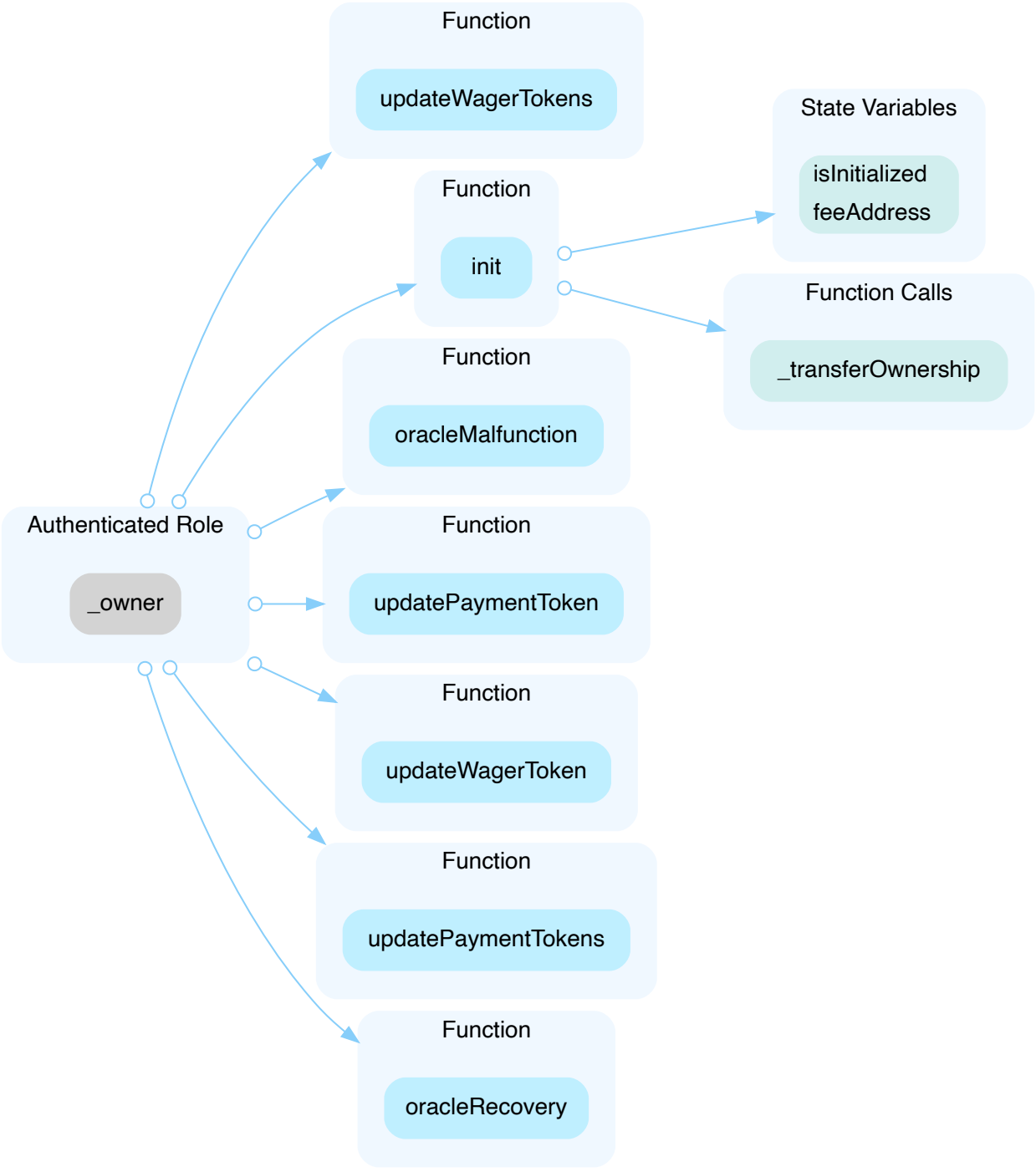
[CertiK] - The `Bartrr` team resolved the issue by following the first recommendation above and adding a require statement in `refundWager()` to ensure it can only be called if the wager is filled. The changes can be seen at this commit 79388211666ec97e60ffcebb45a3881535e8a503.

BBB-01 | CENTRALIZATION RISKS IN BARTRRBASE.SOL

Category	Severity	Location	Status
Centralization / Privilege	**Major**	**projects/Bartrr/BartrrBase.sol (Audit3): 109, 115, 120, 132, 145, 157, 172**	**Mitigated**

Description

In the contract `BartrrBase` the role `_owner` has authority over the functions shown in the diagram below. Any compromise to the `_owner` account may allow the hacker to take advantage of this authority and add hacker-controlled tokens into the `wagerTokens` and `paymentTokens` array. These tokens may contain malicious code.



Recommendation

The risk describes the current project design and potentially makes iterations to improve in the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multisignature wallets. Indicatively, here are some feasible suggestions that would also mitigate the potential risk at a different level in terms of short-term, long-term and permanent:

Short Term:

Timelock and Multi sign (⅔, ⅗) combination *mitigate* by delaying the sensitive operation and avoiding a single point of key management failure.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key compromised;
  AND
- A medium/blog link for sharing the timelock contract and multi-signers addresses information with the public audience.

Long Term:

Timelock and DAO, the combination, *mitigate* by applying decentralization and transparency.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.
  AND
- A medium/blog link for sharing the timelock contract, multi-signers addresses, and DAO information with the public audience.

Permanent:

Renouncing the ownership or removing the function can be considered *fully resolved*.

- Renounce the ownership and never claim back the privileged roles.
  OR
- Remove the risky functionality.

Alleviation

[CertiK] - The `Bartrr` team implemented a time-lock and multi-signature. The following transaction hashes prove that the contract deployer transferred ownership to a time-lock contract.

List of important addresses:

- Multisig - 0xb47f60dE0A20485f6B59EAEC7f34B663B56C1261
- TimeLock - 0x5909BB4AAa64Ed07CEa891ad41cF36890f5FE540
- FixedWager - 0x000000000AF0BB14e1BcfaA67D76cc436dF54A47
- CondtionalWager - 0x0000000003B49017d78475cf112682cb032FF9b1

This transaction https://etherscan.io/tx/0xd4f2d2c4a5b16f51bffce2e7874180460542fcf9050f618939603a81bb98b4f5 proves the deployer of `FixedWager` transferred ownership to the `TimeLock` contract.

This transaction https://etherscan.io/tx/0x0a03c3cad12f589e0c1bbb32d9aa584cb69d99a8b8581e224e7b57e70123fa9d proves the deployer of `ConditionalWager` transferred ownership to the `TimeLock` contract.

In the TimeLock contract, we can see that Multisig contract has the proposer role. This can be seen by visiting the TimeLock contract at https://etherscan.io/address/0x5909BB4AAa64Ed07CEa891ad41cF36890f5FE540 and calling the getter function has role. Once clicking the getter function, pass the role `Proposer` and the address of the multisig. The function will return `True` meaning the multsig is a proposer role.

The necessary addresses can be found at this link in the Bartrr docs: https://bartrr-1.gitbook.io/bartrr/06RAAAvycqP24yXY1DMo/safety-measures/external-security-precautions

BCP-01 | POSSIBILITY TO LOCK OTHER USERS FUNDS

Category	Severity	Location	Status
Logical Issue	Major	projects/Bartrr/ConditionalWager.sol (Base): 305~324; projects/Bartrr/FixedWager.sol (Base): 298~317	Resolved

Description

The function `_refundWager(uint256)` refunds a wager when the `oracleMalfunction()` is called for its wagerToken. However, this function will lock other users' funds.

1. Assume the function `oracleMalfunction()` is not called for the target wagerToken at step 1.
2. Create a wager by calling the `createWager()` and `fillWager()` function.
3. Assume the function `oracleMalfunction()` is called for the target wagerToken from now on.
4. UserA calls the function `redeem()` and gets refunded.
5. UserB also wants to get refunded and so calls the function `redeem()`, but the transaction fails in line 292.

Recommendation

We recommend refunding all users before setting the `isClosed` field to TRUE.

Alleviation

[Bartrr] Issue acknowledged. Changes have been reflected in the latest Gist revision. Two new mappings created inside BartrrBase.sol (refundUserA and refundUserB) to track when each user has refunded. isClosed it marked true at the very end. The nonReentrant modifier is added to the function as this does not follow the Check-Effects-Interactions pattern.

[CertiK] - The `Bartrr` team acknowledged the finding and resolved the issue. The changes can be seen at this commit, 79388211666ec97e60ffcebb45a3881535e8a503.

BCP-02 | POTENTIAL REENTRANCY ATTACK

Category	Severity	Location	Status
Control Flow	Minor	projects/Bartrr/ConditionalWager.sol (Base): 72, 260, 294, 329; projects/Bartrr/FixedWager.sol (Base): 69, 254, 288, 322	Resolved

Description

A reentrancy attack can occur when the contract creates a function that makes an external call to another untrusted contract before resolving any effects. If the attacker can control the untrusted contract, they can make a recursive call back to the original function, repeating interactions that would have otherwise not run after the external call resolved the effects.

We have not found a reentrancy attack that will drain any funds; however, for caution we recommend to add the `nonReentrant` modifier to minimize the attack surface.

Recommendation

We recommend adding the `nonReentrant` modifier for the following functions in ConditionalWager.sol and FixedWager.sol:\

1. function createWager()
2. function cancelWager()
3. function redeem()

Alleviation

[Bartrr] Issue acknowledged. Changes have been reflected in a Gist revision. The nonReentrant modifier was added to the listed functions in both ConditionalWager.sol and FixedWager.sol.

[Bartrr] Issue acknowledged. nonReentrant modifier added to createWager() in ConditionalWager.sol to complete the recommended changes.

[CertiK] - The `Bartrr` team resolved the issue by following the recommendation above. The changes can be seen at this commit, 79388211666ec97e60ffcebb45a3881535e8a503.

BCP-03 | MISSING EMIT EVENTS

Category	Severity	Location	Status
Coding Style	Minor	projects/Bartrr/BartrrBase.sol (Base): 41, 46, 56, 68, 79, 93; projects/Bartrr/ConditionalWager.sol (Base): 294; projects/Bartrr/FixedWager.sol (Base): 288	Resolved

Description

There should always be events emitted in the sensitive functions that are controlled by centralization roles.

Recommendation

It is recommended emitting events for the sensitive functions that are controlled by centralization roles.

Alleviation

[Bartrr] Issue acknowledged. Changes have been reflected in latest Gist revision.

[CertiK] The `Bartrr` team resolved the issue by following the recommendation above. Seven events were included to resolve the issues. The changes can be seen at this commit, 79388211666ec97e60ffcebb45a3881535e8a503.

BKP-01 | CANCELLED WAGERS CAN BE FILLED

Category	Severity	Location	Status
Logical Issue	Minor	projects/Bartrr/ConditionalWager.sol (Audit4): 162, 269; projects/Bartrr/FixedWager.sol (Audit4): 157, 265	Resolved

Description

The function fillWager() never checks if a wager is closed. This could lead to a user potentially filling a closed wager. Closed wagers cannot be cancelled for a refund nor redeemed hence a user will have their money stuck in a wager.

Recommendation

We recommend ensuring the value of `wager.isClosed` is `false` before proceeding with a call to `fillWager()`.

Alleviation

[CertiK] - The Bartrr team resolved the issue above by following our recommendation. The changes were implemented in the following commit: cc7d70a8c0634e7f788a7d9778051fbaeda7e864

TLA-01 | MISSING UPPER BOUND FOR LOCK DURATION

Category	Severity	Location	Status
Volatile Code	Minor	projects/Bartrr/Audit_2/TokenLockup.sol (Audit2): 41	Resolved

Description

Users may lock their tokens for a period of time they choose. To mitigate human or software error, an upper bound should be set to prevent an inordinate lock up duration.

Recommendation

We recommend including a require statement that bounds the lock up duration time.

Alleviation

[CertiK] - An upper bound of 50 year lock up duration has been included to the function `lockTokens()`. The changes can be seen on line 57 in `TokenLockup.sol` at this commit, 79388211666ec97e60ffcebb45a3881535e8a503.

TLB-01 | REENTRANCY ATTACK

Category	Severity	Location	Status
Logical Issue	Critical	projects/Bartrr/TockenLockup.sol (Base): 57~70	Resolved

Description

A reentrancy attack can occur when the contract creates a function that makes an external call to another untrusted contract before resolving any effects. If the attacker can control the untrusted contract, they can make a recursive call back to the original function, repeating interactions that would have otherwise not run after the external call resolved the effects.

Attack Flow to drain ETH:

1. Deposit a small amount of ETH by using the "lockTokens(address, uint256, uint256)" function.
2. Withdraw tokens by calling the "unlockTokens(uint256)" function.
3. When the attacker contract receives ETH, calls the "unlockTokens(uint256)" function again.
4. Repeat steps 2 and 3 to drain all ETH on this contract.

Recommendation

We recommend using the Checks-Effects-Interactions Pattern to avoid the risk of calling unknown contracts or applying OpenZeppelin ReentrancyGuard library - `nonReentrant` modifier for the aforementioned functions to prevent reentrancy attack.

Alleviation

[Bartrr] Issue acknowledged. Changes have been reflected in a Gist revision. The function now has a nonReentrant modifier and follows the Checks-Effects-Interactions Pattern.

[CertiK] - The `Bartrr` team resolved the issue by following the recommendation above.

The changes can be seen on line 92 in `TokenLockup.sol` at this commit, 79388211666ec97e60ffcebb45a3881535e8a503.

BBB-03 | MISSING ZERO ADDRESS VALIDATION

Category	Severity	Location	Status
Volatile Code	Informational	projects/Bartrr/BartrrBase.sol (Base): 48	Resolved

Description

Addresses should be checked before assignment or external call to make sure they are not zero addresses.

```
48          feeAddress = _feeAddress;
```

- `_feeAddress` is not zero-checked before being used.

Recommendation

We advise adding a zero-check for the passed-in address value to prevent unexpected errors.

Alleviation

[Bartrr] Issue acknowledged. Change reflected in a Gist revision. A require statement for _feeAddress != zero address.

[CertiK] - The `Bartrr` team resolved the issue by following the recommendation. The team included a zero address check on line 114 in `BartrrBase.sol` at this commit, 79388211666ec97e60ffcebb45a3881535e8a503

BCP-04 | DIFFERENT SOLIDITY VERSIONS

Category	Severity	Location	Status
Language Specific	Informational	projects/Bartrr/BartrrBase.sol (Base): 2; projects/Bartrr/ConditionalWager.sol (Base): 2; projects/Bartrr/FixedWager.sol (Base): 2; projects/Bartrr/RoundIdFetcher.sol (Base): 2; projects/Bartrr/TockenLockup.sol (Base): 2	Resolved

Description

Multiple Solidity versions are used in the codebase.

Versions used: `^0.8.0` , `^0.8.14` , `^0.8.1`

`^0.8.0` is used in ../node_modules/@openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol file.

```
4  pragma solidity ^0.8.0;
```

`^0.8.14` is used in projects/Bartrr/FixedWager.sol file.

```
2  pragma solidity ^0.8.14;
```

`^0.8.1` is used in ../node_modules/@openzeppelin/contracts/utils/Address.sol file.

```
4  pragma solidity ^0.8.1;
```

Recommendation

We recommend using one Solidity version.

Alleviation

[Bartrr] Issue acknowledged. I won't make any changes for the current version. All files will be compiled with the latest 0.8 solidity version.

[CertiK] The `Bartrr` team acknowledged the findings and are deciding to leave the source code unchanged.

[CertiK] - The Bartrr resolved this issue by compiling all contracts in the same version. The changes were implemented in the following commit: 79388211666ec97e60ffcebb45a3881535e8a503

OPTIMIZATIONS | BARTRR

ID	Title	Category	Severity	Status
BBB-02	Variables That Could Be Declared As `constant`	Gas Optimization	Optimization	Resolved

BBB-02 | VARIABLES THAT COULD BE DECLARED AS `constant`

Category	Severity	Location	Status
Gas Optimization	Optimization	projects/Bartrr/BartrrBase.sol (Base): 18	Resolved

Description

The linked variables could be declared as `constant` since these state variables are never modified.

Recommendation

We recommend to declare these variables as `constant` .

Alleviation

[Bartrr] Issue acknowledged. MIN_WAGER_DURATION declared as constant instead of immutable.

[CertiK] - The `Bartrr` team resolved this informational finding by declaring the linked variables with the `constant` keyword. The changes can be seen at this commit, 79388211666ec97e60ffcebb45a3881535e8a503.

APPENDIX | BARTRR

Finding Categories

Categories	Description
Centralization / Privilege	Centralization / Privilege findings refer to either feature logic or implementation of components that act against the nature of decentralization, such as explicit ownership or specialized access roles in combination with a mechanism to relocate funds.
Gas Optimization	Gas Optimization findings do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.
Logical Issue	Logical Issue findings detail a fault in the logic of the linked code, such as an incorrect notion on how block.timestamp works.
Control Flow	Control Flow findings concern the access control imposed on functions, such as owner-only functions being invoke-able by anyone under certain circumstances.
Volatile Code	Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.
Language Specific	Language Specific findings are issues that would only arise within Solidity, i.e. incorrect usage of private or delete.
Coding Style	Coding Style findings usually do not affect the generated byte-code but rather comment on how to make the codebase more legible and, as a result, easily maintainable.

Checksum Calculation Method

The "Checksum" field in the "Audit Scope" section is calculated as the SHA-256 (Secure Hash Algorithm 2 with digest size of 256 bits) digest of the content of each file hosted in the listed source repository under the specified commit.

The result is hexadecimal encoded and is the same as the output of the Linux "sha256sum" command against the target file.

DISCLAIMER | CERTIK

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to you ("Customer" or the "Company") in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes, nor may copies be delivered to any other person other than the Company, without CertiK's prior written consent in each instance.

This report is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts CertiK to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. CertiK's position is that each company and individual are responsible for their own due diligence and continuous security. CertiK's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

The assessment services provided by CertiK is subject to dependencies and under continuing development. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.

ALL SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF ARE PROVIDED "AS IS" AND "AS AVAILABLE" AND WITH ALL FAULTS AND DEFECTS WITHOUT WARRANTY OF ANY KIND. TO THE MAXIMUM EXTENT PERMITTED UNDER APPLICABLE LAW, CERTIK HEREBY DISCLAIMS ALL WARRANTIES, WHETHER EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS. WITHOUT LIMITING THE FOREGOING, CERTIK SPECIFICALLY DISCLAIMS ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT, AND ALL WARRANTIES ARISING FROM COURSE OF DEALING, USAGE, OR TRADE PRACTICE. WITHOUT LIMITING THE FOREGOING, CERTIK MAKES NO WARRANTY OF ANY KIND THAT THE SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF, WILL MEET CUSTOMER'S OR ANY OTHER PERSON'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULT, BE COMPATIBLE OR WORK WITH ANY SOFTWARE, SYSTEM, OR OTHER SERVICES, OR BE SECURE, ACCURATE, COMPLETE, FREE OF HARMFUL CODE,

OR ERROR-FREE. WITHOUT LIMITATION TO THE FOREGOING, CERTIK PROVIDES NO WARRANTY OR UNDERTAKING, AND MAKES NO REPRESENTATION OF ANY KIND THAT THE SERVICE WILL MEET CUSTOMER'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULTS, BE COMPATIBLE OR WORK WITH ANY OTHER SOFTWARE, APPLICATIONS, SYSTEMS OR SERVICES, OPERATE WITHOUT INTERRUPTION, MEET ANY PERFORMANCE OR RELIABILITY STANDARDS OR BE ERROR FREE OR THAT ANY ERRORS OR DEFECTS CAN OR WILL BE CORRECTED.

WITHOUT LIMITING THE FOREGOING, NEITHER CERTIK NOR ANY OF CERTIK'S AGENTS MAKES ANY REPRESENTATION OR WARRANTY OF ANY KIND, EXPRESS OR IMPLIED AS TO THE ACCURACY, RELIABILITY, OR CURRENCY OF ANY INFORMATION OR CONTENT PROVIDED THROUGH THE SERVICE. CERTIK WILL ASSUME NO LIABILITY OR RESPONSIBILITY FOR (I) ANY ERRORS, MISTAKES, OR INACCURACIES OF CONTENT AND MATERIALS OR FOR ANY LOSS OR DAMAGE OF ANY KIND INCURRED AS A RESULT OF THE USE OF ANY CONTENT, OR (II) ANY PERSONAL INJURY OR PROPERTY DAMAGE, OF ANY NATURE WHATSOEVER, RESULTING FROM CUSTOMER'S ACCESS TO OR USE OF THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS.

ALL THIRD-PARTY MATERIALS ARE PROVIDED "AS IS" AND ANY REPRESENTATION OR WARRANTY OF OR CONCERNING ANY THIRD-PARTY MATERIALS IS STRICTLY BETWEEN CUSTOMER AND THE THIRD-PARTY OWNER OR DISTRIBUTOR OF THE THIRD-PARTY MATERIALS.

THE SERVICES, ASSESSMENT REPORT, AND ANY OTHER MATERIALS HEREUNDER ARE SOLELY PROVIDED TO CUSTOMER AND MAY NOT BE RELIED ON BY ANY OTHER PERSON OR FOR ANY PURPOSE NOT SPECIFICALLY IDENTIFIED IN THIS AGREEMENT, NOR MAY COPIES BE DELIVERED TO, ANY OTHER PERSON WITHOUT CERTIK'S PRIOR WRITTEN CONSENT IN EACH INSTANCE.

NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS.

THE REPRESENTATIONS AND WARRANTIES OF CERTIK CONTAINED IN THIS AGREEMENT ARE SOLELY FOR THE BENEFIT OF CUSTOMER. ACCORDINGLY, NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH REPRESENTATIONS AND WARRANTIES AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH REPRESENTATIONS OR WARRANTIES OR ANY MATTER SUBJECT TO OR RESULTING IN INDEMNIFICATION UNDER THIS AGREEMENT OR OTHERWISE.

FOR AVOIDANCE OF DOUBT, THE SERVICES, INCLUDING ANY ASSOCIATED ASSESSMENT REPORTS OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

CertiK | Securing the Web3 World

Founded in 2017 by leading academics in the field of Computer Science from both Yale and Columbia University, CertiK is a leading blockchain security company that serves to verify the security and correctness of smart contracts and blockchain-based protocols. Through the utilization of our world-class technical expertise, alongside our proprietary, innovative tech, we're able to support the success of our clients with best-in-class security, all whilst realizing our overarching vision; provable trust for all throughout all facets of blockchain.