SOLIDIFIED

Audit Report for Pixowl The Sandbox Token (ERC20) and Sale on September 25th, 2019.

Summary

Audit Report prepared by Solidified for Pixowl covering the Sandbox ERC20 and Sale smart contracts (and inherited dependencies).

Process and Delivery

Three (3) independent Solidified experts performed an unbiased and isolated audit of the code below. The debrief took place on September 25th, 2019, and the final results are presented here.

Audited Files

The following contracts were covered during the audit:

- Sand.sol
- TheSandbox712.sol
- ERC20BaseToken.sol
- ERC20ExecuteExtension.sol
- NativeMetaTransactionProcessor.sol
- Admin.sol
- ProxyImplementation.sol
- SuperOperators.sol
- ERC1271.sol
- ERC1271Constants.sol
- ERC20Events.sol
- BytesUtil.sol
- SafeMath.sol
- SigUtil.sol
- SandSale.sol

Notes

The audit was performed on commit `246b773d1045e657cad3ca661741302b07485d1d` of repository https://github.com/pixowl/contracts_common, and commit `15c875fe37ed5e51ad5e273b8148754b917398fc` of repository https://github.com/pixowl/sandbox-private-contracts, using Solidity compiler version `0.5.9`.

SOLIDIFIED

Audit Report for Pixowl The Sandbox Token (ERC20) and Sale on September 25th, 2019.

Intended Behavior

The contracts implement the Sand ERC20 token, and its crowdsale.

Issues Found

Critical

No critical issues found.

Major

No major issues found.

Minor

1. ERC20 Standard Compliance

The current contract does not fully comply with the ERC20 standard in the following instances:

- The contract does not allow for transfers of zero value, but the standard requires that these are processed normally: "Note Transfers of 0 values MUST be treated as normal transfers and fire the `Transfer` event."
- The contract has functionality to perform approvals without emitting the `Approve` event: "Approval: MUST trigger on any successful call to `approve(address _spender, uint256 _value)`". This is used only on external calls, with temporary approvals to the callee.

Recommendation
Consider amending the items above, to ensure full compliance with the standard.

Amended [30.09.2019]

The contract was amended to accept transfers with 0 value per the standard. Pixowl provided the following response for the approve event, witch we find acceptable and consider it amended:

"ERC-20 is quite open to interpretation and I would argue that our temporary approval mechanism is conformant.

To quote the standard on `transferFrom` : "The function SHOULD throw unless the _from account has deliberately authorized the sender of the message via some mechanism."

If we consider the temporary approval as an alternative mechanism to the `approve` call then the transferFrom happening as part of a meta-tx would still be conformant.

And such alternative mechanism has no obligation to emit an Approval event. The ERC20 specify the following only : "MUST trigger on any successful call to approve(address _spender, uint256 _value)."

2. Points of centralization

Super operators are allowed to transfer and burn tokens from users at will. Execute operators, although designed to be used along with the Native Meta transaction contract, could be granted to an EOA, allowing the address to execute any transaction on behalf of the token contract.

Although we understand that the game design might require a central authority, a better design would be to require users to approve operators from moving funds, and using the native ERC20 transactions to perform them. This will ensure only authorized (by the user) entities will move funds from users.

Execute Operators were designed to execute meta transactions on behalf of users. In the current implementation the Execute Operator is responsible to check the signature, fact that allows for granting access to an EOA, or to a smart contract that verifies the signatures incorrectly.

Lastly, all external calls performed for meta transactions will be executed from the token contract. This means contracts that use msg.sender will see the token contract as the sender of the transaction, a fact that can tamper with results of transactions involving ownership. For example, any user would be able to move any token balance this contract has, or accidentally have the token contract receive tokens on their behalf (tokens that could be transferred by any other user).

Recommendation

We recommend reconsidering the super operator design, switching to the native approve/transferFrom ERC20 functionality that is widely known, tested and trustless. Also consider tying together the execution of meta transactions with the signature verification, ensuring all transactions executed were previously signed by the user, also preventing accounts with full execution rights (that could bypass signature verification).

Pixowl's response [30.09.2019]

"Similarly to our Asset smart contract, the end goal is to remove these responsibility down the line. All of these rights will be managed from a hardware wallet based multi-sig in the mean time.

- Super Operators allow us to add new contract to our system without requiring users to make an tx to approve it, improving the experience
- Execution Operators allow us to support new Meta-tx standard or modify our current one if we find issues with it.

We agree that there is a centralisation weakness but this is by design. We plan to remove the centralisation once the platform is ready. So basically, once all the smart contracts required for the platform are all linked together through these superOperators rights, we would be able to remove admin rights and any future contracts would have to be approved by the users themselves.

The reason behind that choice is to remove the need for our users to pre-approve contracts. This is especially important for meta-tx as we want these meta-tx to enable users without ether to participate on our platform. If these have to first get ether to approve our contract, this defeat one of the main purposes of meta-tx.

The reason we have the ability to add/remove meta tx processors is that it is too early to decide what the best design for those and we want thus to have flexibility to change them. And that's why we decided to move the processing out of the Sand smart contract and leave there only the strict minimum to perform the operation at optimal gas cost."

3. Unchanged allowances in external calls

The function `_approveAndExecuteWithSpecificGas` captures a snapshot of the existing allowances, then it executes an external call, and once it succeeds it restores the initial allowed amount. This structure will make any changes to the allowances that happen in the external call, for example a transferFrom or approve function, to be ignored.

Recommendation

This behaviour seems counterintuitive, and could be removed, or apply a new allowance before the external call is made.

Amended [18.10.2019]
The issue was fixed and is no longer present in commit `a3446efcb78af14beb439722e6f9edc2ed72806d`.

Note: This bug was found by Pixowl team during the audit process.

Notes

4. Hardcoded Number of Decimals in ERC20BaseToken

The `decimals()` method in the `ERC20BaseToken` contract always returns the number 18. While 18 decimals are common practice, the method is unnecessary if the number is hardcoded.

Recommendation
We recommend defining the number of decimals in the derived contracts.

5. Signature Verification does not Check for Malleable Addresses

The signature verification does not check for malleable addresses which are still allowed by `ecrecover`.

Recommendation
We recommend adding checks for malleable addresses. Here is an example of how to check for this:
https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/cryptography/ECDSA.sol

Follow up [30.09.2019]
This issue was previously reported as major, but lowered to a note after discussion with Pixowl. Although vulnerable to transaction malleability, the contract does not rely on the signature to determine uniqueness, and therefore the impact of an exploit is null.

6. Sale contract assumes amount in decimals in ETH but not in DAI

The sale contract can receive the required amount in DAI or in ETH. However, the ETH sale function receives the amount in Wei, whereas the DAI method does not consider the decimals in its parameters, requiring an internal multiplier of 10**18. This is not documented and may be confusing.

Recommendation
We recommend adjusting the buySandWithDai() method to receive the amount in decimals or, alternatively, documenting the parameters correctly.

Amended [30.09.2019]
The issue was fixed and is no longer present in commit `a3446efcb78af14beb439722e6f9edc2ed72806d`.

7. Use require instead of assert in SafeMath

Although Safemath is not used to validate inputs several times in the current codebase, it often is, and might be in future instances of the smart contracts. Reverting the transactions through a `require` is the appropriate way to revert on input validation, while also reimbursing the unused gas back to the user.

Amended [30.09.2019]
The issue was fixed and is no longer present in commit `e1d3ade1876923f6c6f89778efaaac2c9c1454c8`.

Closing Summary

The contracts audited contain couple of minor issues.

We recommend the issues are amended, while the notes are up to the client's discretion, as they mainly refer to improving the operation of the smart contract.

Disclaimer

Solidified audit is not a security warranty, investment advice, or an endorsement of the Pixowl platform or its products. This audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

The individual audit reports are anonymized and combined during a debrief process, in order to provide an unbiased delivery and protect the auditors of Solidified platform from legal and financial liability.

Solidified Technologies Inc.