Smart Contract Source Code Audit

Prepared for Olyseum • Sep 2019

v0923

1. Table Of Contents

© 2019 Coinspect

2. Executive Summary

In May 2019 Olyseum engaged Coinspect to perform a source code review of the OLY token smart contracts. In total, the review comprised three parts: a first review (early May), a second review of the code including fixes for the issues found in the first review (by the end of May), and a third review with the contracts ready for deployment (September 2019). The objective of the audit was to evaluate the security of the smart contracts. During the assessment, Coinspect identified the following issues:

High Risk	Medium Risk	Low Risk	Zero Risk
-	2	7	-

No critical vulnerabilities were found, and all issues were taken care of. The seven low risk issues didn't pose an immediate threat, but general security was improved by addressing them. The two medium risk issues could have led to unexpected behaviour and loss of funds, but they were addressed. In the case of OLY-008, it was not considered a bug and Olyseum decided to address it by making clear in the contract source code what's the expected behaviour of the contract.

Additionally, it is recommended to implement a battery of tests to exercise all contracts and verify their expected behaviour and functionality.

Finally, it is worth mentioning that the contracts are not fully autonomous, and rely on transactions by accounts controlled by Olyseum and whom the users must trust.

© 2019 Coinspect

3. Introduction

The OLY token is an ERC20 compatible token with additional support for deferred transactions (messages that have been signed off-chain and authorize the execution of a transfer from the signer's account). The token also has upgradeability features via a proxy contract, and the bundle also includes other contracts that manage campaigns and funds in OLY tokens.

Coinspect was provided with a snapshot of the repository. The scope of the audit was limited to the following Solidity source files (shown here with their sha256 hash):

```
0a09375ad95a3dcfeb6f1c0228905c1c61118c6c7d6d88b6bf25401aba9d647e  Auction.sol
0755a1f5c6bfde34fed678f0773f9d3e85b8a02f938696ac5072eb21108ccb88  CampaignFund.sol
1febe43b444571599185f6aa74b181af76c2667f0c4772a90713fb19f505bf4b  CampaignRegistry.sol
4df362a1b0d74cfe06ba76c1889a48991fd2c6d3def38af3e4269a03ac4d28d0  IERC20.sol
da1001059e166e9456c786425944e1fb0a78aae7478aa3db4dc48401ebefb855  MarketPlacePool.sol
fa75773246e5e001b78ac8b8261f6600ef92bb0b175e711a77372a3ba3d4ee0c  MerchantBudget.sol
32b75ac2dd7f6f978a2a6aa3d0dc8983c211751e44ec134c6f9135fe0407f44d  Migrations.sol
7923689e758b6ec8687c4d739e448ac52e0769e244fb0470cb99fa418ea524b6  MultiSigWallet.sol
dab3ad5a4f6368bfddc40b79c6bbe1b2759ec28d786fbd158f4a5da339609b32  OlyToken.sol
cb4c8e0cae901786771b09cf89acdad826a703a6075367eaf11fb1997b468e31  Ownable.sol
3911a0c11ae5f6ee8e03760c2e104c497fb351131e31792f602a12a972a3a653  Proxy.sol
bc466218135b71e6b0a7f66e4a8c8d468cf1993ce8ccb88d32f2055421e8b4f2  SafeMath.sol
```

After a report draft was sent to Olyseum, the issues found during the assessment were addressed and in May 28th Olyseum provided Coinspect with updated contracts:

```
e703a384f773295cdee281ffb2d2683ea3aac65c9a77950498f31cc479d3b887  CampaignFund.sol
f0b43e3f940f98bedb50e8694bb2166956d661dc4ea1c5e17db838ddff2125d8  CampaignRegistry.sol
697afc7d5864379c15a906cd2aba829159b6751c72c3b76568ec5f879453d7dd  IERC20.sol
9ac0d42c59902a2030b323faf3eecc19bf693ed214c6ef9cf12eea16441a0f9a  MarketPlacePool.sol
3cb5d682f2a96e438e6dd075b5930ad876976338fac7afaea2e6bf251f622533  MultiSigWallet.sol
abd950be0db0075414d2956f004c95fb7cbc1230fb188ecfc17a8b81c9e405aa  OlyToken.sol
ceafeaa4ed106e73c60584e843b5907dc35820f18d48ae2c914401e490141a75  Ownable.sol
854c5e97a0d8c0d6af309b5fd5b532dbe60d40e31c55852ca2345aeb32afe31c  Proxy.sol
24f9811e83796943c4360f7ac12065076a4a2ad691da12a3faca0a5cf37e1e83  SafeMath.sol
```

All changes were verified by Coinspect and all issues were found to be correctly addressed following the report recommendations.

After this, Olyseum made more changes with the goal of tighten up the code before public deployment of the smart contracts. Contracts taken from OpenZeppelin were replaced with their original version from OpenZeppelin: `Ownable`, `SafeMath`, `IERC20`, and also `Proxy`. Olyseum communicated to Coinspect that the contracts intended to be deployed are OpenZeppelin's `AdminUpgradeabilityProxy` (originally called `Proxy`) and the 3 following Olyseum contracts:

```
b6c483d15ffe18e72706533008f8586a6d3057db5846ac94cb594e5a587e0285  CampaignFund.sol
aef60ea24446eab9cd08085224fb049b7f19d97a753cbd5252fc930df7a797dc  MultiSigWallet.sol
```

```
e41d226876b284a9e0dbb1acc14e2db273f65c6045524e77eb9a2e1de2baa249  OlyToken.sol
```

In September Coinspect reviewed the three upgraded contracts. The new code includes aesthetic improvements, and minor changes to functionality and logic. No new security issues were found. The changes in the upgraded contracts are:

- `CampaignFund`: changed function `withdrawAll` from `onlyOwner` to `onlyJury`, removed functionality for an ‘emergency’ state and now allows withdrawals at jury discretion, introduced the notion of subcampaigns;
- `MultiSigWallet`: replaced `‘uint’` by `‘uint256’` everywhere, and minor indentation changes;
- `OlyToken`: changed `decimals` from constant to a variable that can be set during initialization, changed functions `_burnFrom` and `transferFrom` to emit an `Approval` event indicating the updated allowance.

4. Summary Of Findings

ID	Description	Risk	Fixed
OLY-001	Outdated Solidity version	Low	✔
OLY-002	Use view and pure instead of deprecated constant	Low	✔
OLY-003	Constructors should use the constructor keyword	Low	✔
OLY-004	Use emit keyword to emit events	Low	✔
OLY-005	Mismatch between ERC-20 specification and OlyToken	Low	✔
OLY-006	Superfluous empty contracts Auction and MerchantBudget	Low	✔
OLY-007	Missing consistency checks in MarketPlacePool constructor	Low	✔
OLY-008	Zeno’s Paradox in MarketPlacePool	Medium	✔
OLY-009	Inconsistencies in MarketPlacePool	Medium	✔

© 2019 Coinspect

5. Findings

OLY-001 Outdated Solidity version

Total Risk	Impact	Location
Low	Low	MultiSigWallet.sol
Fixed ✔	**Likelihood** Low	**Status** Contracts were updated to compile with 0.5.8

Description

Currently, the contract code specifies with the `pragma` statement that it is meant to be built with a version of the Solidity compiler older than the latest production release. Newer versions have added additional warnings that can help to detect problems, solve bugs and enforce new rules to enhance security.

The latest Solidity release is 0.5.8, but the Solidity versions currently specified in the Olyseum contracts are:

```
Auction.sol:pragma solidity ^0.5.3;
CampaignFund.sol:pragma solidity ^0.5.3;
CampaignRegistry.sol:pragma solidity ^0.5.3;
IERC20.sol:pragma solidity ^0.5.0;
MarketPlacePool.sol:pragma solidity ^0.5.3;
MerchantBudget.sol:pragma solidity ^0.5.0;
Migrations.sol:pragma solidity ^0.5.0;
MultiSigWallet.sol:pragma solidity ^0.4.15;
OlyToken.sol:pragma solidity ^0.5.3;
Ownable.sol:pragma solidity ^0.5.3;
Proxy.sol:pragma solidity ^0.5.3;
SafeMath.sol:pragma solidity ^0.5.0;
```

Recommendation

Add the latest version to the pragma statement:

```
pragma solidity ^0.5.8;
```

The notation used above allows for the use of versions between 0.5.8 and 0.6.0.

References

For more information on the use of the version pragma and how to handle the different versions accepted, see the following reference links:

- https://solidity.readthedocs.io/en/develop/layout-of-source-files.html#version-pragma
- https://docs.npmjs.com/misc/semver#versions
- http://solidity.readthedocs.io/en/develop/bugs.html

OLY-002 Use *view* and *pure* instead of deprecated *constant*

Total Risk	Impact	Location
Low	Low	MultiSigWallet.sol
Fixed ✔	**Likelihood**	**Status**
	Low	MultiSigWallet was updated to Solidity 0.5.8

Description

The modifier `constant` for functions has been deprecated, and `view` or `pure` should be used instead.

Recommendation

Just replace `constant` with `view` or `pure` in any function definition that uses the `constant` modifier. Functions they don't modify the state can be marked as `view`, and functions that don't read not modify the state can be marked as `pure`.

References

For more information on the deprecation of the `constant` keyword for functions and the introduction of `view` and `pure`, see https://github.com/ethereum/solidity/issues/992

OLY-003 Constructors should use the *constructor* keyword

Total Risk	Impact	Location
Low	Low	MultiSigWallet.sol
Fixed ✔	**Likelihood**	**Status**
	Low	MultiSigWallet was updated to Solidity 0.5.8

Description

In Solidity versions prior to 0.4.22 the constructor for a contract was a function with the same name as the contract:

```
contract Test {
    address owner;
    function Test() public { owner = msg.sender; }
}
```

But this was dangerous because sometimes the developers would rename a contract and forget to rename the constructor:

```
contract Bird {
    address owner;
    function Test() public { owner = msg.sender; }
}
```

Once the `Bird` contract is deployed, anyone could call the `Test` function and set the owner to his own address, and then call any restricted functions. This type of bug resulted in loss of funds in the real world, such as the case of Rubixi [1]. This is why it was deprecated in favor of the new `constructor` keyword:

```
contract Bird {
    address owner;
    constructor() public { owner = msg.sender; }
}
```

Recommendations

Upgrade MultiSigWallet.sol to the latest version of Solidity (see OLY-001) and replace:

```
function MultiSigWallet(address[] _owners, uint _required) public [...]
```

with the new-style constructor:

```
constructor(address[] _owners, uint _required) public [...]
```

References

[1] Atzei N., Bartoletti M., Cimoli, T.: A Survey of Attacks on Ethereum Smart Contracts, Oct, 2016.

OLY-004 Use *emit* keyword to emit events

Total Risk	Impact	Location
Low	Low	MultiSigWallet.sol
Fixed ✔	**Likelihood**	**Status**
	Low	MultiSigWallet was updated to Solidity 0.5.8

Description

Since version 0.4.21, Solidity uses the `emit` keyword for emitting events. This was done in order to make clear the difference between calling functions and emitting events. For example, in version previous to 0.4.21 one would emit a Transfer event like this:

```
Transfer(from, to, value);
```

While in newer versions of Solidity one must do:

```
emit Transfer(from, to, value);
```

Recommendations

Always use the `emit` keyword for emitting events.

References

Proposal: add `emit` keyword; make it the only way to emit events
https://github.com/ethereum/solidity/issues/2877

OLY-005 Mismatch between ERC-20 specification and OlyToken

Total Risk	Impact	Location
Low	Low	OlyToken.sol
Fixed ✔	Likelihood: Low	Status: Type changed to uint8

Description

The `OlyToken` contract, which is intended to be ERC-20 compliant, defines the `decimals` public state variable to be `uint256`. This declaration does not comply with the ERC-20 specification, which defines the `decimals` as a `uint8` variable.

Recommendations

Consider changing the `decimals` variable to `uint8` to match the ERC20 specification.

References

ERC20 specification https://github.com/ethereum/EIPs/blob/master/EIPS/eip-20.md

OLY-006 Superfluous empty contracts Auction and MerchantBudget

Total Risk	Impact	Location
Low	Low	Auction.sol, MerchantBudget.sol
Fixed ✔	Likelihood: Low	Status: Contracts Auction.sol and MerchantBudget.sol were removed

Description

The `MerchantBudget` contract, defined in `MerchantBudget.sol`, is never used and it is empty:

```
pragma solidity ^0.5.0;

import “./SafeMath.sol”;

contract MerchantBudget
{

}
```

Same with Auction.sol. These files seem to be a leftovers from development.

Recommendations

Remove the superfluous files `Auction.sol` and `MerchantBudget.sol`.

OLY-007 Missing consistency checks in MarketPlacePool constructor

Total Risk	Impact	Location
Low	Low	MarketPlacePool.sol
Fixed ✔	Likelihood: Low	Status: Percentage validated at constructor and setPercentage

Description

The `MarketPlacePool` contract keeps a balance of tokens and allows periodic withdrawals of a given percentage. The percentage is initially set by the constructor:

```
constructor (address _olyTokenAddr, address _jury, uint _percentage, uint
_period) public {
    olyToken = IERC20(_olyTokenAddr);
    percentage = _percentage;
    period = _period;
    nextWithdraw = now.add(period);
    emergency = false;
    jury = _jury;
}
```

and can later be modified by calling the function `changePercentage`:

```
function changePercentage(uint _percentage) public onlyJury returns (bool)
{
    require(_percentage < 101, "The percentage cannot be higher than 100");
    percentage = _percentage;
    return true;
}
```

Notice that the constructor is missing the check that is performed in function `changePercentage`.

Recommendations

Consider adding a check to the constructor to make sure the provided percentage doesn't exceed 100. Also, consider checking that the percentage is not 0 in both the constructor and the function `changePercentage`, since 0 percentage doesn't make sense either (it allows periodic withdrawals of 0 tokens).

OLY-008 Zeno's Paradox in MarketPlacePool

Total Risk	Impact	Location
Medium	Medium	MarketPlacePool.sol
Fixed ✔	Likelihood: High	Status: Behavior is by-design, added comment to clarify contract behaviour

Description

The `MarketPlacePool` contract keeps a balance of tokens and allows periodic withdrawals of a given percentage by calling the function `periodicWithdraw`:

```
function periodicWithdraw() public onlyOwner returns (bool) {
    require(nextWithdraw < now, "The period of time isn't over yet");
    uint256 time_delta = now.sub(nextWithdraw);
    uint256 periods_until_next_withdraw = time_delta.div(period).add(1);
                                                    nextWithdraw            =
nextWithdraw.add(periods_until_next_withdraw.mul(period));

    uint subtotal = Balance().mul(percentage).div(100);
    require(olyToken.transfer(jury, subtotal), "Token transfer failed.");

    return true;
}
```

Each time this function is called, the balance is decreased by a percentage. This means that, unless the percentage is 100, the balance will never reach 0.

For example, if the balance is 1000 and the percentage is set to 20, the balance will not be 0 after 5 withdrawals. This is what the balance would look after 10 periodic withdrawals:

#	Withdrawal amount	Balance after withdrawal
1	200	800
2	160	640
3	128	512
4	102.4	409.6
5	81.92	327.68
6	65.536	262.144
7	52.4288	209.7152
8	41.94304	167.77216
9	33.554432	134.217728
10	26.8435456	107.3741824

In fact, in order to get the balance below 1 in this example it would take 31 periods (and 31 withdrawals), not 5 as a naive reading of the code would suggest. The problem exacerbates with smaller percentages (for 5 percent it would require 135 periods to get the balance below

1). And in any case, the balance would never reach 0 (except once it reaches the precision limit of 256 bits, that would take a long time).

Recommendations

Consider reimplementing the withdrawal logic in order to avoid Zeno's paradox.

OLY-009 Inconsistencies in MarketPlacePool

Total Risk	Impact	Location
Medium	Medium	MarketPlacePool.sol
Fixed ✔	Likelihood: Medium	Status: Jury is now, consistently, the destination of both emergency and periodic withdrawals

Description

The contract `MarketPlacePool` keeps a balance of tokens and allows periodic withdrawals of a given percentage by calling the function `periodicWithdraw`:

```
function periodicWithdraw() public onlyOwner returns (bool) {
    require(nextWithdraw < now, "The period of time isn't over yet");
    uint256 time_delta = now.sub(nextWithdraw);
    uint256 periods_until_next_withdraw = time_delta.div(period).add(1);
    nextWithdraw =
nextWithdraw.add(periods_until_next_withdraw.mul(period));

    uint subtotal = Balance().mul(percentage).div(100);
    require(olyToken.transfer(jury, subtotal), "Token transfer failed.");

    return true;
}
```

The percentage that is transferred in each periodic withdrawal can be set by the *jury* by calling `changePercentage`:

```
function changePercentage(uint _percentage) public onlyJury returns (bool)
{
    require(_percentage < 101, "The percentage cannot be higher than 100");
    percentage = _percentage;
    return true;
}
```

The contract `MarketPlacePool` also allows “emergency withdrawals” by calling the function `emergencyWithdraw`, provided that the *jury* allowed it by first calling the function `allowEmergencyWithdraw`:

```
function allowEmergencyWithdraw() public onlyJury returns (bool) {
    emergency = true;
    return true;
}

function emergencyWithdraw() public onlyOwner returns (bool) {
    require(emergency, "It's not an emergency");
    emergency = false;

    require(olyToken.transfer(owner, Balance()), "Token transfer failed.");
```

```
        return true;
    }
```

Notice that the function `periodicWithdrawal` transfers funds to the *jury*, while `emergencyWithdraw` transfers funds to the *owner* (which is the sender too, since the function is `onlyOwner`).

This is inconsistent. Since the *jury* can set the percentage to be sent in the periodic withdrawals (`changePercentage` is `onlyJury`) and cannot perform periodic withdrawals himself (`periodicWithdraw` is `onlyOwner`), it doesn't make sense for the *jury* to be the recipient of the transfers in `periodicWithdraw`.

Recommendations

Change the recipient of the transfers in `periodicWithdraw` to *owner*, or reconsider the role of the *jury* and the permissions logic in the contract `MarketPlacePool`.

6. Disclaimer

The present security audit is limited to smart contract code. It does not cover the technologies and designs related to these smart contracts, nor the frameworks and wallets that communicate with the contracts, nor the general operational security of the company whose contracts have been audited. This document should not be read as investment advice or an offering of tokens.