Pichi Finance Marketplace

Smart Contract Security Assessment

June 28, 2024

ABSTRACT

Dedaub was commissioned to perform a security audit of Pichi Finance's Marketplace contracts that offers a marketplace for points via NFTs, ensuring secure transactions with no collateral required.

BACKGROUND

The Pichi Finance protocol allows users to create ERC-6551 (tokenbound) accounts to earn points. Each account is owned by an NFT and can be traded through the Pichi marketplace by trading the NFT.

The protocol consists of three main contracts:

1. The `PichiWalletNFT` contract allows the minting of NFTs at a price set by the owner of the contract.
2. The `PichiHelper` contract uses the PichiWalletNFT contract to simultaneously mint NFTs and create the tokenbound accounts controlled by these NFTs. It also provides functionality for depositing whitelisted tokens into a wallet controlled by the owner of an NFT, potentially against a deposit fee.
3. The `PichiMarketplace` contract allows the trading of NFTs from whitelisted collections for various whitelisted currencies. A buyer can either execute a listing which was signed off-chain by a seller, or alternatively a seller can accept an offer which was signed off-chain by a buyer, using the EIP-712 signature scheme. A user can cancel their orders individually or in bulk by updating a nonce. The contract also has functionality to validate off-chain the above off-chain orders, as well as to handle the transfer of NFTs and settle payments.

SETTING & CAVEATS

This audit report mainly covers the contracts of the michi-contracts repository of the Pichi Finance protocol at commit `d9cee365b59ed7798a8f958cd8b23574c1e08b9a`.

Two auditors worked on the codebase for 2 days on the following contracts:

```
src
├── PichiHelper.sol
├── PichiWalletNFT.sol
├── interfaces
│   ├── IPichiMarketplace.sol
│   └── IPichiWalletNFT.sol
├── libraries
│   ├── OrderTypes.sol
│   └── SignatureAuthentication.sol
└── marketplace
    └── PichiMarketplace.sol
```

Prior to the audit the Pichi team communicated to the audit team that there is a known "issue" in the PichiHelper contract allowing users to bypass the deposit fee. The Pichi team has indicated that for the moment the deposit fee is set to 0. If in the future it is increased, Pichi will reward users with additional benefits in case they choose to pay it.

The audit's main target is security threats, i.e., what the community understanding would likely call "hacking", rather than the regular use of the protocol. Functional correctness (i.e. issues in "regular use") is a secondary consideration. Typically it can only be covered if we are provided with unambiguous (i.e. full-detail) specifications of what is the expected, correct behavior. In terms of functional correctness, we often trusted the code's calculations and interactions, in the absence of any other specification. Functional correctness relative to low-level calculations (including units, scaling and quantities returned from external protocols) is generally most effectively done through thorough testing rather than human auditing.

PROTOCOL-LEVEL CONSIDERATIONS

ID	Description	STATUS
P1	TBA trustless sales can be vulnerable to order misrepresentation	**INFO**

Token bound accounts are designed to hold different kinds of tokens and thus In order to enable trustless sales of TBAs, decentralized marketplaces such as PichiMarketplace will need to implement safeguards against fraudulent behavior by malicious account owners. Consider the following potential vulnerability taken from the Security Considerations section of ERC6551:

1. Alice owns an ERC-721 token X, which owns token bound account Y.
2. Alice deposits 10 ETH into account Y.
3. Bob offers to purchase token X for 11 ETH via a decentralized marketplace, assuming he will receive the 10 ETH stored in account Y along with the token.
4. Alice withdraws 10 ETH from the token bound account, and immediately accepts Bob's offer.
5. Bob receives token X, but account Y is empty.

PichiMarketplace does not implement any mitigations for such fraudulent behavior on the contract level. Pichi implements the following alerts in the application's UI (frontend) to inform and protect users of its marketplace:

- A TBA owner who wants to list their TBA for sale on the PichiMarketplace, will be warned by Pichi's UI prompts that they should withdraw all the tokens owned by the TBA before listing it for sale.

- On the buyer side, there are prompts to inform the buyer that the owner of the TBA they are bidding on can withdraw tokens before accepting the offer and that they should make their offer solely on the Pichi points accrued in that TBA.

VULNERABILITIES & FUNCTIONAL ISSUES

This section details issues affecting the functionality of the contract. Dedaub generally categorizes issues according to the following severities, but may also take other considerations into account such as impact or difficulty in exploitation:

Category	Description
CRITICAL	Can be profitably exploited by any knowledgeable third-party attacker to drain a portion of the system's or users' funds OR the contract does not function as intended and severe loss of funds may result.
HIGH	Third-party attackers or faulty functionality may block the system or cause the system or users to lose funds. Important system invariants can be violated.
MEDIUM	Examples: • User or system funds can be lost when third-party systems misbehave. • DoS, under specific conditions. • Part of the functionality becomes unusable due to a programming error.
LOW	Examples: • Breaking important system invariants but without apparent consequences. • Buggy functionality for trusted users where a workaround exists. • Security issues which may manifest when the system evolves.

Issue resolution includes "dismissed" or "acknowledged" but no action taken, by the client, or "resolved", per the auditors.

CRITICAL SEVERITY:

[No Critical Severity Issues]

HIGH SEVERITY:

[No High Severity Issues]

MEDIUM SEVERITY:

[No Medium Severity Issues]

LOW SEVERITY:

ID	Description	STATUS
L1	Missing check in initialize function of PichiMarketplace	**RESOLVED**
	`PichiMarketplace::setMarketplaceFee` checks that the `newFee <= 1000`, but this is not enforced in `PichiMarketplace::initialize` for the `marketplaceFee_` parameter.	
L2	Missing check in constructor of PichiHelper	**RESOLVED**
	`PichiHelper`'s `setDepositFee` function checks that the `newDepositFee <= 500`, but this is not enforced on the `_depositFee` parameter of the constructor.	
L3	Occurrence of owner mismatch when using PichiHelper::createWallet requires manual intervention to unblock	**RESOLVED**
	`PichiHelper::createWallet` checks whether the owner of the `newWallet` != `msg.sender` and reverts if this is the case. Our understanding is that this can occur if	

another user coincidentally generates the same address when creating his own tokenbound account (potentially when using a different NFT contract).

However if this call reverts, the currentIndex of the NFT contract will not advance, because `_mint` will also revert, resulting in this error being reproduced during subsequent calls. Hence the wallet creation mechanism will be stuck until someone goes into the NFT contract and manually mints a dummy NFT to skip the problematic index.

Resolution:
The issue was addressed by minting a dummy NFT (at no cost to the user) whenever the address which would be used by createWallet already contains some code. The address is then generated again using the following NFT index as salt.

L4	Fee percentage's precision should be hardcoded	RESOLVED

The variables `PichiHelper::feePrecision` and `PichiMarketplace::precision`, which represent the fee percentage's precision (at 100%), can be initialized to an arbitrary value, possibly to something even smaller than the actual fee percentage. A definitive value should be chosen and hardcoded to increase the contracts' clarity.

L5	PichiMarketplace implementation contract's initializer is not disabled	RESOLVED

The initializer function of the PichiMarketplace implementation contract should be disabled as suggested by the OpenZeppelin best practices. This can be done by calling the Initializable::_disableInitializers() function in the PichiMarketplace constructor.

```
constructor() {
    _disableInitializers();
}
```

L6	The version of the OZ libraries used is not the latest	**ACKNOWLEDGED**
The protocol uses version 4.6 of the OpenZeppelin libraries instead of the latest standard that is 5.0.2.		

CENTRALIZATION ISSUES:

It is often desirable for DeFi protocols to assume no trust in a central authority, including the protocol's owner. Even if the owner is reputable, users are more likely to engage with a protocol that guarantees no catastrophic failure even in the case the owner gets hacked/compromised. We list issues of this kind below. (These issues should be considered in the context of usage/deployment, as they are not uncommon. Several high-profile, high-value protocols have significant centralization threats.)

ID	Description	STATUS
N1	PichiMarketplace is upgradeable	**ACKNOWLEDGED**
The PichiMarketplace contract inherits from the OpenZeppelin OwnableUpgradeable contract meaning that the protocol owners are able to upgrade its implementation.		
N2	PichiHelper's owner controls the ERC-6551 account creation	**ACKNOWLEDGED**
PichiHelper's purpose is to ease the process of minting PichiWalletNFTs and of the creation of ERC-6551 accounts owned by the minted NFTs. This means that the PichiHelper owner defines the ERC-6551 registry, proxy and implementation used for the tokenbound account creation. The owner can also upgrade the proxy and implementation used for the creation at any time via the `updateImplementation` and `updateProxy` functions.		

OTHER / ADVISORY ISSUES:

This section details issues that are not thought to directly affect the functionality of the project, but we recommend considering them.

ID	Description	STATUS
A1	Improve naming of OrderAlreadyCancelled error in PichiMarketplace	**RESOLVED**
	`PichiMarketplace::cancelOrdersForCaller` reverts with the error `OrderAlreadyCancelled()` both if the order has been canceled, and if it has been Executed. Consider improving the naming for better error readability.	
A2	Divergence between PichiMarketplace's `executeListing` and `acceptOffer` functions in terms of the settlement currencies available	**ACKNOWLEDGED**
	The `PichiMarketplace` contract has two functions which allow a match to be settled. The `executeListing` function allows a buyer to match a seller's listing, while the `acceptOffer` function allows a seller to match a buyer's offer. However, while executeListing allows settlement in the native token (`ETH`), the acceptOffer function does not. We recommend reviewing the asymmetry between these two methods of matching and to use a consistent settlement method.	
A3	PichiMarketplace's use of WETH as a placeholder for ETH means WETH can never be a settlement currency.	**RESOLVED**
	The `PichiMarketplace` contract expects the currency of a listing to be set to `WETH` as a placeholder, whenever the listing actually needs to be settled in `ETH`.	

For instance `PichiMarketplace::validateListing` enforces the constraint that if the listing currency is `WETH`, then the `msg.value` (denominated in `ETH`) should be equal to the order amount.

This assumption is also present in `PichiMarketplace::executeListing`, which will always call `_transferWalletForPayment` with the `isETH` parameter set to true whenever the listing's order currency is set to `WETH`.

This means that a listing can never be paid in actual `WETH`, but can be paid in any other allowed token. We recommend considering whether this is the intended behaviour of the system, as other placeholders such as `address(0)` could be used for this purpose.

Resolution:
The logic of the contract was modified to allow for payments in WETH whenever WETH is listed as the currency of the listing and msg.value == 0.

A4	Storage variables can be made immutable	**RESOLVED**

The storage variables `pichiWalletNFT`, `erc6551Registry` and `feePrecision` of the `PichiHelper` contract can be made immutable.

A5	PichiMarketplace does not need to inherit from Initializable	**RESOLVED**

PichiMarketplace does not need to inherit from the OZ Initializable contract, as the OwnableUpgradeable contract already does so.

A6	PichiMarketplace::cancelOrdersForCaller reverts if order has already been canceled.	**RESOLVED**

The function `cancelOrdersForCaller` of the PichiMarketplace contract reverts if `orderNonces[i] <= userMinOrderNonce[msg.sender]` holds or if the i-th order

has already been canceled or executed via the `isUserNonceExecutedOrCancelled` mapping. In other words, if the set of orders to be canceled contains at least one canceled or executed order, the whole tx reverts. We consider this unnecessary and possibly confusing. If the tx reverts for the aforementioned reason and the caller/user does not realize it in order to resubmit a valid set of orders to be canceled, their orders will remain open when this is clearly not the desired behavior.		
A7	Extra space in domainSeparator computation in PichiMarketplace::initialize	**RESOLVED**
When the `domainSeparator` is computed by `PichiMarketplace::initialize` there is a space between `chainId` and `address` when there should not be one. We do not believe this is a real problem except if there is such a difference between the front and backend code.		
A8	Use of reinitializer(1) instead of initializer PichiMarketplace::initialize	**RESOLVED**
`PichiMarketplace::initialize` could use the `reinitializer(1)` modifier instead of the `initializer` modifier. This is to be consistent with future upgrades where you seem to intend to use `reinitializer(2)`.		
A9	Compiler bugs	**INFO**
The code is compiled with Solidity `0.8.13`. Version `0.8.13`, in particular, has some known bugs, which we do not believe affect the correctness of the contracts.		

DISCLAIMER

The audited contracts have been analyzed using automated techniques and extensive human inspection in accordance with state-of-the-art practices as of the date of this report. The audit makes no statements or warranties on the security of the code. On its own, it cannot be considered a sufficient assessment of the correctness of the contract. While we have conducted an analysis to the best of our ability, it is our recommendation for high-value contracts to commission several independent audits, a public bug bounty program, as well as continuous security auditing and monitoring through Dedaub Security Suite.

ABOUT DEDAUB

Dedaub offers significant security expertise combined with cutting-edge program analysis technology to secure some of the most prominent protocols in DeFi. The founders, as well as many of Dedaub's auditors, have a strong academic research background together with a real-world hacker mentality to secure code. Protocol blockchain developers hire us for our foundational analysis tools and deep expertise in program analysis, reverse engineering, DeFi exploits, cryptography and financial mathematics.