Security Assessment Report

Digishares SC Audit

February 18, 2025

Version 0.1

Contents

1 Confidentiality statement

This document is the exclusive property of DigiShares and Sub7 Security. This document contains proprietary and confidential information. Duplication, redistribution, or use, in whole or in part, in any form, requires consent of both DigiShares and Sub7 Security.

2 Disclaimer

This report, analysis, or any information provided by Sub7 Security is subject to the terms and conditions outlined in the agreement with our clients, including but not limited to limitations of liability, confidentiality clauses, and terms of use. The contents of this report or information provided may only be utilized in accordance with the agreed-upon scope of services and are intended solely for the recipient's internal use as stipulated in the engagement agreement.

This report is not, and should not be construed as, an endorsement or disapproval of any project, product, or team associated with the assessed technology. Sub7 Security does not provide financial, investment, or legal advice, nor does this report serve as a guarantee or certification of the security, legality, or operational integrity of the analyzed technology.

While Sub7 Security strives to identify and mitigate vulnerabilities through rigorous assessments, no technology can be deemed entirely free of risks. This report does not warrant the absolute bug-free nature or risk-free operation of the audited code or systems. Sub7 Security disclaims any responsibility for future vulnerabilities, exploits, or operational failures that may arise post-assessment.

The recipient of this report or any information provided by Sub7 Security is responsible for conducting their own due diligence and maintaining robust security practices. Cryptographic and blockchain technologies present a high level of ongoing risk due to their evolving nature. Sub7 Security advises all stakeholders to remain vigilant and adapt to emerging threats.

By utilizing our services, the recipient acknowledges that Sub7 Security's role is to reduce attack vectors and enhance the security posture of the analyzed systems to the best of our abilities within the agreed scope. Sub7 Security does not claim or assume any liability for the ultimate functionality, performance, or security of the technology reviewed.

For further inquiries or clarification, please contact Sub7 Security at hello@sub7.tech

3 About Sub7

Founded in 2022, Sub7 Security is a pioneering cybersecurity company dedicated to creating innovative and efficient solutions for a secure digital future. In 2023, we established our presence in Luxembourg after a successful pitch of our cutting-edge platform, SecHub. Our solutions earned us the prestigious recognition of being named one of the top three cybersecurity solutions in Luxembourg, further cementing our position as a leader in the field.

We are the creators of SecHub, a transformative platform designed to deliver fast, transparent, and secure cybersecurity management. SecHub empowers clients by providing direct access to top-tier security researchers, real-time findings, and rapid issue resolution. Our services include smart contract audits, penetration testing, and a range of tailored security solutions. By significantly reducing audit timelines while

maintaining robust security, SecHub is revolutionizing the way organizations manage their digital risks.

Our team brings together a wealth of expertise spanning banking, finance, cybersecurity, law, and business management. We also work closely with a dedicated lawyer, ensuring compliance with regulatory standards and providing a comprehensive approach to security and legal considerations.

Our team members have professional experience at leading organizations such as ServiceNow, Polygon, Tokeny, Bitstamp, Kreditech, Mash, Deloitte, and Bitflyer, bringing invaluable insights from diverse industries.

At Sub7 Security, we are committed to innovation, trust, and resilience. By combining cutting-edge technology, industry expertise, and legal acumen, we deliver solutions that empower businesses and build a safer, more secure digital ecosystem.

Join us as we shape the future of cybersecurity.

4 Project Overview

End to End Platform for Investment and Digital Asset Management

5 Executive Summary

Sub7 Security has been engaged to what is formally referred to as a Security Audit of Solidity Smart Contracts, a combination of automated and manual assessments in search for vulnerabilities, bugs, unintended outputs, among others inside deployed Smart Contracts.

The goal of such a Security Audit is to assess project code (with any associated specification, and documentation) and provide our clients with a report of potential security-related issues that should be addressed to improve security posture, decrease attack surface and mitigate risk.

As well general recommendations around the methodology and usability of the related project are also included during this activity

1 (One) Security Auditors/Consultants were engaged in this activity.

5.1 Scope

https://github.com/DigiShares/SwapContract

5.2 Timeline

From 10/02/2025 to 18/02/2025

5.3 Summary of Findings Identified

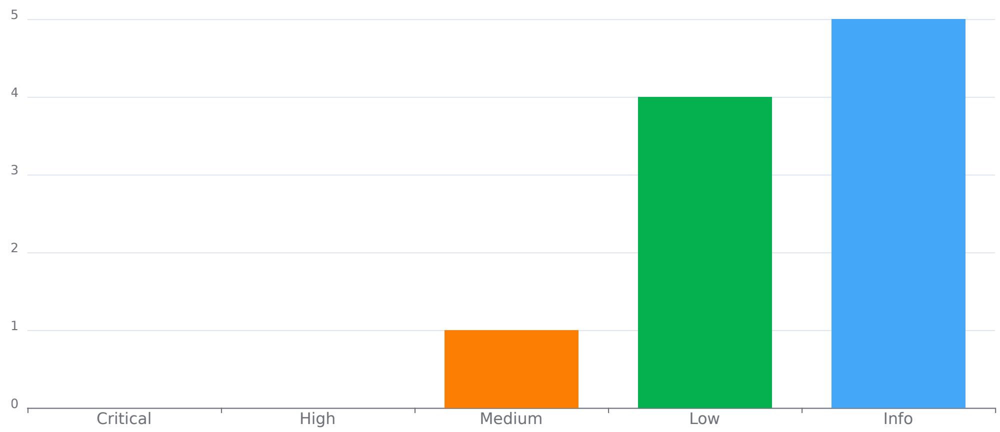

Figure 1: Executive Summary

 # 1 Medium Fee Evasion Vulnerability in SwapContract – ***Fixed***

 # 2 Low L-1 Use Ownable2Step Instead of Ownable – ***Fixed***

 # 3 Low L-2 Unsafe Downcast from uint256 to uint160 in Token Transfer Operations – ***Fixed***

 # 4 Low L-3: Missing Events in Critical Functions – ***Fixed***

 # 5 Low L-4 Event-Based Reentrancy Vulnerability in SwapContract – ***Fixed***

 # 6 Info I-1: Functions Not Used Internally Could Be Marked External – ***Fixed***

 # 7 Info I-2 Missing State Variable Visibility – ***Fixed***

 # 8 Info I-3 Name Mapping Parameters – ***Fixed***

 # 9 Info I-4 Prefix Increment Operator ++i Can Save Gas in Loops – ***Fixed***

 # 10 Info I-5 Lack of Security Contact – ***Acknowledged***

5.4 Methodology

SUB7's audit methodology involves a combination of different assessments that are performed to the provided code, including but not limited to the following:

Specification Check

Manual assessment of the assets, where they are held, who are the actors, privileges of actors, who is allowed to access what and when, trust relationships, threat model, potential attack vectors, scenarios, and mitigations. Well-specified code with standards such as NatSpec is expected to save time.

Documentation Review

Manual review of all and any documentation available, allowing our auditors to save time in inferring the architecture of the project, contract interactions, program constraints, asset flow, actors, threat model, and risk mitigation measures

Automated Assessments

The provided code is submitted via a series of carefully selected tools to automatically determine if the code produces the expected outputs, attempt to highlight possible vulnerabilities within non-running code (Static Analysis), and providing invalid, unexpected, and/or random data as inputs to a running code, looking for exceptions such as crashes, failing built-in code assertions, or potential memory leaks.

Examples of such tools are Slither, MythX, 4naly3er, Sstan, Natspec-smells, and custom bots built by partners that are actively competing in Code4rena bot races.

Manual Assessments

Manual review of the code in a line-by-line fashion is the only way today to infer and evaluate business logic and application-level constraints which is where a majority of the serious vulnerabilities are being found. This intensive assessment will check business logics, intended functionality, access control & authorization issues, oracle issues, manipulation attempts and multiple others.

Security Consultants make use of checklists such as SCSVS, Solcurity, and their custom notes to ensure every attack vector possible is covered as part of the assessment

6 Findings and Risk Analysis

6.1 Fee Evasion Vulnerability in SwapContract

Severity: Medium
Status: Fixed

Description

The fee calculation in the SwapContract performs division before multiplication, allowing users to completely avoid fees by splitting their transactions into amounts less than 1000 tokens. This vulnerability can lead to significant loss of protocol revenue.

The vulnerability exists in the fee calculation formula in line 177-178:

```
uint256 fee = (swap.closingTokenAmount / 1000) * percentageFees[i].percentage;
```

When the amount is less than 1000, the division results in 0, making the entire fee 0. This allows users to avoid fees by splitting large transactions into many small transactions.Users can completely avoid paying fees by splitting transactions. The vulnerability can be exploited multiple times by the same or different users. No special privileges are required to exploit this vulnerability.

The following test demonstrates the fee evasion vulnerability:

- Copy the POC to path: `digishares/test/audit-test`

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

import "forge-std/Test.sol";
import {console} from "forge-std/console.sol";
import "../../../src/SwapContract.sol";
import "permit2/src/interfaces/IPermit2.sol";

contract DivideBeforeMultTest is Test {
    SwapContract swapContract;
    IPermit2 permit2;

    function setUp() public {
        // We can use a dummy address for permit2 since we're only testing fee
           calculations
        swapContract = new SwapContract(
            address(1), // dummy permit2 address
            32 days // default swap expiry
        );
        permit2 = IPermit2(address(1)); // Assuming a dummy permit2 address
    }

    function testFeeEvasion() public {
        console.log("\n=== Fee Evasion Vulnerability Demonstration ===");
        console.log(
            "This test shows how the current fee calculation can be exploited"
```

```
        );

        uint8 feePercentage = 24; // 2.4% fee
        uint256 largeAmount = 1_000_000 * 1e18; // 1M tokens

        // Normal fee calculation
        uint256 normalFee = (largeAmount * feePercentage) / 1000;
        console.log("\n[Expected Behavior]");
        console.log("For a single transfer of %s tokens:", largeAmount / 1e18);
        console.log(
            "Fee should be: %s tokens (%.1f%%)",
            normalFee / 1e18,
            feePercentage / 10.0
        );

        // Exploit: Split into transactions under 1000 to avoid fees
        uint256 splitAmount = 999; // Just under 1000 threshold
        uint256 exploitFee = (splitAmount / 1000) * feePercentage; // Will be 0
        uint256 numTx = largeAmount / splitAmount;

        console.log("\n[Exploit Details]");
        console.log(
            "By splitting the transfer into %s transactions of %s tokens each:",
            numTx,
            splitAmount
        );
        console.log(
            "Fee calculation per transaction: (%s / 1000) * %s = %s",
            splitAmount,
            feePercentage,
            exploitFee
        );
        console.log("Total fees paid: %s tokens", exploitFee * numTx);

        console.log("\n[Impact Analysis]");
        console.log(
            "Normal fee that should be paid: %s tokens", normalFee / 1e18
        );
        console.log("Fees paid using exploit: %s tokens", exploitFee * numTx);
        console.log("Total fees avoided: %s tokens", normalFee / 1e18);
        console.log("This represents a 100%% fee evasion!");

        console.log("\n[Root Cause]");
        console.log(
            "The vulnerability exists because the fee calculation performs division before multiplication:"
        );
        console.log("(amount / 1000) * feePercentage");
        console.log(
            "When amount < 1000, the division results in 0, making the entire fee 0"
        );

        assertEq(exploitFee, 0, "Exploit successful: no fees paid");
    }
}
```

- Run `forge test --mc DivideBeforeMultTest -vvv`
- Output:

```
Ran 1 test for test/audit-test/DivideBeforeMultTest.t.sol:DivideBeforeMultTest
[PASS] testFeeEvasion() (gas: 17985)
Logs:
```

```

=== Fee Evasion Vulnerability Demonstration ===
  This test shows how the current fee calculation can be exploited

[Expected Behavior]
  For a single transfer of 1000000 tokens:
  Fee should be: 24000 tokens (%.1f%) 2

[Exploit Details]
  By splitting the transfer into 1001001001001001001001 transactions of 999 tokens each:
  Fee calculation per transaction: (999 / 1000) * 24 = 0
  Total fees paid: 0 tokens

[Impact Analysis]
  Normal fee that should be paid: 24000 tokens
  Fees paid using exploit: 0 tokens
  Total fees avoided: 24000 tokens
  This represents a 100% fee evasion!

[Root Cause]
  The vulnerability exists because the fee calculation performs division before multiplication:
  (amount / 1000) * feePercentage
  When amount < 1000, the division results in 0, making the entire fee 0
```

Location

a8d6a3967894b851b2deacae6c61f898814a1295/src/SwapContract.sol#L177

Recommendation

- Consider ordering multiplication before division.
- Validate that calculated fees are greater than zero.

Comments

6.2 L-1 Use Ownable2Step Instead of Ownable

Severity: Low
Status: Fixed

Description

The contract uses OpenZeppelin's `Ownable` for ownership management, which is less secure than `Ownable2Step`. `Ownable2Step` provides an additional layer of safety by requiring the new owner to explicitly accept ownership, preventing accidental transfers to invalid addresses.

If ownership is accidentally transferred to an invalid address, the contract could become permanently inaccessible, leading to loss of control over critical functions such as fee management, token whitelisting, and swap configuration.

Location

a8d6a3967894b851b2deacae6c61f898814a1295/src/SwapContract.sol#L10

Recommendation

Replace `Ownable` with `Ownable2Step` from OpenZeppelin. Update the contract as follows:

```
// ... existing code ...
import {Ownable2Step} from "openzeppelin-contracts/contracts/access/Ownable2Step.sol";
contract SwapContract is Multicall, Ownable2Step {
// ... existing code ...
}
```

Comments

6.3 L-2 Unsafe Downcast from uint256 to uint160 in Token Transfer Operations

Severity: Low
Status: Fixed

Description

The contract contains instances of unsafe downcasting from `uint256` to `uint160` when calling `permit2.transferFrom`. This can lead to truncation of bits and unintended behavior if the `_openingTokenAmount` or `fee` values exceed the maximum value of `uint160`.

If `_openingTokenAmount` or `fee` exceeds the maximum value of `uint160` (2ˆ160 - 1), the downcast will truncate the higher bits, leading to incorrect values being passed to `permit2.transferFrom`.

- **Value Truncation**: If `_openingTokenAmount` or `fee` exceeds `2^160 - 1`, the downcast will truncate the value, leading to incorrect token transfers.
- **Logical Errors**: The contract may behave unexpectedly if the truncated values are used in calculations or comparisons.
- **Funds at Risk**: Incorrect token transfers could result in loss of funds or failed transactions.

Location

a8d6a3967894b851b2deacae6c61f898814a1295/src/SwapContract.sol#L140

a8d6a3967894b851b2deacae6c61f898814a1295/src/SwapContract.sol#L183

Recommendation

Use OpenZeppelin's `SafeCast` library to safely downcast `uint256` to `uint160`. The `SafeCast` library ensures that the downcast operation reverts if the value exceeds the maximum value of the target type, preventing truncation and logical errors.

```
import {SafeCast} from "openzeppelin-contracts/contracts/utils/math/SafeCast.sol";

// ... existing code ...

// Location 1: Safe downcast for openingTokenAmount
permit2.transferFrom(
    msg.sender,
    address(this),
    SafeCast.toUint160(openingTokenAmount),
    openingToken
);

// Location 2: Safe downcast for fee
permit2.transferFrom(
    msg.sender,
    percentageFees[i].recipient,
    SafeCast.toUint160(fee),
    swap.closingToken
);
```

Comments

6.4 L-3: Missing Events in Critical Functions

Severity: Low
Status: Fixed

Description

The `SwapContract` is missing events in several critical functions. Events are essential for off-chain tracking of contract state changes, and their absence makes it difficult or impossible to monitor and audit transactions related to these functions.

- **Lack of Transparency**: Without events, off-chain systems (e.g., frontends, monitoring tools, or auditors) cannot track state changes.
- **Auditability Issues**: Missing events make it difficult to audit historical changes, reducing the contract's transparency and trustworthiness.
- **User Experience Degradation**: Users and stakeholders cannot be notified of critical changes, leading to potential confusion or mistrust.

Location

a8d6a3967894b851b2deacae6c61f898814a1295/src/SwapContract.sol#L269

a8d6a3967894b851b2deacae6c61f898814a1295/src/SwapContract.sol#L284

a8d6a3967894b851b2deacae6c61f898814a1295/src/SwapContract.sol#L300

a8d6a3967894b851b2deacae6c61f898814a1295/src/SwapContract.sol#L316

a8d6a3967894b851b2deacae6c61f898814a1295/src/SwapContract.sol#L333

a8d6a3967894b851b2deacae6c61f898814a1295/src/SwapContract.sol#L349

a8d6a3967894b851b2deacae6c61f898814a1295/src/SwapContract.sol#L363

a8d6a3967894b851b2deacae6c61f898814a1295/src/SwapContract.sol#L373

Recommendation

Consider emitting events for all state-changing functions, particularly for administrative operations. Each event should include relevant parameters that were used to make the change.

Comments

6.5 L-4 Event-Based Reentrancy Vulnerability in SwapContract

Severity: Low
Status: Fixed

Description

The SwapContract is vulnerable to event-based reentrancy attacks in both its `close()` and `expire()` functions. This vulnerability occurs because both functions emit events after making external calls, which can be exploited by a malicious contract to reenter the functions before the state is fully updated.

The vulnerability exists in both functions where the contract:

1. Makes external calls to transfer tokens
2. Emits events after the external calls

In `close()`:

```
// External calls
permit2.transferFrom(...);
IERC20(swap.openingToken).safeTransfer(...);
// Event emitted after external calls
emit Closed(...);
```

In `expire()`:

```
// External call
IERC20(swap.openingToken).safeTransfer(...);
// Event emitted after external call
emit Expired(...);
```

The primary impact of this vulnerability is on event tracking and monitoring:

1. Missing Event Calls: Reentrancy can cause events to be skipped or not emitted, leading to incomplete transaction histories.
2. Monitoring System Failures: Off-chain systems that rely on events for tracking may miss critical state changes.
3. Audit Trail Gaps: Missing events create gaps in the audit trail, making it difficult to reconstruct transaction histories. Frontend Inconsistencies: User interfaces that rely on events may display incorrect or incomplete information.
4. Analytics Discrepancies: Data analysis based on events may produce inaccurate results. Compliance Issues: Missing events could lead to regulatory compliance problems for certain applications. While this vulnerability doesn't directly lead to fund loss, it significantly impacts the reliability and transparency of the contract's operation.

Location

a8d6a3967894b851b2deacae6c61f898814a1295/src/SwapContract.sol#L161

a8d6a3967894b851b2deacae6c61f898814a1295/src/SwapContract.sol#L222

Recommendation

It is recommended to add re-entrancy guard from Openzeppelin to the functions making external calls. The functions should use a Checks-Effects-Interactions pattern. The external calls should be executed at the end of the function and all the state-changing and event emits must happen before the call.

Comments

6.6 I-1: Functions Not Used Internally Could Be Marked External

Severity: Info
Status: Fixed

Description

The `singlePermit` function in `SwapContract.sol` is declared as `public`, but it is not used internally within the contract. Functions that are only called externally should be marked as `external` to optimize gas usage and improve code clarity.

- **Gas Inefficiency**: Using `public` instead of `external` for functions that are only called externally results in slightly higher gas costs due to unnecessary memory operations.
- **Code Clarity**: Marking functions as `external` when they are only called externally improves code readability and makes the intended usage clearer.

Location

a8d6a3967894b851b2deacae6c61f898814a1295/src/SwapContract.sol#L269

Recommendation

Update the `singlePermit` function to use the `external` visibility modifier.

1. **Gas Optimization**: `external` functions are more gas-efficient than `public` functions because they avoid copying arguments to memory.
2. **Code Clarity**: Marking the function as `external` clearly indicates that it is only intended to be called from outside the contract, improving readability and maintainability.

Comments

6.7 I-2 Missing State Variable Visibility

Severity: Info
Status: Fixed

Description

The issue is found in the following state variable declarations in `SwapContract.sol`:

1. **Line 34**: `IPermit2 immutable permit2;`
2. **Line 35**: `PercentageFee[] percentageFees;`
3. **Line 36**: `uint32 swapExpiry;`
4. **Line 38**: `mapping(address => mapping(string => Swap))swaps;`
5. **Line 39**: `mapping(address => bool)openingTokens;`
6. **Line 40**: ‘mapping(address => bool) closingTokens;

- **Security Risk**: Missing visibility modifiers default to `internal`, which may not be the intended access level.
- **Code Clarity**: Lack of explicit visibility makes it harder to understand the intended access control.
- **Maintainability**: Future developers may misinterpret the intended access level of these variables.

Location

a8d6a3967894b851b2deacae6c61f898814a1295/src/SwapContract.sol#L34

a8d6a3967894b851b2deacae6c61f898814a1295/src/SwapContract.sol#L35

a8d6a3967894b851b2deacae6c61f898814a1295/src/SwapContract.sol#L36

a8d6a3967894b851b2deacae6c61f898814a1295/src/SwapContract.sol#L38

a8d6a3967894b851b2deacae6c61f898814a1295/src/SwapContract.sol#L39

a8d6a3967894b851b2deacae6c61f898814a1295/src/SwapContract.sol#L40

Recommendation

Explicitly define visibility for all state variables. These variables can be specified as public, internal or private based on their intended usage. Public variables should be used for data that needs to be accessible externally, internal for data that should only be accessible within the contract and derived contracts, and private for data that should only be accessible within the defining contract.

Adding explicit visibility modifiers improves code clarity, security, and maintainability by making the intended access control explicit. It helps prevent unintended access to sensitive data and makes the contract's design intentions clearer to developers and auditors.

Comments

6.8 I-3 Name Mapping Parameters

Severity: Info
Status: Fixed

Description

The `SwapContract` is missing events in several critical functions. Events are essential for off-chain tracking of contract state changes, and their absence makes it difficult or impossible to monitor and audit transactions related to these functions.

- **Lack of Transparency**: Without events, off-chain systems (e.g., frontends, monitoring tools, or auditors) cannot track state changes.
- **Auditability Issues**: Missing events make it difficult to audit historical changes, reducing the contract's transparency and trustworthiness.
- **User Experience Degradation**: Users and stakeholders cannot be notified of critical changes, leading to potential confusion or mistrust.

Location

a8d6a3967894b851b2deacae6c61f898814a1295/src/SwapContract.sol#L38

a8d6a3967894b851b2deacae6c61f898814a1295/src/SwapContract.sol#L39

a8d6a3967894b851b2deacae6c61f898814a1295/src/SwapContract.sol#L40

Recommendation

It is recommended to name the mapping parameters if Solidity 0.8.18 and above is used. This improves code clarity and maintainability by making the purpose of each mapping key and value explicit.

Comments

6.9 I-4 Prefix Increment Operator ++i Can Save Gas in Loops

Severity: Info
Status: Fixed

Description

The contract uses the post-increment operator `i++` in loops, which is less gas efficient than the prefix increment operator `++i`. The prefix increment operator skips storing the value before the incremental operation, saving gas when the return value of the expression is ignored.

- **Gas Inefficiency**: Using `i++` instead of `++i` results in slightly higher gas costs due to unnecessary storage operations.
- **Cumulative Cost**: In loops with many iterations, the additional gas costs can become significant.

Location

a8d6a3967894b851b2deacae6c61f898814a1295/src/SwapContract.sol#L176

Recommendation

Replace the post-increment operator `i++` with the prefix increment operator `++i` in all loops where the return value is not used.

Comments

6.10 I-5 Lack of Security Contact

Severity: Info
Status: Acknowledged

Description

The contract does not include a security contact, which is a recommended best practice for smart contract development. Providing a specific security contact (such as an email or ENS name) simplifies the process for individuals to report vulnerabilities and ensures proper communication channels are established for security disclosures

Location

a8d6a3967894b851b2deacae6c61f898814a1295/src/SwapContract.sol#L10

Recommendation

Add a NatSpec comment containing a security contact above the contract definition using the `@custom:security-contact` convention. This should include a dedicated email address or other secure contact method for reporting security issues.

Comments

sub7

FOLLOW US