SECURITY AUDIT OF

AVT PLATFORM

Public Report

Jun 20, 2023

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.
Optimism	The Optimism protocol is a Layer2 protocol that is intended to help Ethereum users speed up and pay less fees for transactions made on the Ethereum network.

EXECUTIVE SUMMARY

This Security Audit Report was prepared by Verichains Lab on Jun 20, 2023. We would like to thank the AVT Team 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 AVT Platform. 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 found some vulnerabilities in the application.

Table of Contents

1. MANAGEMENT SUMMARY

1.1. About AVT Platform

AVT is a decentralised, transparent and non custodial perpetual platform built on Optimism where users can swap, long or short BTC, ETH and OP with up to 50x leverage.

1.2. Audit scope

The AVT Platform smart contracts was based on a version of GMX smart contracts on commit `aa34bafd9a52f14af4b82484ec0286f4d5564fb9` (*https://github.com/gmx-io/gmx-contracts/tree/aa34bafd9a52f14af4b82484ec0286f4d5564fb9*).

This audit focused on identifying security flaws in the design and implementation of the modified AVT smart contracts compared to the original GMX smart contracts.

It was conducted on commit `17ebf5ea371d62d12b6d567b40a37e29b894683f` on AVT contracts repo (*https://github.com/vhuarui/mw-avt-contracts/tree/17ebf5ea371d62d12b6d567b40a37e29b894683f*).

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 1. Severity levels

1.4. Disclaimer

Please note that security auditing cannot uncover all existing vulnerabilities, and even an audit in which no vulnerabilities are found is not a guarantee for a 100% secure smart contract. However, auditing allows discovering vulnerabilities that were unobserved, overlooked during development and areas where additional security measures are necessary.

2. AUDIT RESULT

2.1. Overview

The AVT Platform smart contracts (as well as original GMX smart contracts) was written in `Solidity` language. The code base also imported several contracts from OpenZeppelin's public library.

2.2. Findings

During the audit process, the audit team found some vulnerabilities in the given version of AVT Platform.

AVT Team fixed some issues, according to Verichains's draft report in commit `645ceb940fa6acd10e8b1c7514ae4ab1979049f0`.

#	Issue	Severity	Status
1	Incorrect ALP referral reward increasing	MEDIUM	Fixed
2	Outdated version of base GMX smart contracts	HIGH	Acknowledged
3	Incorrect check during trader referral code update in `ReferralStorage`	LOW	Acknowledged

2.2.1. Incorrect ALP referral reward increasing MEDIUM

Affected files:

- *`contracts/staking/RewardRouterV2.sol`*

Current implementation of `handleRewards` function:

```
function handleRewards(

    uint256 avtAmount = 0;
    if (_shouldClaimStakedAvtReward) {
        avtAmount = IRewardTracker(stakedAvtTracker).claimForAccount(account, account);
    }
    if (_shouldClaimStakedAlpReward) {
        uint256 avtAmount1 = IRewardTracker(stakedAlpTracker).claimForAccount(account, account);
        uint256 avtAmount2 = IRewardTracker(stakedAlpTracker2).claimForAccount(
            account,
            account
        );
        avtAmount = avtAmount.add(avtAmount1).add(avtAmount2);

        IAlpReferralReward(alpReferralReward).increaseAlpReferral(account, avtAmount);
    }

}
```

In this part of function, we have two rewards to claim: one for AVT staked, and one for ALP staked. The ALP staked rewards includes rewards from tracker1 and tracker2. Only the rewards from ALP staked will be used to calculated the increasing ALP referral reward.

But the current implementation code includes both AVT staked and ALP staked in reward calculation, as in the following statements:

```
avtAmount = avtAmount.add(avtAmount1).add(avtAmount2);

IAlpReferralReward(alpReferralReward).increaseAlpReferral(account, avtAmount);
```

RECOMMENDATION

Update the code to include only ALP staked rewards in the calculation, as follows:

```
if (_shouldClaimStakedAlpReward) {

    IAlpReferralReward(alpReferralReward).increaseAlpReferral(account, avtAmount1.add(avtAmount2));
}
```

UPDATES

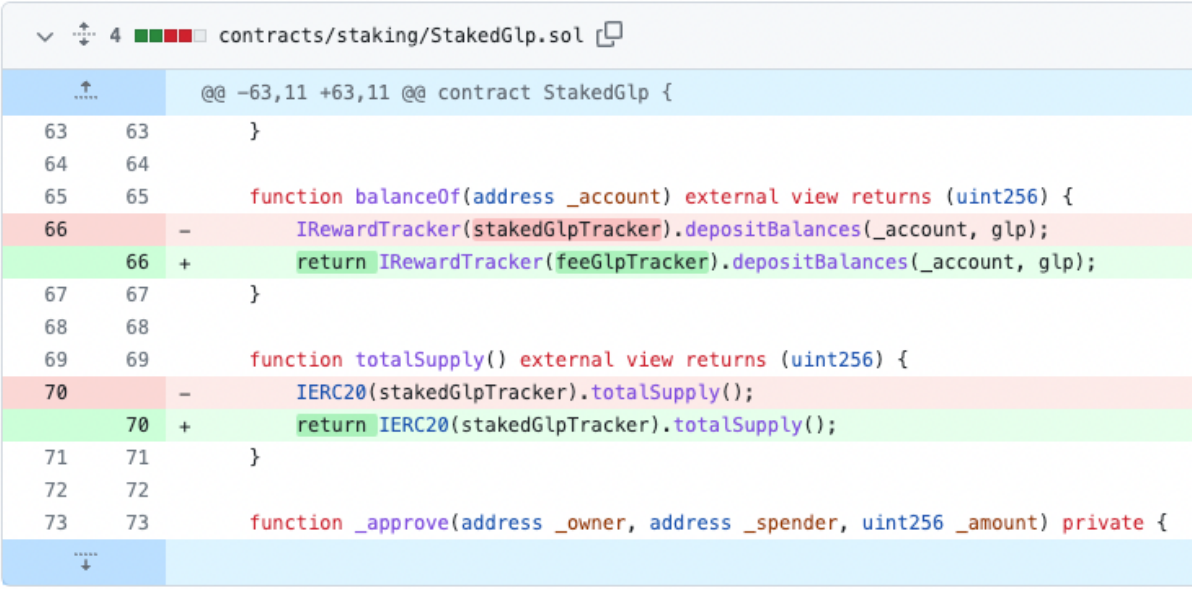
- *Jun 14, 2023*: This issue has been acknowledged and fixed by the AVT Team in commit `645ceb940fa6acd10e8b1c7514ae4ab1979049f0`.

2.2.2. Outdated version of base GMX smart contracts HIGH

Compared to the original version on which the AVT contracts were based, the latest version of the GMX contracts includes numerous updates, including bug fixes and improvements.

- Latest version of GMX contracts (as of writing time): *https://github.com/gmx-io/gmx-contracts/tree/649a1b328725c4523530ff8fc7ca5fc2bd4167a0*
- Changes between latest version and original version: *https://github.com/gmx-io/gmx-contracts/compare/aa34bafd9a52f14af4b82484ec0286f4d5564fb9..649a1b328725c4523530ff8fc7ca5fc2bd4167a0*

For example, this is one of the bug fixes in the updated contracts:



```
4  contracts/staking/StakedGlp.sol
@@ -63,11 +63,11 @@ contract StakedGlp {
      }

      function balanceOf(address _account) external view returns (uint256) {
-         IRewardTracker(stakedGlpTracker).depositBalances(_account, glp);
+         return IRewardTracker(feeGlpTracker).depositBalances(_account, glp);
      }

      function totalSupply() external view returns (uint256) {
-         IERC20(stakedGlpTracker).totalSupply();
+         return IERC20(stakedGlpTracker).totalSupply();
      }

      function _approve(address _owner, address _spender, uint256 _amount) private {
```

Image 1. A bug fix from updated GMX contracts

We recommend updating the AVT contracts to utilize the latest version of the GMX contracts.

UPDATES

- *Jun 14, 2023*: This issue has been acknowledged by the AVT Team.

2.2.3. Incorrect check during trader referral code update in ReferralStorage LOW

Affected files:

- *contracts/referrals/ReferralStorage.sol*

Current implementation of `setTraderReferralCode` function is as follow:

```
function setTraderReferralCode(address _account, bytes32 _code) external override onlyHandler {
    require(traderReferralCodes[msg.sender] == bytes32(0), "ReferralStorage: already set");
    _setTraderReferralCode(_account, _code);
}
```

Because we are updating the referral code for `_account`, not for `msg.sender`, so instead of `require(traderReferralCodes[msg.sender] == bytes32(0), ...);` the checking statement should be `require(traderReferralCodes[_account] == bytes32(0), ...);`

By the way, these require statements should be placed in `_setTraderReferralCode` function.

RECOMMENDATION

Move the require statements to the internal function `_setTraderReferralCode`

```
function _setTraderReferralCode(address _account, bytes32 _code) private {
    require(traderReferralCodes[_account] == bytes32(0), "ReferralStorage: already
set");
    traderReferralCodes[_account] = _code;
    emit SetTraderReferralCode(_account, _code);
}
```

UPDATES

- *Jun 14, 2023*: This issue has been acknowledged by the AVT Team.

2.3. Additional notes and recommendations

2.3.1. Use modifier instead of private view to validate handler

Contract
`AlpReferralReward` **(contracts/staking/AlpReferralReward.sol)**

To enhance handler permission validation, it is advisable to replace the usage of a private view (`_validateHandler`) with a modifier. This approach aligns with a common pattern employed in multiple instances within the AVT smart contract source codes.

```
modifier onlyHandler() {
    require(isHandler[msg.sender], "AlpReferralReward: forbidden");
    _;
}
```

UPDATES

- *Jun 14, 2023*: This issue has been acknowledged by the AVT Team.

3. VERSION HISTORY

Version	Date	Status/Change	Created by
1.0	*Jun 20, 2023*	Public Report	Verichains Lab

Table 2. Report versions history