CODE SECURITY ASSESSMENT

HOVERPETS

Triwei Assessed on 01 October 2024

CODE SECURITY ASSESMENT

HOVERPETS

Triwei Assessed on

01 October 2024

TABLE OF CONTENTS

EXECUTIVE SUMMARY

Type	Tokens	**Languages**	FunC
Methods	Architecture Review, Manual Review, Unit Testing, Functional Testing, Automated Review		
Documentation	https://github.com/TheAnomalyXYZ/ton-hoverpets/blob/audit/review_01/README.md		
Repository	Detailed scope can be found at the end of the document		

REVIEWS

Review	Date	Commit
#1	13/09/2024	
#2	01/10/2024	9172a151

SECURITY FINDINGS

▲ ▲ ▲ ▲ Critical

No critical severity issues were found.

▲ ▲ ▲ High

H-01. FUNDS LOCK

Severity	High	Impact	High	Likelihood	High
Type	Financial	Commit	9172a151	Status	Fixed
Target	hoverpet_store.fc: fn op::withdraw_all				

Description: The withdraw_all function initially reserved 0.01 TON for fees, which proved insufficient for reliable execution of the withdrawal transaction. Because the send mode is set to 2, the transaction does not revert if the fee is insufficient, and results in the successful execution of the transaction without really transferring the funds to the owner's address.

Recommendation: The reserved amount for fees needs to be increase, ensuring sufficient funds are available for transaction execution.

int withdraw_amount = my_balance - 2000000; // Leave 0.02 TON for fees

▲ ▲ Medium

M-01. LACK OF DEPLOY FEES DEDUCTION IN PURCHASE CALCULATION

Severity	Medium	Impact	Medium	Likelihood	High
Type	Financial	Commit	9172a151	Status	Fixed
Target	hoverpet_store.fc: fn process_payment()				

Description: The current implementation of the process_payment() function does not account for the deploy fees when calculating the number of items purchased. This oversight can lead to an incorrect calculation of items_purchased, potentially allowing users to receive more items than they should for the amount paid.

Recommendation: Modify the process_payment() function to account for deploy fees in the calculation:

```
int deploy_fee = [calculate_deploy_fee_here];
int effective_payment = msg_value - deploy_fee;
int items_purchased = effective_payment / price;
```

Low

L-01. CREATION OF EMPTY COLLATERALIZED DEBT POSITIONS

Severity	Low	Impact	Low	Likelihood	High
Type	Financial	Commit	9172a151	Status	Fixed
Target	hoverpet_store.fc: fn process_payment()				

Description: The current implementation of the process_payment() function does not handle change when the sent value is not exactly divisible by the item price. This could result in users losing small amounts of TON on each purchase.

Impact: While the impact per transaction is small, over time and with multiple users, this could lead to accumulated TON trapped in the contract, effectively lost to the users.

Recommendation: Implement a change return mechanism in the process_payment() function:

```
int items_purchased = msg_value / price; ;;also notice the issue with the deduction of fees.
int change = msg_value % price;
if (change > 0) {
    // Implement logic to return change to the sender
}
```

DETAILED SCOPE

Last revision - Commit 9172a15146c7b9b2a999e197d6cca499cc86a095

Full path	LOCs
hoverpet_payment_tracker.fc	54
hoverpet_store.fc	143

APPROACH AND METHODOLOGY

To establish a uniform evaluation, we define the following terminology in accordance with the OWASP Risk Rating Methodology:

Likelihood

indicates the probability of a specific vulnerability being discovered and exploited in real-world scenarios

Impact

measures the technical loss and business repercussions resulting from a successful attack

Severity

reflects the comprehensive magnitude of the risk, combining both the probability of occurrence (likelihood) and the extent of potential consequences (impact)

Likelihood and impact are divided into three levels: High (H), Medium (M), and Low (L). The severity of a risk is a blend of these two factors, leading to its classification into one of four tiers: Critical, High, Medium, or Low.

When we identify an issue, our approach may include deploying contracts on our private testnet for validation through testing. Where necessary, we might also create a Proof of Concept (PoC) to demonstrate potential exploitability.

In particular, we perform the audit according to the following procedure:

Security Analysis

The process begins with a comprehensive examination of the system to gain a deep understanding of its internal mechanisms, identifying any irregularities and potential weak spots.

Semantic Consistency Checks

We then manually check the logic of implemented smart contracts and compare with the description in the white paper.

Advanced DeFi Scrutiny

We further review business logics, examine system operations, and place DeFi-related aspects under scrutiny to uncover possible pitfalls and/or bugs.

Additional Recommendations: We also provide additional suggestions regarding the coding and development of smart contracts from the perspective of proven programming practices.

To effectively classify each detected issue, we utilize the Common Weakness Enumeration (CWE-699), a community-curated catalogue of software weakness types. This helps in precisely defining and organizing software development weaknesses. While some CWE-699 categories may not directly apply to smart contracts, we adapt them in our classification process. Furthermore, for issues impacting active protocols, the public report version may temporarily exclude specific details, which will be fully disclosed once the protocol is updated with the necessary fixes.

LIMITATIONS AND USE OF REPORT

Security evaluations can't identify all vulnerabilities; a vulnerability-free assessment doesn't equate to a fully secure system. Nonetheless, code reviews are crucial for uncovering overlooked vulnerabilities and pinpointing areas needing enhanced security. Typically, applications are either entirely secure against a specific attack or wholly vulnerable. Some vulnerabilities may impact the whole application, whereas others only affect specific sections. Therefore, we conduct a thorough source code review to identify all areas requiring attention. Within the timeframe set by the client, Chain Security strives to detect as many vulnerabilities as possible.

Our analysis was confined to the code sections specified in the engagement letter. We examined if the project adhered to the given specifications, guided by the outlined threat model and trust assumptions. It's important to note that due to the inherent limitations of any software development process and product, there is always a risk of significant undetected errors or malfunctions. Additionally, uncertainties arise from any software or application used in development, as they too can harbor errors or failures. These factors can influence the system's code, functions, or operation. Our assessment did not cover the underlying third-party infrastructure, introducing additional inherent risks depending on the accurate functioning of the third-party technology stack. Readers of this report should also consider that software changes over its life cycle, as well as changes in its operating environment, can lead to operational behaviors that deviate from the original business specifications.

Check out our collection of articles exploring in-depth security options for your project!
TriWei.io/education