StakePet Audit Report

Prepared by Cyfrin

Version 1.1

Lead Auditors

Hans

Assisting Auditors

0kage

September 19, 2023

Contents

1 About Cyfrin

Cyfrin is a Web3 security company dedicated to bringing industry-leading protection and education to our partners and their projects. Our goal is to create a safe, reliable, and transparent environment for everyone in Web3 and DeFi. Learn more about us at cyfrin.io.

2 Disclaimer

The Cyfrin team makes every effort to find as many vulnerabilities in the code as possible in the given time but holds no responsibility for the findings in this document. A security audit by the team does not endorse the underlying business or product. The audit was time-boxed and the review of the code was solely on the security aspects of the solidity implementation of the contracts.

3 Risk Classification

	Impact: High	**Impact: Medium**	**Impact: Low**
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

4 Protocol Summary

StakePet.com is a platform where users invest in virtual pets and earn rewards. Each pet requires regular feeding, and neglecting to feed them may result in losing funds. Users can withdraw their funds at any time, but there are reduced rewards for early withdrawals. The earlier you enter and accumulate "milk" tokens, the greater your share of the total rewards.

5 Audit Scope

All files in src and script directory except mock contracts are in scope.

6 Executive Summary

Over the course of 6 days, the Cyfrin team conducted an audit on the StakePet smart contracts provided by StakePet. In this period, a total of 12 issues were found.

Summary

Project Name	StakePet
Repository	StakePet
Commit	48d46306e7ad...
Audit Timeline	Sep 1st - Sep 8th
Methods	Manual Review

Issues Found

Critical Risk	0
High Risk	2
Medium Risk	2
Low Risk	2
Informational	0
Gas Optimizations	6
Total Issues	12

7 Findings

7.1 High Risk

7.1.1 Attackers can use a malicious yield token to steal funds from users

Severity: High

Description: According to the documentation and the current implementation, anyone can create a new StakePet contract and feed any address for the `YIELD_TOKEN`. As long as a contract implements `IYieldToken` interface, the contract will be created without problems.

An attacker can create a malicious `IYieldToken` implementation and use that to steal funds from users. The StakePet contract relies on `YIELD_TOKEN.toToken()` and `YIELD_TOKEN.toValue()` in numerous places for accounting. Consider a contract that has implemented different logic in `toToken()` and `toValue()` according to the owner's hidden flag. The attacker is likely to let the malicious token contract work normally till the StakePet contract gets enough deposits. Then they can switch the hidden flag as they needed to mess the accounting and take profit from it. In the worst case, they can even manipulate the output of `IYieldToken::ERC20_TOKEN()` (maybe to freeze the user funds permanently).

Impact: User funds can be stolen or permanently locked.

Recommended Mitigation: Consider maintaining a whitelist of YIELD_TOKEN and allow creation of StakePet for only allowed yield tokens.

Client: Fixed in commit 308672e.

Cyfrin: Verified.

7.1.2 Inflation attack can cause early users to lose their deposit

Severity: High

Description: A malicious `StakePet` contract creator can steal funds from depositors by launching a typical inflation attack. To execute the attack, the creator can first deposit `1 wei` to get `1 wei` of ownership. Creator can subsequently send a big amount of collateral directly to the `StakePet` contract - this will hugely inflate the value of the single share.

Now, all subsequent pet owners who deposit their collateral will get no ownership in return. The `StakePet::ownershipToMint` function uses `StakePet::totalValue` to calculate the ownership of a new depositor. While the total ownership represented by `s_totalOwnership` remains the same `1 wei`, the `totalValueBefore` is a huge number, thanks to a large direct deposit done by the creator. This ensures that the 1 wei of share represents a huge value of collateral & causes the ownership of new depositors to round to 0.

Impact: Potential complete loss of funds for new depositors, given they receive no ownership in exchange for their deposited tokens.

Proof of Concept:

- Bob, a malicious actor, initiates the StakePet contract.
- By calling `StakePet::create`, Bob creates a pet depositing a mere `1 wei`, which grants him `1 wei` of ownership.
- Bob then directly transfers a significant amount, like 10 ether, to the `StakePet` contract.
- Consequently, a single `1 wei` share becomes equivalent to `10 ether`.
- An innocent user, Pete, tries to create a pet by calling `StakePet::create` and deposits 1 ether.
- Pete, unfortunately, receives zero ownership while his deposit remains within the contract

Recommended Mitigation: Inflation attacks have known defences. A comprehensive discussion can be found here.

One noteworthy method, as implemented by Uniswap V2, involves depositing minimal liquidity into the contract and transferring its ownership to a null address, creating "dead shares". This technique protects the subsequent depositor from potential inflation attacks.

In this case, it might be beneficial to introduce a minimum collateral requirement during contract initiation, and accordingly adjust `s_totalOwnership` to match this preset collateral.

Client: Fixed in commit a692abc and 21dd15b.

Cyfrin: Verified.

7.2 Medium Risk

7.2.1 A malicious user can grief a `StakePet` contract by creating massive number of pets

Severity: Medium

Description: The `StakePet::create` function facilitates the minting of a pet NFT by depositing collateral. However, its lack of a minimum deposit requirement for minting exposes it to potential abuse. A malicious user can exploit this by minting an excessive number of NFTs. Notably, this behaviour can strain functions like `StakePetManager::buryAllDeadPets`, which in turn calls `StakePetManager::getDeadNonBuriedPets`. This latter function iterates through all pet IDs to identify pets that are dead but not yet buried.

Impact: When a function processes an extensive and potentially unlimited list of pet IDs, there's a risk of it consuming all available gas. Consequently, it can fail, throwing an out-of-gas exception, which negatively affects users trying to interact with the contract.

Recommended Mitigation: To deter such griefing attacks, it's advisable to introduce a minimum deposit requirement for the creation of a new pet. Setting this threshold ensures that the mass-minting strategy becomes cost-prohibitive for attackers.

Client: Fixed in commit a692abc.

Cyfrin: Verified.

7.3 Low Risk

7.3.1 Closedown condition is inconsistent with the stated documentation of majority agreement

Severity: Low

Description: Documentation states the following:

"Closing the Contract: If the majority of the pets agree, they can vote to close the contract. Once closed, the remaining funds will be divided among the surviving pets. This is the most beneficial scenario for you, as you'll earn the base rewards, early withdrawal rewards, and rewards from dead pets."

Inline comments for the `StakePet::closedown` function state the following"

```
    /// @notice Close down the contract if majority wants it, after closedown everyone can withdraw
    ↪  without getting a yield cut and no pet can die.
    function closedown(uint256[] memory _idsOfMajorityThatWantsClosedown) external {
...
}
```

In both cases, condition for closedown is for `majority of pets` to agree for a closedown. However, the check used for `closedown` is that the total collateral of pets wanting a closedown should be atleast 50% of the total collateral. This would mean that a single or few pet owners with large collateral deposits can trigger a closedown even if its not something that a majority of pet owners agree to.

Having 50% of value agreement and having majority agreement could be 2 different things.

Impact: The current model can be hijacked by whales who can trigger closedown of contract whenever they wish to. This could create a bad user experience for majority of pet owners who want to stay in the contract

Recommended Mitigation: Please make documentation consistent with the vision for stake pets.

Client: Fixed in 54a4dcb

Cyfrin: Verified.

7.3.2 Exit fees implementation is inconsistent with documentation

Severity: Low

Description: Inline comments of `StakePet` contract indicate that exit fee is charged as % of the collateral.

```
The contract also has an early exit fee, which is a percentage of the collateral taken if a participant
↪  chooses to exit early.
```

However, implementation shows that exit fee is charged as a percent of yield

```
uint256 earlyExitFee = (uint256(yieldToWithdraw) * EARLY_EXIT_FEE) / BASIS_POINT
```

Recommended Mitigation: Consider correcting code documentation to reflect actual implementation

Client: Fixed in 54a4dcb

Cyfrin: Verified.

7.4 Gas Optimizations

7.4.1 Using bools for storage incurs overhead

Use uint256(1) and uint256(2) for true/false to avoid a Gwarmaccess (100 gas), and to avoid Gsset (20000 gas) when changing from 'false' to 'true', after having been 'true' in the past. See source.

```
File: StakePet.sol

91:     bool public constant TESTING = true; // TODO: Remove this when not testing

107:     bool public immutable HARDCORE; // Whether the initial collateral is taken if failing to proof
↪  of life or not
```

Client: Fixed in aea1f74

Cyfrin: Verified.

7.4.2 Cache array length outside of loop

If not cached, the solidity compiler will always read the length of the array during each iteration. That is, if it is a storage array, this is an extra sload operation (100 additional extra gas for each iteration except for the first) and if it is a memory array, this is an extra mload operation (3 additional gas for each iteration except for the first).

```
File: StakePet.sol

410:         for (uint256 i = 0; i < _idsOfMajorityThatWantsClosedown.length; i++) {
```

```
File: StakePetManager.sol

73:         for (uint256 i = 0; i < _contractIDs.length; i++) {

75:             for (uint256 j = 0; j < _petIDs[i].length; j++) {

108:         for (uint256 i = 0; i < _contractIDs.length; i++) {

110:             for (uint256 j = 0; j < _petIDs[i].length; j++) {

147:         for (uint256 i = 0; i < _contractIDs.length; i++) {
```

Client: Fixed in 627d09c

Cyfrin: Verified.

7.4.3 Don't initialize variables with default value

```
File: StakePet.sol

128:     uint256 public s_closedAtTimestamp = 0; // The timestamp that the contract was closed down

409:         uint256 _totalValueWantsClosedown = 0;

410:         for (uint256 i = 0; i < _idsOfMajorityThatWantsClosedown.length; i++) {
```

```
File: StakePetManager.sol

73:         for (uint256 i = 0; i < _contractIDs.length; i++) {

75:             for (uint256 j = 0; j < _petIDs[i].length; j++) {

108:         for (uint256 i = 0; i < _contractIDs.length; i++) {

110:             for (uint256 j = 0; j < _petIDs[i].length; j++) {

123:         uint256 j = 0;

137:         for (uint256 i = 0; i < j; i++) {

147:         for (uint256 i = 0; i < _contractIDs.length; i++) {
```

Client: Fixed in 970b71c

Cyfrin: Verified.

7.4.4 `++i` costs less gas than `i++`, especially when it's used in `for`-loops (`--i`/`i--` too)

```
File: StakePet.sol

410:         for (uint256 i = 0; i < _idsOfMajorityThatWantsClosedown.length; i++) {
```

```
File: StakePetManager.sol

73:         for (uint256 i = 0; i < _contractIDs.length; i++) {

75:             for (uint256 j = 0; j < _petIDs[i].length; j++) {

108:         for (uint256 i = 0; i < _contractIDs.length; i++) {

110:             for (uint256 j = 0; j < _petIDs[i].length; j++) {

127:         for (uint256 i = 1; i <= currentPetId; i++) {

131:                 j++;

137:         for (uint256 i = 0; i < j; i++) {

147:         for (uint256 i = 0; i < _contractIDs.length; i++) {
```

Client: Fixed in 27225c2

Cyfrin: Verified.

7.4.5 Use shift Right/Left instead of division/multiplication if possible

```
File: StakePet.sol

420:         if (_totalValueWantsClosedown <= totalValue() / 2) {
```

Client: Fixed in 540cca1

Cyfrin: Verified.

7.4.6 Use != 0 instead of > 0 for unsigned integer comparison

File: StakePet.sol

```
269:         if (_amount > 0) {

299:         if (!petAlive && pet.ownership > 0) {

432:         if (totYield > 0) {

497:             if (_milkAmount > 0) {

541:         if (yieldToWithdraw > 0) {

558:                 require(yieldToWithdraw > 0); // This should never be hit and is maybe not needed,
↪  but just in case.

562:                 require(yieldToWithdraw > 0); // This should never be hit and is maybe not needed,
↪  but just in case.

709:         if (_totalYieldNoMilk > 0) {

723:         if (s_totalOwnership > 0) {

741:         if (s_totalOwnership > 0) {
```

File: StakePetManager.sol

```
129:             if (!stakePetContract.alive(pet.lastProofOfLife) && pet.ownership > 0) {
```

Client: Fixed in 9e3d0d0

Cyfrin: Verified.