verichains

SECURITY AUDIT OF

UPAD AMM SMART CONTRACT

Public Report

Mar 25, 2024

Verichains Lab

info@verichains.io

https://www.verichains.io

Driving Technology > Forward

ABBREVIATIONS

Name	Description
Ethereum	An open source platform based on blockchain technology to create and distribute smart contracts and decentralized applications.
Ether (ETH)	A cryptocurrency whose blockchain is generated by the Ethereum platform. Ether is used for payment of transactions and computing services in the Ethereum network.
Smart contract	A computer protocol intended to digitally facilitate, verify or enforce the negotiation or performance of a contract.
Solidity	A contract-oriented, high-level language for implementing smart contracts for the Ethereum platform.
Solc	A compiler for Solidity.
ERC20	ERC20 (BEP20 in Binance Smart Chain or *x*RP20 in other chains) tokens are blockchain-based assets that have value and can be sent and received. The primary difference with the primary coin is that instead of running on their own blockchain, ERC20 tokens are issued on a network that supports smart contracts such as Ethereum or Binance Smart Chain.

EXECUTIVE SUMMARY

This Security Audit Report was prepared by Verichains Lab on Mar 25, 2024. We would like to thank the uPad for trusting Verichains Lab in auditing smart contracts. Delivering high-quality audits is always our top priority.

This audit focused on identifying security flaws in code and the design of the uPad AMM Smart Contract. The scope of the audit is limited to the source code files provided to Verichains. Verichains Lab completed the assessment using manual, static, and dynamic analysis techniques.

During the audit process, the audit team had identified a vulnerable issue in the smart contracts code.

TABLE OF CONTENTS

1. MANAGEMENT SUMMARY

1.1. About uPad AMM Smart Contract

uPad AMM Smart Contract stands as an Automated Market Maker (AMM) decentralized exchange built on the Scroll blockchain. This innovative protocol enables users to trade and generate earnings by providing liquidity for various token pairs.

1.2. Audit scope

This audit focused on identifying security flaws in code and the design of the uPad AMM Smart Contract that was deployed on Scroll blockchain.

The latest version was made available in the course of the review at the block number 4253542:

Contract	Address
UPadFactory	0xEc7f3E8183FD174aD02c32ecE0b2e3669cA39d63
UPadRouter	0xE19Cc1B1820676723dAf50C0f46F5FAe66227199
Multicall	0x411f9d731695e4aA068f2D07d4C2dD5cA79b2306
UPadPair	0x76ab7Fe5A2DaCE36c3e35008099cbF7e81D6F044 (represent pair)
Registry	0xF206184386E0D8e7dd6cb82dA04Bf5c946b23012

Table 1. Audit Scope

1.3. Audit methodology

Our security audit process for smart contract includes two steps:

- Smart contract codes are scanned/tested for commonly known and more specific vulnerabilities using public and RK87, our in-house smart contract security analysis tool.
- Manual audit of the codes for security issues. The contracts are manually analyzed to look for any potential problems.

Following is the list of commonly known vulnerabilities that were considered during the audit of the smart contract:

- Integer Overflow and Underflow
- Timestamp Dependence

- Race Conditions
- Transaction-Ordering Dependence
- DoS with (Unexpected) revert
- DoS with Block Gas Limit
- Gas Usage, Gas Limit and Loops
- Redundant fallback function
- Unsafe type Inference
- Reentrancy
- Explicit visibility of functions state variables (external, internal, private and public)
- Logic Flaws

For vulnerabilities, we categorize the findings into categories as listed in table below, depending on their severity level:

SEVERITY LEVEL	DESCRIPTION
CRITICAL	A vulnerability that can disrupt the contract functioning; creates a critical risk to the contract; required to be fixed immediately.
HIGH	A vulnerability that could affect the desired outcome of executing the contract with high impact; needs to be fixed with high priority.
MEDIUM	A vulnerability that could affect the desired outcome of executing the contract with medium impact in a specific scenario; needs to be fixed.
LOW	An issue that does not have a significant impact, can be considered as less important.

Table 2. Severity levels

1.4. Disclaimer

uPad acknowledges that the security services provided by Verichains, are conducted to the best of their professional abilities but cannot guarantee 100% coverage of all security vulnerabilities. uPad understands and accepts that despite rigorous auditing, certain vulnerabilities may remain undetected. Therefore, uPad agrees that Verichains shall not be held responsible or liable, and shall not be charged for any hacking incidents that occur due to security vulnerabilities not identified during the audit process.

1.5. Acceptance Minute

This final report served by Verichains to the uPad will be considered an Acceptance Minute. Within 7 days, if no any further responses or reports is received from the uPad, the final report will be considered fully accepted by the uPad without the signature.

2. AUDIT RESULT

2.1. Overview

The uPad AMM Smart Contract was written in the `Solidity` language. The contract bases on `Router`, `Factory` and `Pair` of `Uniswap V2`.

2.1.1. Router

This smart contract, called UPadRouter02, facilitates decentralized trading on the Scroll blockchain. It allows users to add and remove liquidity, as well as swap tokens efficiently. The contract supports various functionalities such as adding liquidity with Ethereum (ETH), removing liquidity in ETH or tokens, and swapping tokens with ETH or other tokens.

Key Features:

- Users can add liquidity to token pairs, enabling trading with minimal slippage.
- Liquidity can be removed from token pairs as needed.
- The contract supports swapping tokens with other tokens or ETH.
- It also supports fee-on-transfer tokens, ensuring compatibility with tokens that charge fees during transfers.

Overall, UPadRouter02 provides essential functions for decentralized trading and liquidity management on the Scroll blockchain.

2.1.2. Factory

This smart contract, `UPadFactory`, serves as a factory for creating pairs of tokens on the UPad decentralized exchange. It allows users to create token pairs, set fees, and manage the factory.

Key Features:

- Users can create token pairs for trading on the UPad decentralized exchange.
- The contract ensures that token pairs are unique and not duplicated.
- It allows for setting a fee recipient address for collecting trading fees.
- Users can also designate a fee setter address for managing the fee recipient.

Overall, UPadFactory provides essential functionalities for managing token pairs and fees on the UPad decentralized exchange.

2.1.3. Pair

This smart contract, `UPadPair`, represents a pair of tokens on the UPad decentralized exchange. It facilitates token swapping, liquidity provision, fee calculation, and synchronization of reserves.

Key Features:

- Supports token swapping between two tokens in the pair.
- Facilitates liquidity provision by minting and burning LP (liquidity provider) tokens.
- Calculates and collects fees for liquidity provision.
- Ensures synchronization of on-chain reserves with token balances.
- Implements safety measures to prevent reentrancy attacks and ensure proper function execution.

Overall, UPadPair provides essential functionalities for managing token pairs and enabling decentralized trading on the UPad exchange.

2.1.4. Registry

The `ConverterRegistryContract` is a smart contract designed to manage a registry of converters associated with tokens in a decentralized network. Its main functionalities include:

- Registration and Removal of Converters: Allows the addition and removal of converter addresses for a given token.
- Ownership Management: Implements ownership functionality, ensuring that only the owner can register or remove converters.
- Event Emission: Emits events to notify when a converter is added or removed from the registry.
- Querying Information: Provides functions to retrieve information about tokens and converters, such as the number of tokens in the registry, the number of converters associated with a token, and the converter address associated with a token.

Overall, the ConverterRegistryContract facilitates the organization and management of converter addresses in a decentralized network, ensuring transparency and accessibility.

2.1.5. Multicall

The `Multicall` contract is a utility smart contract that enables batched function calls on the Scroll blockchain. It allows users to execute multiple functions in a single transaction, reducing gas costs and improving efficiency. **Should not approve this contract to spend your tokens.**

2.2. Findings

During the audit process, the audit team found a vulnerability in the given version of uPad AMM Smart Contract.

2.2.1. [HIGH] - Incorrect handling of the `unregisterConverter()` function

Positions:

- `ConverterRegistryContract.sol#unregisterConverter()`

Description:

The function removes a converter from the `tokensToConverters[token][]` array by shifting the elements to left. Unfortunately, the function does not remove the last element from the array, which can lead to a situation where the same converter can be added multiple times to the array. This can result in unexpected behavior and potential security issues.

```
function unregisterConverter(address _token, uint32 _index)
        public
        ownerOnly
        validAddress(_token)
    {
        require(_index < tokensToConverters[_token].length);

        address converter = tokensToConverters[_token][_index];

        for (uint32 i = _index + 1; i < tokensToConverters[_token].length; i++) {
            tokensToConverters[_token][i - 1] = tokensToConverters[_token][i];
        }
        //@audit: the last element should be removed here

        tokensToConverters[_token].length--;

        delete convertersToTokens[converter];

        emit ConverterRemoval(_token, converter);
    }
```

RECOMMENDATION

Code logic code may be fixed as below:

```
function unregisterConverter(address _token, uint32 _index)
    public
    ownerOnly
    validAddress(_token)
{
    require(_index < tokensToConverters[_token].length, "Invalid index");
```

```
    address converter = tokensToConverters[_token][_index];
    address lastConverter = tokensToConverters[_token][tokensToConverters[_token].length - 1];

    // Move the last converter to the position of the converter to remove
    tokensToConverters[_token][_index] = lastConverter;

    // Update the mapping for the last converter to point to its new position
    convertersToTokens[lastConverter] = _token;

    // Remove the last element from the array
    tokensToConverters[_token].pop();

    // Remove the converter from the converters -> tokens mapping
    delete convertersToTokens[converter];

    // Dispatch the converter removal event
    emit ConverterRemoval(_token, converter);
}
```

UPDATES

- ***Mar 25, 2024***: The issue has been mitigated but not completely fixed.

uPad AMM Smart Contract has updated:

```
function unregisterConverter(address _token, uint32 _index)
    public
    ownerOnly
    validAddress(_token)
{
    uint256 sizeTokenArray = tokensToConverters[_token].length;
    require(_index < sizeTokenArray);
    address converter = tokensToConverters[_token][_index];
    for (uint32 i = _index + 1; i < sizeTokenArray; i++) {
        tokensToConverters[_token][i - 1] = tokensToConverters[_token][i];
    }
    delete tokensToConverters[_token][sizeTokenArray - 1];  //@audit: the last element should be popped instead of deleted (in Solidity, delete is used to clear value, not to remove elements from an array)
    delete convertersToTokens[converter];

    emit ConverterRemoval(_token, converter);
}
```

3. VERSION HISTORY

Version	Date	Status/Change	Created by
1.0	*Mar 19, 2024*	Public Report	Verichains Lab
1.1	*Mar 25, 2024*	Public Report	Verichains Lab

Table 3. Report versions history