Furucombo

Trevi

SMART CONTRACT AUDIT

14.09.2021

Made in Germany by Chainsulting.de

Table of contents

1. Disclaimer

The audit makes no statements or warrantees about utility of the code, safety of the code, suitability of the business model, investment advice, endorsement of the platform or its products, regulatory regime for the business model, or any other statements about fitness of the contracts to purpose, or their bug free status. The audit documentation is for discussion purposes only.

The information presented in this report is confidential and privileged. If you are reading this report, you agree to keep it confidential, not to copy, disclose or disseminate without the agreement of Furucombo by DINNGO Pte. Ltd. . If you are not the intended receptor of this document, remember that any disclosure, copying or dissemination of it is forbidden.

Major Versions / Date	Description
0.1 (06.07.2021)	Layout
0.2 (07.07.2021)	Test Deployment
0.5 (08.07.2021)	Manual & Automated Security Testing
0.6 (08.07.2021)	Testing SWC Checks
0.7 (09.07.2021)	Verify Claims
0.9 (09.07.2021)	Summary and Recommendation
1.0 (10.07.2021)	Final document
1.1 (20.07.2021)	Re-check
1.2 (27.07.2021)	Re-check (d31131dd2ac61f509b57813c29c4828ae27c1b12)
1.3 (01.08.2021)	Re-check (6d68386b4077a1e29386344369ab9b4bc7f8af89)
1.5 (09.09.2021)	Re-check (58aecaa5432b893386345706dc173f8282cffb69)
2.0 (14.09.2021)	Added deployed contract addresses

2. About the Project and Company

Company address:

DINNGO Pte. Ltd.
100 Tras Street #16-01
Singapore 079027

Website: https://furucombo.app

Twitter: https://twitter.com/furucombo

Medium: https://medium.com/furucombo

Telegram: https://t.me/furucombo

YouTube: https://www.youtube.com/channel/UCa1kGD4lvTSrmfKbDjQNOxQ

Discord: https://discord.furucombo.app

2.1 Project Overview

Trevi is an ERC20-based staking system. It enables users to stake their token and join different Angel's reward program without the necessity of moving their funds again, which can greatly improve the capital efficiency.

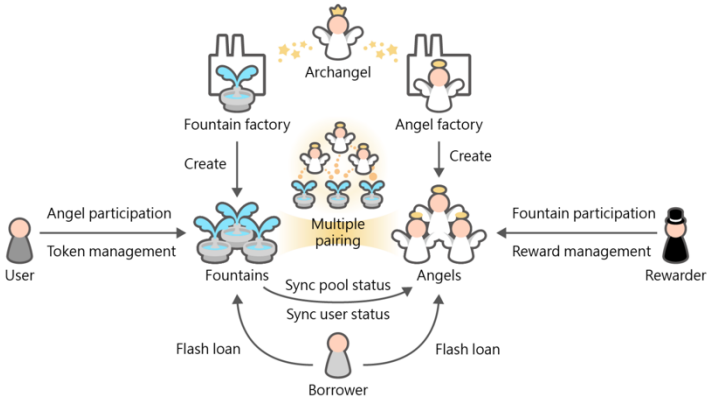
Anyone can create Fountain for a token through Fountain factory. Anyone can create Angel to run their own rewarding program through Angel factory.

The interaction between a user and Fountain can be divided into two groups

- Angel participation, users can have their own choice on which Angel to join. Joining multiple Angel is possible.
- Token management, user can
    - Deposit or withdraw their funds into Fountain to manage their assets to participate in the ongoing or future program from Angel.
    - Harvest the reward from the Angel they joined.

The interaction between a rewarder and Angel can be divided into two groups

- Fountain participation, rewarder can assign which token to be rewarded, which leads to different Fountain. Configuration is assigned separately, which means that different Fountain may have different reward allocation point.
- Reward management, rewarder can assign the rewarding amount and time interval to manage the distribution speed of reward.



3. Vulnerability & Risk Level

Risk represents the probability that a certain source-threat will exploit vulnerability, and the impact of that event on the organization or system. Risk Level is computed based on CVSS version 3.0.

Level	Value	Vulnerability	Risk (Required Action)
Critical	9 – 10	A vulnerability that can disrupt the contract functioning in a number of scenarios, or creates a risk that the contract may be broken.	Immediate action to reduce risk level.
High	7 – 8.9	A vulnerability that affects the desired outcome when using a contract, or provides the opportunity to use a contract in an unintended way.	Implementation of corrective actions as soon as possible.
Medium	4 – 6.9	A vulnerability that could affect the desired outcome of executing the contract in a specific scenario.	Implementation of corrective actions in a certain period.
Low	2 – 3.9	A vulnerability that does not have a significant impact on possible scenarios for the use of the contract and is probably subjective.	Implementation of certain corrective actions or accepting the risk.
Informational	0 – 1.9	A vulnerability that have informational character but is not effecting any of the code.	An observation that does not determine a level of risk

4. Auditing Strategy and Techniques Applied

Throughout the review process, care was taken to evaluate the repository for security-related issues, code quality, and adherence to specification and best practices. To do so, reviewed line-by-line by our team of expert pentesters and smart contract developers, documenting any issues as there were discovered.

4.1 Methodology

The auditing process follows a routine series of steps:

1. Code review that includes the following:
   i. Review of the specifications, sources, and instructions provided to Chainsulting to make sure we understand the size, scope, and functionality of the smart contract.
   ii. Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
   iii. Comparison to specification, which is the process of checking whether the code does what the specifications, sources, and instructions provided to Chainsulting describe.
2. Testing and automated analysis that includes the following:
   i. Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run those test cases.
   ii. Symbolic execution, which is analysing a program to determine what inputs causes each part of a program to execute.
3. Best practices review, which is a review of the smart contracts to improve efficiency, effectiveness, clarify, maintainability, security, and control based on the established industry and academic practices, recommendations, and research.
4. Specific, itemized, actionable recommendations to help you take steps to secure your smart contracts.

4.2 Used Code from other Frameworks/Smart Contracts (direct imports)

Dependency / Import Path	Source
./libraries/boringcrypto/…	https://github.com/boringcrypto/BoringSolidity/tree/master/contracts
@openzeppelin/contracts/access/Ownable.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v3.4.1-solc-0.7/contracts/access/Ownable.sol
@openzeppelin/contracts/cryptography/ECDSA.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v3.4.1-solc-0.7/contracts/cryptography/ECDSA.sol
@openzeppelin/contracts/drafts/EIP712.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v3.4.1-solc-0.7/contracts/drafts/EIP712.sol
@openzeppelin/contracts/drafts/IERC20Permit.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v3.4.1-solc-0.7/contracts/drafts/IERC20Permit.sol
@openzeppelin/contracts/math/Math.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v3.4.1-solc-0.7/contracts/math/Math.sol
@openzeppelin/contracts/math/SafeMath.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v3.4.1-solc-0.7/contracts/math/SafeMath.sol
@openzeppelin/contracts/token/ERC20/IERC20.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v3.4.1-solc-0.7/contracts/token/ERC20/IERC20.sol
@openzeppelin/contracts/token/ERC20/SafeERC20.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v3.4.1-solc-0.7/contracts/token/ERC20/SafeERC20.sol
@openzeppelin/contracts/utils/Context.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v3.4.1-solc-0.7/contracts/utils/Context.sol

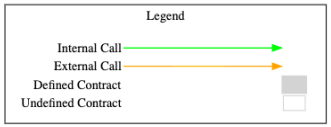
Dependency / Import Path	Source
@openzeppelin/contracts/utils/Counters.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v3.4.1-solc-0.7/contracts/utils/Counters.sol
@openzeppelin/contracts/utils/ReentrancyGuard.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v3.4.1-solc-0.7/contracts/utils/ReentrancyGuard.sol

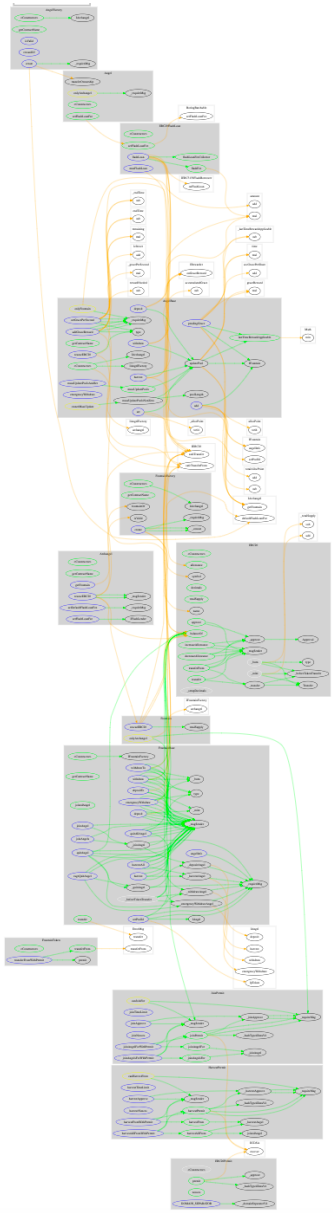
4.3 Tested Contract Files

The following are the MD5 hashes of the reviewed files. A file with a different MD5 hash has been modified, intentionally or otherwise, after the security review. You are cautioned that a different MD5 hash could be (but is not necessarily) an indication of a changed condition or potential vulnerability that was not within the scope of the review

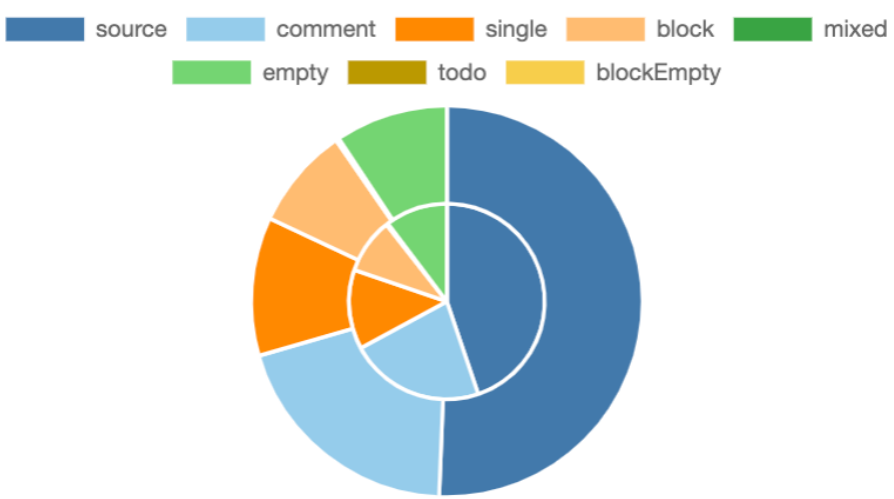
File	Fingerprint (MD5)
./interfaces/IAngelFactory.sol	80d2431162453133d88a41d098273bdc
./interfaces/IFlashLender.sol	9854e006ba499e1aaec163f9e23af0dd
./interfaces/IFountain.sol	37e1e49313db0cb90d2ac12344111a84
./interfaces/IArchangel.sol	f040929bedc4a960336ee1e42b435c75
./interfaces/IMasterChef.sol	e9ccc95296f872ab0581ad960b4de529
./interfaces/IAngel.sol	eb8899459985b7a3c838e7ec6d5146f1
./interfaces/IRewarder.sol	2080295a4b3cc19b35f68aabbe2e980f
./interfaces/IFountainFactory.sol	48ac64f94a5cfadd79685078d19e1bc8
AngelFactory.sol	62236def967a7c9e49c18e9dbc9bfa4a
JoinPermit.sol	47c4436f252139597c85db010d7665e0
HarvestPermit.sol	4879b853dd3bab4b9a73dc3bb20b76a5
FountainToken.sol	26c199889eca9e5891b241267c154130
ERC20FlashLoan.sol	4511d0dcbd1d88fbd83c6c270bacb2bf
ERC20.sol	a7938c7bb9d958898cdd6fa1dc78c14b
ERC20Permit.sol	a8384f4f34dc209d95ef8303c4eccf00
Angel.sol	9c93f27cd7d8e0dbba62e79dff6eed10
./utils/ErrorMsg.sol	4efe51f214d74d873985816ad27c7c88
FountainBase.sol	e624d22530c8d0e840ff6b1b3ae0c50d
AngelBase.sol	5a15a02771aabe0af0b209508842f676
Fountain.sol	d1120e5fc125bdf71752b10fe4917ef5
FountainFactory.sol	2829654f1d31d0e576810e137e73cdfd
Archangel.sol	ab31fd03a07430d0e6ac8ba215bf1da8

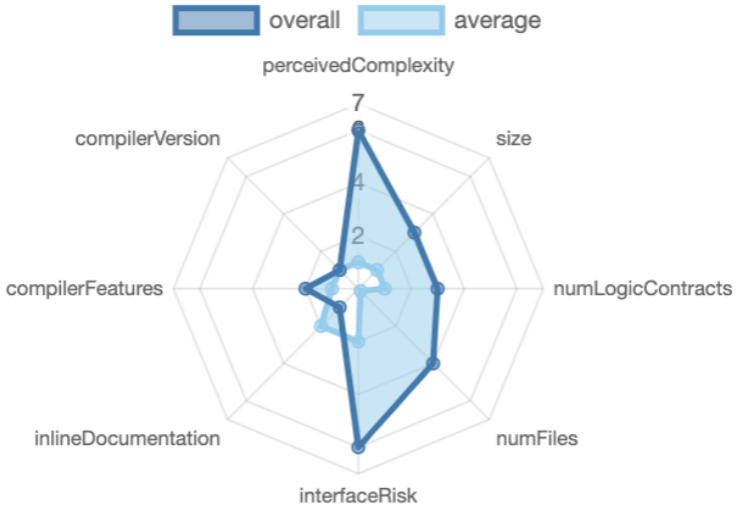
4.4 Metrics / CallGraph





4.5 Metrics / Source Lines & Risk





4.6 Metrics / Capabilities

Solidity Versions observed	🧪 Experimental Features	💰 Can Receive Funds	🖥 Uses Assembly	💣 Has Destroyable Contracts

`0.6.12`
`>=0.6.0 <0.8.0`
`>=0.6.5 <0.8.0`
	`ABIEncoderV2`	`yes`	`yes`
			(2 asm blocks)	

📪 Transfers ETH	⚡ Low-Level Calls	👥 DelegateCall	🧱 Uses Hash Functions	🪚 ECRecover	🌀 New/Create/Create2
`yes`		`yes`	`yes`		`yes` → `NewContract:Angel` → `AssemblyCall:Name:create` → `NewContract:Fountain` → `NewContract:AngelFactory` → `NewContract:FountainFactory`

🌐Public	💰Payable
178	3

External	Internal	Private	Pure	View
123	182	0	33	58

Total	🌐Public
48	25

4.7 Metrics / Source Unites in Scope

Type	File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score	Capabilities
🔍	contracts/interfaces/IAngelFactory.sol		1	12	7	3	2	9	
🔍	contracts/interfaces/IFlashLender.sol		1	10	8	4	1	7	
🔍	contracts/interfaces/IFountain.sol		1	105	11	6	2	75	🧪
🔍	contracts/interfaces/IArchangel.sol		1	14	7	3	2	13	
🔍	contracts/interfaces/IMasterChef.sol		1	28	20	16	7	7	🧪
🔍	contracts/interfaces/IAngel.sol		1	26	17	13	1	19	🧪
🔍	contracts/interfaces/IRewarder.sol		1	22	9	5	1	5	
🔍	contracts/interfaces/IFountainFactory.sol		1	12	7	3	2	9	
📝	contracts/AngelFactory.sol	1		55	55	33	11	34	🧪🌀
🎨	contracts/JoinPermit.sol	1		201	158	88	49	61	🧱
🎨	contracts/HarvestPermit.sol	1		205	161	91	49	63	🧱

Type	File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score	Capabilities
📝	contracts/FountainToken.sol	1		30	22	16	1	11	🧪
📝	contracts/ERC20FlashLoan.sol	1		116	95	61	22	51	🧱
📝	contracts/Angel.sol	1		37	37	25	6	18	🧪
🎨	contracts/utils/ErrorMsg.sol	1		31	21	15	1	9	
🎨	contracts/FountainBase.sol	1		364	339	214	76	232	🧪📮
📝	contracts/AngelBase.sol	1		556	516	340	123	211	🧪
📝	contracts/Fountain.sol	1		66	62	45	10	32	🧪
📝	contracts/ERC20.sol	1		379	333	114	186	83	
📝	contracts/FountainFactory.sol	1		64	60	37	13	40	🌀
📝	contracts/Archangel.sol	1		76	72	43	18	56	🌀
🎨	contracts/ERC20Permit.sol	1		90	82	40	28	29	🧱
📝🔍 🎨	**Totals**	**14**	**8**	**2499**	**2099**	**1215**	**611**	**1074**	🧪📮🧱🌀

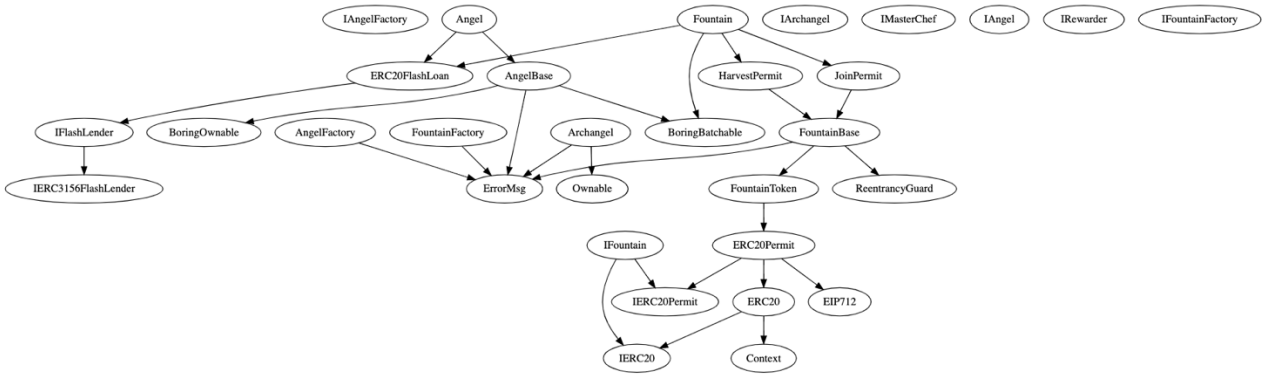
5. Scope of Work

The Furucombo Team provided us with the files that needs to be tested. The scope of the audit are the Trevi contracts.

The team put forward the following assumptions regarding the security, usage of the contracts:

- The smart contract is coded according to the newest standards and in a secure way
- Currently the flashLoan does not involve any reentrance guard since we haven't think of any possible vulnerability. However, if there is no actual scenario for using our functions among flashLoan process, would you suggest having it being guarded with the functions in Fountain / Angel?
- Aave applied a rounding solution (https://github.com/aave/protocol-v2/blob/master/contracts/protocol/libraries/math/PercentageMath.sol) when they deal with the percentage calculation. What do you think about this feature, is it for avoiding attacks or just a feature.
- Compare source with https://github.com/sushiswap/sushiswap/tree/canary/contracts MiniChefV2 and point out the diffs and check them twice.
- List all audits from SushiSwap and check if the findings effecting Trevi and if they are fixed within there source.

The main goal of this audit was to verify these claims. The auditors can provide additional feedback on the code upon the client's request.



5.1 Manual and Automated Vulnerability Test

CRITICAL ISSUES
During the audit, Chainsulting's experts found **no Critical issues** in the code of the smart contract.

HIGH ISSUES
During the audit, Chainsulting's experts found **no High issues** in the code of the smart contract.

MEDIUM ISSUES

5.1.1 add() does not prevent the same LP token from being added more than once
Severity: MEDIUM
Status: False positive. Test added.
File(s) affected: AngelBase.sol

Attack / Description	Code Snippet	Result/Recommendation

On L133 it is noted: "// XXX DO NOT add the same LP token more than once. Rewards will be messed up if you do." If a token is mistakenly added more than once, it would reset the rewards variables associated with the token (e.g., accSushiPerShare).	Line: 137 - 162
	
	`function add(`
	`uint256 allocPoint,`
	`IERC20 _lpToken,`
	`IRewarder _rewarder`
	`) public onlyOwner {`
	`uint256 pid = lpToken.length;`
	
	`totalAllocPoint = totalAllocPoint.add(allocPoint);`
	`lpToken.push(_lpToken);`
	`rewarder.push(_rewarder);`
		This could be prevented by creating a mapping from addresses to booleans, such that LP tokens get mapped to true once they've been added. The function could then have a require-statement preventing the same LP token from being added twice.

```
        poolInfo.push(
            PoolInfo({
                allocPoint: allocPoint.to64(),
                lastRewardTime: block.timestamp.to64(),
                accSushiPerShare: 0
            })
        );
        emit LogPoolAddition(pid, allocPoint, _lpToken,
_rewarder);

        ////////////////////////// New
        // Update pid in fountain
        IFountain fountain =

IFountain(archangel.getFountain(address(_lpToken)));
        fountain.setPoolId(pid);
    }
```

5.1.2 State variables written after external call
Severity: MEDIUM
Status: Protected through nonReentrant modifier. / Fixed
File(s) affected: FountainBase.sol, AngelFactory.sol

Attack / Description	Code Snippet	Result/Recommendation

In the current implementation, state variables are written after the external calls. External call can potentially execute malicious code. In this case, the function calls are to trusted contracts.	FountainBase.sol Line 289 -290:
	`angel.deposit(info.pid, amount, user);`
	`info.totalBalance = info.totalBalance.add(amount);`
	
	FountainBase.sol Line 304 - 305:
	`angel.withdraw(info.pid, amount, user);`
	`info.totalBalance = info.totalBalance.sub(amount);`
		We highly recommend writing to state variables before using external calls to avoid reentrancy attacks.
		
		See SWC-107:
		https://swcregistry.io/docs/SWC-107

	FountainBase.sol Line 333 - 334:
	`angel.emergencyWithdraw(info.pid, user);`
	`info.totalBalance = info.totalBalance.sub(amount);`
	
	`AngelFactory.sol Line 53 – 54:`
	`newAngel.transferOwnership(msg.sender, true, false);`
	`_rewards[newAngel] = reward;`	

5.1.3 Shadowing state variables
Severity: MEDIUM
Status: False positive. No-op.
File(s) affected: JoinPermit.sol, Harvest.sol

Attack / Description	Code Snippet	Result/Recommendation
In the current implementation, state variables in the deriving contract are shadowed. Unlike functions, state variables cannot be overridden by re-declaring it in the child contract.	JoinPermit.sol Line 12: `mapping(address => Counters.Counter) private _nonces;` HarvestPermit.sol Line 12: `mapping(address => Counters.Counter) private _nonces;`	We highly recommend removing the shadowed state variables to avoid unintended behaviour. The *_nonces* state variable is already defined in ERC20Permit. See SWC-119: https://swcregistry.io/docs/SWC-119

LOW ISSUES

5.1.4 Wrong import of OpenZeppelin library
Severity: LOW
Status: Fixed
File(s) affected: All

Attack / Description	Code Snippet	Result/Recommendation
In the current implementation, some OpenZeppelin files are part of the repository. This violates OpenZeppelin's MIT license, which requires the license and copyright notice to be included if its code is used. Moreover, updating code manually is error-prone.	`Address, Context, Counters, Create2, ECDSA, EIP712, ERC20, ERC20Permit, Ownable, Pausable, ReentrancyGuard, SafeERC20, SafeMath, SignedSafeMath, IERC20, IERC20Permit, IERC3156`	We highly recommend using npm (import "@openzeppelin/contracts/..) in order to guarantee that original OpenZeppelin contracts are used with no modifications. This also allows for any bug-fixes to be easily integrated into the codebase. https://www.npmjs.com/package/@openzeppelin/contracts/v/3.4.1

5.1.5 Shadowing local variables
Severity: LOW
Status: Fixed
File(s) affected: Angel.sol, Fountain.sol

Attack / Description	Code Snippet	Result/Recommendation

In the current implementation, local variables are shadowing inherited state variables.	Angel.sol Line 20:
	`constructor(IERC20 token, uint256 flashLoanFee)`
	
	Fountain.sol Line 31:
	`constructor(…, uint256 flashLoanFee)`
		We highly recommend renaming the shadowing variables in the constructors to avoid unintended behaviour.

INFORMATIONAL ISSUES

5.1.6 Rename Sushi branding
Severity: INFORMATIONAL
Status: Fixed
File(s) affected: AngelBase.sol, IAngel.sol, IMasterChef.sol, IRewarder.sol, RewarderMock.sol

Attack / Description	Code Snippet	Result/Recommendation
There are still branding from SushiSwap included, which would lead to confusion.	Line 39 `uint128 accSushiPerShare;` and more …	It is recommended to rename functions / variables / comments and keep the branding consistent to Fountain, Angel, Archangel.

5.1.7 Public functions could be external

Severity: INFORMATIONAL

Status: Fixed

File(s) affected: ERC20FlashLoan.sol, JoinPermit.sol, HarvestPermit.sol, FountainToken.sol, FountainBase.sol, AngelBase.sol

Attack / Description	Code Snippet	Result/Recommendation

In the current implementation several functions are declared as public where they could be external. For public functions Solidity immediately copies array arguments to memory, while external functions can read directly from calldata. Because memory allocation is expensive, the gas consumption of public functions is higher.	ERC20FlashLoan .sol Line 65:
	`function flashLoan(…) public{`
	
	JoinPermit.sol Line 45 & 109:
	`function joinTimeLimit(…) public{`
	`function joinNonces(…) public{`
	
	HarvestPermit.sol Line 45 & 109:
	`function harvestTimeLimit (…) public{`
	`function harvestNonces(…) public{`
	
	FountainToken.sol Line 17:
	`function transferFromWithPermit (…) public{`
	
	FountainBase.sol Line 75:
	`function angelInfo (…) public{`
	
	AngelBase.sol Line 128, 137, 169, 192, 275, 307, 344, 391
		We recommend declaring functions as external if they are not used internally. This leads to lower gas consumption and better code readability.

5.1.8 State variables should be constant
Severity: INFORMATIONAL
Status: Fixed
File(s) affected: JoinPermit.sol, HarvestPermit.sol

Attack / Description	Code Snippet	Result/Recommendation

In the current implementation two state variables are declared as immutable, where they should be constant. State variables can be declared as constant or immutable. In both cases, the variables cannot be modified after the contract has been constructed. For constant variables, the value has to be fixed at compile-time, while for immutable, it can still be assigned at construction time.	JoinPermit.sol Line 832 - 838:
	`bytes32 private immutable _JOIN_PERMIT_TYPEHASH = keccak256( "JoinPermit(address user,address sender,uint256 timeLimit, uint256 nonce,uint256 deadline)" );`
	
	HarvestPermit.sol Line 863 - 869:
	`bytes32 private immutable _HARVEST_PERMIT_TYPEHASH = keccak256( "JoinPermit(address user,address sender,uint256 timeLimit, uint256 nonce,uint256 deadline)" );`
		We recommend declaring state variables as constant if they are not set in the constructor.

5.1.9 Checking for Boolean equality
Severity: INFORMATIONAL
Status: Fixed
File(s) affected: FountainBase.sol

Attack / Description	Code Snippet	Result/Recommendation
In the current implementation one require check uses a comparison to a Boolean constant. This leads to unnecessary gas consumption.	FountainBase.sol Line 88: `_requireMsg(info.isSet == false, "setPoolId", "Fountain: angel is set");`	It is recommended to remove the equality check to the Boolean constant and use the value itself.

Other modifications (20.07.2021 / Re-check)

- Timely massUpdatePool During Pool Updates
  - Added when calling `set()`, `add()`, `addGraceReward()`, `setGracePerSecond()`.
- Sanity Checks For System/Function Parameters
  - Add sanity check for `gracePerSecond`
- Rounding half up in fee calculation
- Apply try/catch for the function call to `_rewarder.onSushiReward` in `emergencyWithdraw`
- Apply endTime to avoid miscalculation caused by insufficient reward in Angel

Update (01.08.2021 / Re-check)

Commit: https://github.com/dinngodev/trevi/commit/6d68386b4077a1e29386344369ab9b4bc7f8af89

The commit was addressing the incident from PolyYeld (28.07.2021 / Source: https://twitter.com/peckshield/status/1420274048496328705) which was affecting the Fountain contract on some degree.

The Trevi development team correctly added the fix and verified the case with a deflationary token unit test.

Update (09.09.2021 / Re-check)

Commit: https://github.com/dinngodev/trevi/commit/58aecaa5432b893386345706dc173f8282cffb69

The commit added an event for the try/catch in emergencyWithdraw to record the info when the rewarder fails to execute.
We also checked the issues by different audits (https://github.com/dinngodev/trevi/tree/master/audit) and the latest commits.

5.2. SWC Attacks

ID	Title	Relationships	Test Result
SWC-131	Presence of unused variables	CWE-1164: Irrelevant Code	✅
SWC-130	Right-To-Left-Override control character (U+202E)	CWE-451: User Interface (UI) Misrepresentation of Critical Information	✅
SWC-129	Typographical Error	CWE-480: Use of Incorrect Operator	✅
SWC-128	DoS With Block Gas Limit	CWE-400: Uncontrolled Resource Consumption	✅
SWC-127	Arbitrary Jump with Function Type Variable	CWE-695: Use of Low-Level Functionality	✅
SWC-125	Incorrect Inheritance Order	CWE-696: Incorrect Behavior Order	✅
SWC-124	Write to Arbitrary Storage Location	CWE-123: Write-what-where Condition	✅
SWC-123	Requirement Violation	CWE-573: Improper Following of Specification by Caller	✅

ID	Title	Relationships	Test Result
SWC-122	Lack of Proper Signature Verification	CWE-345: Insufficient Verification of Data Authenticity	✅
SWC-121	Missing Protection against Signature Replay Attacks	CWE-347: Improper Verification of Cryptographic Signature	✅
SWC-120	Weak Sources of Randomness from Chain Attributes	CWE-330: Use of Insufficiently Random Values	✅
SWC-119	Shadowing State Variables	CWE-710: Improper Adherence to Coding Standards	✅
SWC-118	Incorrect Constructor Name	CWE-665: Improper Initialization	✅
SWC-117	Signature Malleability	CWE-347: Improper Verification of Cryptographic Signature	✅
SWC-116	Timestamp Dependence	CWE-829: Inclusion of Functionality from Untrusted Control Sphere	✅
SWC-115	Authorization through tx.origin	CWE-477: Use of Obsolete Function	✅
SWC-114	Transaction Order Dependence	CWE-362: Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')	✅

ID	Title	Relationships	Test Result
SWC-113	DoS with Failed Call	CWE-703: Improper Check or Handling of Exceptional Conditions	✅
SWC-112	Delegatecall to Untrusted Callee	CWE-829: Inclusion of Functionality from Untrusted Control Sphere	✅
SWC-111	Use of Deprecated Solidity Functions	CWE-477: Use of Obsolete Function	✅
SWC-110	Assert Violation	CWE-670: Always-Incorrect Control Flow Implementation	✅
SWC-109	Uninitialized Storage Pointer	CWE-824: Access of Uninitialized Pointer	✅
SWC-108	State Variable Default Visibility	CWE-710: Improper Adherence to Coding Standards	✅
SWC-107	Reentrancy	CWE-841: Improper Enforcement of Behavioral Workflow	✅
SWC-106	Unprotected SELFDESTRUCT Instruction	CWE-284: Improper Access Control	✅
SWC-105	Unprotected Ether Withdrawal	CWE-284: Improper Access Control	✅
SWC-104	Unchecked Call Return Value	CWE-252: Unchecked Return Value	✅

ID	Title	Relationships	Test Result
SWC-103	Floating Pragma	CWE-664: Improper Control of a Resource Through its Lifetime	✅
SWC-102	Outdated Compiler Version	CWE-937: Using Components with Known Vulnerabilities	✅
SWC-101	Integer Overflow and Underflow	CWE-682: Incorrect Calculation	✅
SWC-100	Function Default Visibility	CWE-710: Improper Adherence to Coding Standards	✅

5.3. Associated audits with the forked codebase

Quantstamp	https://github.com/quantstamp/sushiswap-security-review	Sep. 2020
Peckshield	https://github.com/peckshield/publications/blob/master/audit_reports/PeckShield-Audit-Report-SushiSwap-v1.0.pdf	Sep. 2020

5.4. Verify Claims

5.4.1 The smart contract is coded according to the newest standards and in a secure way
Status: tested and verified ✅

5.4.2 Currently the flashLoan does not involve any reentrance guard since we haven't think of any possible vulnerability. However, if there is no actual scenario for using our functions among flashLoan process, would you suggest having it being guarded with the functions in Fountain / Angel?
Status:

We have not been able to identify any possible reentrance scenarios.

5.4.3 Aave applied a rounding solution (https://github.com/aave/protocol-v2/blob/master/contracts/protocol/libraries/math/PercentageMath.sol) when they deal with the percentage calculation. What do you think about this feature, is it for avoiding attacks or just a feature.
Status:

We don't see it nescarry for the particular contracts to be implemented.

5.4.4 Compare source with https://github.com/sushiswap/sushiswap/tree/canary/contracts MiniChefV2 and point out the diffs and check them twice.
Status:

MiniChefV2: https://github.com/sushiswap/sushiswap/tree/canary/contracts

Differences to MiniChefV2:

- Actions regarding staking tokens are located at fountain.
- Migrate function has been removed
- harvestAndWithdraw function has been removed
- Deposit, Withdraw and EmergencyWithdraw functions can only be called from the fountain contract

5.4.5 List all audits from SushiSwap and check if the findings effecting Trevi and if they are fixed within there source.

Status: tested and verified ✅

6. Executive Summary

Two (2) independent Chainsulting experts performed an unbiased and isolated audit of the smart contract codebase. The final debrief took place on the July 10, 2021. The overall code quality of the project is good and the modifications of the forked SushiSwap MasterChef contracts did not increased the attack surface. It correctly implemented widely-used and reviewed contracts from OpenZeppelin, but we would recommend to do the import correctly.

The main goal of the audit was to verify the claims regarding the security of the smart contract and the functions. During the audit, no critical issues were found after the manual and automated security testing and the claims been successfully verified.

Update (27.07.2021): The codebase has been re-checked twice by the audit team from Chainsulting. The trevi development team was very responsive, fixed all issues and improved the project quality significantly.

Update (01.08.2021): The codebase has been re-checked by the audit team from Chainsulting. The trevi development team successfully added the fix from a known incident by PolyYeld, which was using a similar contract (MasterChef).

Update (09.09.2021): The codebase has been re-checked by the audit team from Chainsulting and approved the latest commits.

7. Deployed Smart Contract

VERIFIED

Archangel: https://polygonscan.com/address/0xf40388b593efb236d1AB314A6aa969F9487890d8#code
AngelFactory: https://polygonscan.com/address/0x66Ab9f76e7822B7160E22f8b02Dbd2D757FabF32#code
FountainFactory: https://polygonscan.com/address/0xDE7DBC03c90b0C6029F435865Cd92212D0e0cAc3#code