SMART CONTRACT AUDIT (HTTPS://BLOG.COINFABRIK.COM/CATEGORY/SMART-CONTRACTS/SMART-CONTRACT-AUDIT-SMART-CONTRACTS/)

DreamTeam Smart Contracts Audit for Players' Compensation

Ariel Yabo (https://blog.coinfabrik.com/author/ariel-yabo/)

April 3, 2018 (https://blog.coinfabrik.com/smart-contracts/smart-contract-audit-smart-contracts/dreamteam-smart-contract-for-players-compensation/)

Coinfabrik was asked to audit the smart contracts for the DreamTeam system. The DreamTeam contracts implement a system that teams can use to compensate their members with tokens. In the first part, we will give a summary of our discoveries and follow them with the details of our findings.

Contents

Summary

The contracts audited are from the DreamTeam source code repository (https://github.com/dreamteam-gg/smart-contracts) at commit c1efcabf8a56a4d0a1fa5185a095052c110bdd87 (https://github.com/dreamteam-gg/smart-contracts/commit/c1efcabf8a56a4d0a1fa5185a095052c110bdd87).

The audited contracts are:

- SafeMath.sol: Safe integer calculations
- BasicStorage.sol: Multiple ownership contract base

- DreamTeamStorage.sol: Generic storage of primitive types, deployed separately uses BasicStorage for ownership.
- StorageInterface.sol: Interface for DreamTeamStorage
- TeamStorageController.sol: Functions to abstract high level types for storage in DreamTeamStorage
- TeamContracts.sol: Inherits TeamStorageController. Implements token payments to team members and general management of teams.
- ERC20TokenInterface.sol: ERC20 Token Interface
- TDTT.sol: Simple ERC20 Token implementation

These checks are performed:

- Misuse of the different call methods: call.value(), send() and transfer().
- Integer rounding errors, overflow, underflow and related usage of SafeMath functions.
- Old compiler version pragmas.
- Race conditions such as reentrancy attacks or front running.
- Misuse of block timestamps, assuming things other than them being strictly increasing.
- Contract softlocking attacks (DoS).
- Potential gas cost of functions being over the gas limit.
- Missing function qualifiers or misuse of them.
- Fallback functions with higher gas cost than a what a transfer or send call allows.
- Underhanded or erroneous code.
- Code and contract interaction complexity.
- Wrong or missing error handling.
- Overuse of transfers in a single transaction instead of using withdrawal patterns.
- Insufficient coverage of function input requirements.

None of the issues found was of critical severity.

Detailed findings

Medium severity

Ambiguous return value

There is a function called *storageGetTeamMemberIndexByAddress*, it returns the member index given a member address:

```
function storageGetTeamMemberIndexByAddress (address db, uint teamId, address memberAddress)
internal view returns(uint) {

    uint i = 0;

    address addr;

    do {

        addr = StorageInterface(db).getAddress(keccak256(Storage.teams, teamId, i));

        if (addr == memberAddress)

            return i;

        if (addr == 0x0)

            return 0x0;

        ++i;

    } while (true);

}
```

The problem with this function is that it does not distinguish between returning 0 because nothing was found and returning the member with index 0. This means that *removeMember* will remove the member index 0 if an invalid address is given:

```
function removeMember (uint teamId, address memberAccount) dreamTeamOnly public {


uint memberIndex = storageGetTeamMemberIndexByAddress(db, teamId, memberAccount);

uint payoutDate = storageGetTeamMemberPayoutDate(db, teamId, memberIndex);


(...)



// Actually delete team member from a storage

storageDeleteTeamMember(db, teamId, memberIndex);

TeamMemberRemoved(teamId, memberAccount);

}
```

Consider returning a tuple with an index and a bool that is false for the case when nothing is found.

This was fixed by the DreamTeam developers in commit *dbb8182f40eb15e2cd73b6f65014a280057fbaf4* (https://github.com/dreamteam-gg/smart-contracts/commit/dbb8182f40eb15e2cd73b6f65014a280057fbaf4).

Minor severity

Vesting mechanism does not emit Transfer events

The *TDTT* contract does not emit Transfer events when balances are assigned to token holders as a consequence of its vesting mechanism. This could make it difficult for someone to examine the historical token movements in a block explorer. In particular, it poses an inconvenience to users of applications that depend on these events to interact with the token contract.

Due to the nature of the mechanism, we recommend emitting a Transfer event once the balance of an address is updated according to the current block in the *subFromBalance* function.

Enhancements

These are presented as mere suggestions to the smart contract development team. They may help with the smart contract system overall.

Addresses can be declared as contracts

Lines like these:

```
ERC20TokenInterface(erc20TokenAddress).transfer(newDeployedTeamContracts,
ERC20TokenInterface(erc20TokenAddress).balanceOf(this)); // Move all funds to a new contract
```

Can be simplified by making the variable *erc20TokenAddress* an *ERC20TokenInterface* instead of an untyped address:

```
ERC20TokenInterface erc20TokenAddress;
```

Now the line above can be turned to:

```
erc20TokenAddress.transfer(newDeployedTeamContracts,erc20TokenAddress.balanceOf(this));
// Move all funds to a new contract
```

The same can be done with contracts that are not interfaces.

Add token transfer failsafe mechanisms

We recommend adding failsafe mechanisms to ensure tokens don't get stuck indefinitely in any of the contracts involved. While this makes for a nice feature in user-facing smart contracts, in this case it may be a good idea to have it considering the existence of an upgrade mechanism. The failsafe could even work for the DreamTeam token itself in the *TeamContracts* contract with adequate restrictions.

An easy way to implement such a failsafe is to write a function like this one:

```
 function tokenFailsafeTransfer(address agent, uint tokens, ERC20 token_contract)
public ownersOnly {

// We use approve instead of transfer to minimize the possibility

// of getting them stuck in another address by accident.

token_contract.approve(agent, tokens);

 }
```

This implementation could be used as is in *DreamTeamStorage*, but would require some tweaks if added in *TeamContracts*.

Conclusion

All the audited contracts are simple, straightforward and have an adequate amount of documentation to easily understand what they do. In particular, we haven't found any important issues that we consider critical.

Do you want to know what is Coinfabrik Auditing Process?

Check our A-Z Smart Contract Audit Guide (https://blog.coinfabrik.com/smart-contract-audits-ultimate-security-guide/) or you could request a quote (https://www.coinfabrik.com#contact_us) for your project.

Related Posts

(https://blog.coinfabrik.com/smart-contracts/smart-contract-audit-smart-contracts/rcn-smart-contracts-audit-v2/)

RCN Smart Contracts Audit v2 (https://blog.coinfabrik.com/smart-contracts/smart-contract-audit-smart-contracts/rcn-smart-contracts-audit-v2/)

The smart contracts that have been audited were taken from the RCN repository at: https://github.com/ripio/rcn-network/tree/v2.…

(https://blog.coinfabrik.com/smart-contracts/etherparty-token-smart-contract-security-audit-coinfabrik/)

EtherParty Smart Contract Security Audit (https://blog.coinfabrik.com/smart-contracts/etherparty-token-smart-contract-security-audit-coinfabrik/)

Coinfabrik team has been hired to audit Etherparty smart contracts. Firstly, we will provide a…

(https://blog.coinfabrik.com/smart-contracts/cryptocup-audit/)

CryptoCup: Smart Contract Audit (https://blog.coinfabrik.com/smart-contracts/cryptocup-audit/)

Coinfabrik was asked to audit the contracts for the CryptoCup ERC721 Token. Firstly, we will…

Tags:

dreamteam (https://blog.coinfabrik.com/tag/dreamteam/)

gaming (https://blog.coinfabrik.com/tag/gaming/)

players (https://blog.coinfabrik.com/tag/players/)

Token (https://blog.coinfabrik.com/tag/token/)

SHARE ON

(https://www.facebook.com/sharer/sharer.php?u=https://blog.coinfabrik.com/smart-contracts/smart-c contracts/dreamteam-smart-contract-for-players-compensation/)
(https://twitter.com/intent/tweet?text=DreamTeam%20Smart%20Contracts%20Audit%20for%20Players'%20Compensation&url=https://blc contracts/smart-contract-audit-smart-contracts/dreamteam-smart-contract-for-players-compensation/)
(https://pinterest.com/pin/create/button/?url=&media=https://blog.coinfabrik.com/wp-content/uploads/2018/04/dreamteam.png&description=DreamTeam+Smart+Contracts+Audit+for+Play
(https://www.linkedin.com/shareArticle?mini=true&url=https://blog.coinfabrik.com/smart-contracts/sma contracts/dreamteam-smart-contract-for-players-compensation/&title=DreamTeam%20Smart%20Contracts%20Audit%20for%20Players'%20Compensatio

← PREVIOUS ARTICLE

RCN Smart Contracts Audit v2 (https://blog.coinfabrik.com/smart-contracts/smart-contract-audit-smart-contracts/rcn-smart-contracts-audit-v2/)

NEXT ARTICLE →

Cryptosolartech Security Audit (https://blog.coinfabrik.com/smart-contracts/smart-contract-audit-smart-contracts/cryptosolartech-security-audit/)

You may also like

(https://blog.coinfabrik.com/smart-contracts/magic-bridge-audit/)

Magic Bridge Audit (https://blog.coinfabrik.com/smart-contracts/magic-bridge-audit/)

(https://blog.coinfabrik.com/smart-contracts/smart-contract-audit-smart-contracts/mintingfactoryv2-baseupgradablemarketplace-kodav3upgradablegatedmarketplace/)

2 months ago

Smart Contract Audit (https://blog.coinfabrik.com/category/smart-contracts/smart-contract-audit-smart-contracts/)

MintingFactoryV2, BaseUpgradableMarketplace & KODAV3UpgradableGatedMarketplace (https://blog.coinfabrik.com/smart-contracts/smart-contract-audit-smart-contracts/mintingfactoryv2-baseupgradablemarketplace-kodav3upgradablegatedmarketplace/)

(https://blog.coinfabrik.com/feed/)

(https://ar.linkedin.com/company/coinfabrik)

(https://twitter.com/coinfabrik)

(https://www.youtube.com/channel/UC2GmjCr7aEz-il31kqOy9aw)

(https://www.facebook.com/CoinFabrik/)

(https://www.reddit.com/r/CoinFabrik/)

(https://github.com/coinfabrik)