Security Assessment

GYSR

Jun 9th, 2021

Table of Contents

Summary

This report has been prepared for GYSR smart contracts, to discover issues and vulnerabilities in the source code of their Smart Contract as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Manual Review techniques.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

The security assessment resulted in findings that ranged from minor to informational. We recommend addressing these findings to ensure a high level of security standards and industry practices. We suggest recommendations that could better serve the project from the security perspective:

- Enhance general coding practices for better structures of source codes;
- Provide more transparency on privileged activities once the protocol is live.

Overview

Project Summary

Project Name	GYSR
Platform	Ethereum
Language	Solidity
Codebase	b455a66a431b6868a00e67f434c81903ba436f49
Commit	b455a66a431b6868a00e67f434c81903ba436f49, 85f49e3847e0b64353566bd848f18a7b8548cbd3, ce06acf4b889a8ff9a10fe4ebb0773a92b2a4d27, 73278bea761c7d799bd890da66ebc737c49ed912

Audit Summary

Delivery Date	Jun 09, 2021
Audit Methodology	Manual Review
Key Components	

Vulnerability Summary

Total Issues	18
Critical	0
Major	0
Medium	0
Minor	5
Informational	13
Discussion	0

Audit Scope

ID	file	SHA256 Checksum
ERC	ERC20BaseRewardModule.sol	a1c3efc16ed7948ffab750a44d6d4b34c4b45e3f163fa1d5ac04a74127965aab
ERR	ERC20CompetitiveRewardModule.sol	cb4eab244e12b6275adf5be03f274c728eba77c962d1b22a673f6878fae57cf4
ERM	ERC20CompetitiveRewardModuleFactory.sol	6ab91285000de1fca8ebad5d7b51ed7ebea2010ff8e13cf4a069f74fb53b19c4
ERF	ERC20FriendlyRewardModule.sol	b34c65b09230a0fd902c413c911047814d822212e6e8a28e865f8d1d899df0d9
ECF	ERC20FriendlyRewardModuleFactory.sol	2e3e02dfb7786ddf8e1137024caf2742cbdc986e7b848467c6a16ff84c74c04c
ERS	ERC20StakingModule.sol	b0b50fa663e80c46dc6df1b3b9394a8ec6dd84cace8500434cd106f0c865a2dd
ECS	ERC20StakingModuleFactory.sol	490d9cc5eb9f51422bdd732b4de9dcaf05138c4040f08f84555f137524e7ecf3
GTO	GeyserToken.sol	5dd3d4f3d3f223d264a2698044b7744a9005dac968602091b3fecbf5767570db
GUO	GysrUtils.sol	3538eb8f10f46d34545cca109c301d0771a473d0bac6341abf9ad5292e11b397
MUO	MathUtils.sol	9d788694c6a5181a3a4c724d806f2d01d1dca3a5e2797e69ce8ab3f2062d8bab
OCO	OwnerController.sol	01311656889510ec216e26cfa4a01af313154c6c8ac4cbda5eec8455404de96a
POO	Pool.sol	4b6ce1904a6cda4a90c2c4c740eac577eecc0d1c17e62feef41fb91467c6f13b
PFO	PoolFactory.sol	fc3f297489a22283d5c09bb24139eaabb8a876981efc4caa0972e7de58e3d173
ERI	info/ERC20CompetitiveRewardInfo.sol	c8a61b4e0e5a45df5f5fa38753de60a9dc3ea31cace1e287e506f40e4a33faed
PIO	info/PoolInfo.sol	9aefeb4eae05bcdc5e3d6f1a42cfe1617be3f684249339f9771d5e85be398fd1
IEO	interfaces/IEvents.sol	83fc8fe312e626071a51d2d488c81ed5f6defd09a85c1d7e60866d7d3b9094ba
IMF	interfaces/IModuleFactory.sol	3319ac3b868307aa26c425b6436609ebcb5e1738debc0b86a7c02c02606b5f38
IPO	interfaces/IPool.sol	9f2a85fcc8c3dcb0024a91ff29dce7cc5de042655c93c932a937dc7279ac5236
IPF	interfaces/IPoolFactory.sol	96f9a676016e9c959a2c8eabd7a412c82e669177df011bc397505666d91a47c1
IRM	interfaces/IRewardModule.sol	bfdf48d1c39284182d45e0491248a6ff1725b9e077456f6bd27c167bd869e5e3
ISM	interfaces/IStakingModule.sol	f1a81bd63ff067f98e638a3ef42db144364457b8a7db2c4fdfaf1346e7517f6f

Executive summary

Project is inspired by several farming contracts and aims to generalize their capabilities for no-code deployment. These include the Ampleforth and Synthetix contracts. There are similarities between the mentioned projects and GYSR but the code and logic is only loosely based on said projects.

Code is well documented, well thought-out and we haven't found any major or critical issues. The team has provided to us a draft version of their white paper that documents well the product and how protocol is intended to work. The development team uses natspecs comments to describe the code which is a nice addition to the technical description inside white paper.

Staking and Rewards modules can be only called by the owner of the contract which is the pool. All public/external functions that changes the state variables are protected from re-entrancy by `nonReentrant` modifier from `ReentrancyGuard.sol` from OpenZeppelin.

In the Pool contract, there is a withdraw functions that originally was suspicious to us as the controller could get all vested tokens. We got client's response to our concern: "That spent GYSR is designed as an income stream for the pool creator. Once it's vested, it's considered fully used and fair game to withdraw".

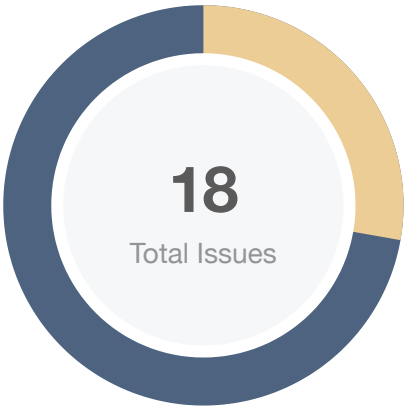
Overall, we're pleased with the quality of the code and the protocol. All the issues with the code can be found below.

System Analysis

Controller of the `PoolFactory.sol` can modify the Treasury/Fee/Whitelist parameters of the contract. In case of lost access to the private key of an account or mishandling security of private keys, an attacker could benefit from that and replace key parameters. We advise that a governance system or multi-signature wallet is utilized instead of a single account in this case.

Please refer to the PFO-01 finding for more details.

Findings



Critical	**0** (0.00%)
Major	**0** (0.00%)
Medium	**0** (0.00%)
Minor	**5** (27.78%)
Informational	**13** (72.22%)
Discussion	**0** (0.00%)

ID	Title	Category	Severity	Status
CON-01	Unlocked Compiler Version	Language Specific	Informational	Resolved
ERC-01	Function returns local variables explicitly.	Gas Optimization, Language Specific	Informational	Resolved
ERF-01	Recurring calls to internal _update() function.	Gas Optimization, Volatile Code	Informational	Resolved
ERF-02	Unnecessary lower than 0 check on uint variable	Language Specific	Informational	Resolved
ERF-03	Redundant Variable Initialization	Coding Style	Informational	Resolved
ERF-04	Unchecked Return Value	Volatile Code	Minor	Acknowledged
ERF-05	Function returns local variables explicitly.	Gas Optimization	Informational	Resolved
ERI-01	Function returns local variables explicitly.	Gas Optimization	Informational	Resolved
ERR-01	Unchecked Return Value	Volatile Code	Minor	Acknowledged
ERR-02	Function returns local variables explicitly.	Gas Optimization, Language Specific	Informational	Resolved
ERS-01	Function returns local variables explicitly.	Gas Optimization, Language Specific	Informational	Resolved
ERS-02	Checks-effect-pattern not applied	Volatile Code	Minor	Resolved

ID	Title	Category	Severity	Status
GTO-01	Centralization concern over total supply of tokens	**Centralization / Privilege**	**Minor**	**Acknowledged**
OCO-01	Lack of input validation	Volatile Code	Informational	Resolved
OCO-02	Lack of ownership renouncement functions	Control Flow	Informational	Acknowledged
OCO-03	User-Defined Getters	Gas Optimization	Informational	Acknowledged
PFO-01	Centralization concern	**Centralization / Privilege**	**Minor**	**Acknowledged**
PFO-02	Lack of input validation	Volatile Code	Informational	Resolved

CON-01 | Unlocked Compiler Version

Category	Severity	Location	Status
Language Specific	Informational	: 1	Resolved

Description

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for version `v0.8.4` the contract should contain the following line:

```
pragma solidity 0.8.4;
```

Alleviation

Issue has been resolved.

ERC-01 | Function returns local variables explicitly.

Category	Severity	Location	Status
Gas Optimization, Language Specific	Informational	ERC20BaseRewardModule.sol: 198, 226	Resolved

Description

Linked functions returns locally created variables explicitly. For gas optimization, named return variables are cheaper

Recommendation

We would suggest to utilize named return variables for gas optimizations. For example

```
function totals() public view override returns (uint256[] memory arr) {
        arr = new uint256[](1);
        arr[0] = _token.balanceOf(address(this));
}
```

Alleviation

Issue has been resolved.

ERF-01 | Recurring calls to internal _update() function.

Category	Severity	Location	Status
Gas Optimization, Volatile Code	Informational	ERC20FriendlyRewardModule.sol: 274~279, 180, 163, 132, 107	Resolved

Description

Functions `_stake()` and `_unstake()` make calls to the `_update()` which is also called at the beginning of the `claim()` function. There's no need to call `_update()` three times before each operations. If that would be needed, we would recommend making a call to `_update()` on the `claim()` function.

Recommendation

We would recommend removing `_update()` call from `_unstake` and `_stake` functions and move that call to the `stake()` and `unstake()` function.

Alleviation

Issue has been resolved.

ERF-02 | Unnecessary lower than 0 check on uint variable

Category	Severity	Location	Status
Language Specific	Informational	ERC20FriendlyRewardModule.sol: 373	Resolved

Description

Linked code for `if (totalStakingShares <= 0)` performs an unnecessary check for lower than 0 on an uint variable which cannot be lower than 0.

Recommendation

We would recommend to change this check to only `if (totalStakingShares == 0)`.

Alleviation

Issue has been resolved

ERF-03 | Redundant Variable Initialization

Category	Severity	Location	Status
Coding Style	Informational	ERC20FriendlyRewardModule.sol: 33~39	Resolved

Description

All variable types within Solidity are initialized to their default "empty" value, which is usually their zeroed out representation. Particularly:\n\n- `uint` / `int`: All `uint` and `int` variable types are initialized at `0`\n- `address`: All `address` types are initialized to `address(0)`\n- `byte`: All `byte` types are initialized to their `byte(0)` representation\n- `bool`: All `bool` types are initialized to `false`\n- `ContractType`: All contract types (i.e. for a given `contract ERC20 {}` its contract type is `ERC20`) are initialized to their zeroed out address (i.e. for a given `contract ERC20 {}` its default value is `ERC20(address(0))`)\n- `struct`: All `struct` types are initialized with all their members zeroed out according to this table

Recommendation

We advise that the linked initialization statements are removed from the codebase to increase legibility.

Alleviation

Issue has been resolved.

ERF-04 | Unchecked Return Value

Category	Severity	Location	Status
Volatile Code	Minor	ERC20FriendlyRewardModule.sol: 255, 277	Acknowledged

Description

The linked functions invocations do not check the return value of the function call which returns `uint256` in case of a proper call.

Recommendation

We would advise to check the return value of the function utilize it within the function.

Alleviation

Issue has been acknowledged by the client. Client's comment "This particular reward module does not have a need to check the return values mentioned. However, these values are generally useful and may be required in other reward module implementations."

ERF-05 | Function returns local variables explicitly.

Category	Severity	Location	Status
Gas Optimization	Informational	ERC20FriendlyRewardModule.sol: 268, 75, 91	Resolved

Description

Linked functions returns locally created variables explicitly. For gas optimization, named return variables are cheaper

Recommendation

We would suggest to utilize named return variables for gas optimizations. For example

```
function totals() public view override returns (uint256[] memory arr) {
        arr = new uint256[](1);
        arr[0] = _token.balanceOf(address(this));
}
```

Alleviation

Issue has been resolved.

ERI-01 | Function returns local variables explicitly.

Category	Severity	Location	Status
Gas Optimization	Informational	info/ERC20CompetitiveRewardInfo.sol: 124, 51	Resolved

Description

Linked functions returns locally created variables explicitly. For gas optimization, named return variables are cheaper.

Recommendation

We would suggest to utilize named return variables for gas optimizations. For example

```
function totals() public view override returns (uint256[] memory arr) {
        arr = new uint256[](1);
        arr[0] = _token.balanceOf(address(this));
}
```

Alleviation

Issue has been resolved.

ERR-01 | Unchecked Return Value

Category	Severity	Location	Status
Volatile Code	Minor	ERC20CompetitiveRewardModule.sol: 329, 301	Acknowledged

Description

The linked functions invocations do not check the return value of the function call which returns `uint256` in case of a proper call.

Recommendation

We would advise to check the return value of the function utilize it within the function.

Alleviation

Issue has been acknowledged by the client. Client's comment "This particular reward module does not have a need to check the return values mentioned. However, these values are generally useful and may be required in other reward module implementations."

ERR-02 | Function returns local variables explicitly.

Category	Severity	Location	Status
Gas Optimization, Language Specific	Informational	ERC20CompetitiveRewardModule.sol: 70, 79, 129, 259, 347	Resolved

Description

Linked functions returns locally created variables explicitly. For gas optimization, named return variables are cheaper.

Recommendation

We would suggest to utilize named return variables for gas optimizations. For example

```
function totals() public view override returns (uint256[] memory arr) {
        arr = new uint256[](1);
        arr[0] = _token.balanceOf(address(this));
}
```

Alleviation

Issue has been resolved.

ERS-01 | Function returns local variables explicitly.

Category	Severity	Location	Status
Gas Optimization, Language Specific	Informational	ERC20StakingModule.sol: 42, 51, 72, 175	Resolved

Description

Linked functions returns locally created variables explicitly. For gas optimization, named return variables are cheaper.

Recommendation

We would suggest to utilize named return variables for gas optimizations. For example

```
function totals() public view override returns (uint256[] memory arr) {
        arr = new uint256[](1);
        arr[0] = _token.balanceOf(address(this));
}
```

Alleviation

Issue has been resolved.

ERS-02 | Checks-effect-pattern not applied

Category	Severity	Location	Status
Volatile Code	Minor	ERC20StakingModule.sol: 115~133	Resolved

Description

State variables are changed after transfer function call.

Recommendation

It is recommended to follow checks-effects-interactions pattern for cases like this. It shields public functions from re-entrancy attacks. It's always a good practie to follow this pattern. `checks-effects-interaction` pattern also applies to ERC20 tokens as they can inform the recipient of a transfer in certain implementations.

Alleviation

Issue has been resolved.

GTO-01 | Centralization concern over total supply of tokens

Category	Severity	Location	Status
Centralization / Privilege	**Minor**	GeyserToken.sol: 21	**Acknowledged**

Description

Owner of a token contract gets all of the total supply. In case of lost access to the private key of an account or mishandling security of private keys, an attacker could benefit from that.

Recommendation

Total supply should be distributed to the users at start or be distributed to a handful of trusted addresses so no one address can hold all of the tokens at start. We would advise using multi-sig wallet for that.

Alleviation

Client's comment "This token has already been minted (v1 launch) and we will not be deploying this contract again. The entire supply was locked into a set of v1 pools for vesting and distribution."

OCO-01 | Lack of input validation

Category	Severity	Location	Status
Volatile Code	Informational	OwnerController.sol: 92~106	Resolved

Description

Linked functions lacks input validation on address parameters against 0x0 address. In case of making a mistake, controller or an owner could get set to the 0x0 address.

Recommendation

We would recommend adding require check against 0x0 address.

Alleviation

Issue has been resolved.

OCO-02 | Lack of ownership renouncement functions

Category	Severity	Location	Status
Control Flow	Informational	OwnerController.sol: 19	Acknowledged

Description

Contract is based off Ownable.sol from OpenZeppelin but it's lacking a function named `renounceOwnership`. Having this function helps to transition into self governed protocol instead of an owner having a lot of power over most crucial functions.

Recommendation

We would recommend to add renounceOwnership functions for the owner of the contract as well as for the controller contract.

Alleviation

Client's comment "These functions were intentionally removed to reduce contract size and reduce the gas cost of Pool deployment. In the case of our modular system, every component that inherits from the OwnerController contract will always need both an owner and controller to function properly."

OCO-03 | User-Defined Getters

Category	Severity	Location	Status
Gas Optimization	Informational	OwnerController.sol: 43~55	Acknowledged

Description

The linked variables contain user-defined getter functions that are equivalent to their name barring for an underscore (`_`) prefix / suffix.

Recommendation

We advise that the linked variables are instead declared as `public` and that they are renamed to their respective getter's name as compiler-generated getter functions are less prone to error and much more maintainable than manually written ones.

Alleviation

Client's comment "This was an intentional design decision for data encapsulation and safety. We do not want the inheriting contracts to have the ability to modify the _owner or _controller variables."

PFO-01 | Centralization concern

Category	Severity	Location	Status
Centralization / Privilege	**Minor**	PoolFactory.sol: 115~142	**Acknowledged**

Description

Owner has too much power over linked functions of a contract. In case of lost access to the private key of an account or mishandling security of private keys, an attacker could disturb operations and benefit from it.

Recommendation

Mentioned functions should be called by governance or be handled by multi-sig wallet.

Alleviation

Issue has been acknowledged by the team and will use multi-sig wallet. Client's comment: "As suggested, the PoolFactory will be owned and controlled by a multi-sig wallet. Additionally, it is worth noting that the privileged methods mentioned will be clearly highlighted and are relatively low risk/impact on the broader platform."

PFO-02 | Lack of input validation

Category	Severity	Location	Status
Volatile Code	Informational	PoolFactory.sol: 115~142	Resolved

Description

Linked functions lacks input validation on address parameters against 0x0 address. In case of making a mistake, controller or an owner could get set to the 0x0 address.

Recommendation

We would recommend adding require check against 0x0 address.

Alleviation

Issue has been resolved.

Appendix

Finding Categories

Centralization / Privilege

Centralization / Privilege findings refer to either feature logic or implementation of components that act against the nature of decentralization, such as explicit ownership or specialized access roles in combination with a mechanism to relocate funds.

Gas Optimization

Gas Optimization findings do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.

Control Flow

Control Flow findings concern the access control imposed on functions, such as owner-only functions being invoke-able by anyone under certain circumstances.

Volatile Code

Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.

Language Specific

Language Specific findings are issues that would only arise within Solidity, i.e. incorrect usage of private or delete.

Coding Style

Coding Style findings usually do not affect the generated byte-code but rather comment on how to make the codebase more legible and, as a result, easily maintainable.

Checksum Calculation Method

The "Checksum" field in the "Audit Scope" section is calculated as the SHA-256 (Secure Hash Algorithm 2 with digest size of 256 bits) digest of the content of each file hosted in the listed source repository under the specified commit.

The result is hexadecimal encoded and is the same as the output of the Linux "sha256sum" command against the target file.

Disclaimer

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to the Company in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes without CertiK's prior written consent.

This report is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts CertiK to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. CertiK's position is that each company and individual are responsible for their own due diligence and continuous security. CertiK's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

About

Founded in 2017 by leading academics in the field of Computer Science from both Yale and Columbia University, CertiK is a leading blockchain security company that serves to verify the security and correctness of smart contracts and blockchain-based protocols. Through the utilization of our world-class technical expertise, alongside our proprietary, innovative tech, we're able to support the success of our clients with best-in-class security, all whilst realizing our overarching vision; provable trust for all throughout all facets of blockchain.