CERTIK

Security Assessment

Gelato-UNI

Jul 22nd, 2021

Table of Contents

Summary

This report has been prepared for Gelato Digital to discover issues and vulnerabilities in the source code of the Gelato-UNI project as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Static Analysis and 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 critical 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;
- Add enough unit tests to cover the possible use cases given they are currently missing in the repository;
- Provide more comments per each function for readability, especially contracts that are verified in public;
- Provide more transparency on privileged activities once the protocol is live.

Overview

Project Summary

Project Name	Gelato-UNI
Platform	Ethereum
Language	Solidity
Codebase	https://github.com/gelatodigital/g-uni-v1-core
Commit	0faae15cb5dfdadf413a7908232b1be209507ee9 f6250c9b2a520b166b68027df4b624709604b082 56a8e5403ba1823db0f108d4e85ad5d26699c0bb

Audit Summary

Delivery Date	Jul 22, 2021
Audit Methodology	Static Analysis, Manual Review
Key Components	

Vulnerability Summary

Vulnerability Level	Total	Pending	Partially Resolved	Resolved	Acknowledged	Declined
● Critical	0	0	0	0	0	0
● Major	0	0	0	0	0	0
● Medium	1	0	0	1	0	0
● Minor	6	0	0	1	5	0
● Informational	3	0	0	2	1	0
● Discussion	0	0	0	0	0	0

Audit Scope

ID	file	SHA256 Checksum
GUF	abstract/GUniFactoryStorage.sol	1ce6c23bee94d7be5c3b8c4eff75a7e9814156fe2dcb20dbb593d304a4ea74c2
GUP	abstract/GUniPoolStorage.sol	8debf96c229761d6b83f1157dad4921555848c5171c6cb983181191b17614138
GGG	abstract/Gelatofied.sol	1b43153cd0e9708f2b0ee624d627da29d1c408154d2be04538dca816b6468986
OUG	abstract/OwnableUninitialized.sol	fc6cb485ffc180d0016b474601d14a01b234db2e374782d174ad8ef1d772516f
IEI	interfaces/IEIP173Proxy.sol	dc21f6adb23c972676329e8db1ece154cc6af9d7e0c3e2419dbb29b823f5f10c
IGU	interfaces/IGUniFactory.sol	307ec7ba000ddd158a6cff5823f5f90a6e6ac2b62acfb6838aa949158bc93c95
IGP	interfaces/IGUniPoolStorage.sol	6f8442c3cb0f6288e36e443b15d0bf330556a03a71c998c9d720842e958578f5
IUV	interfaces/IUniswapV3Factory.sol	8f3f11cacbe3cb4b5ac8e193504f098a089979f0662083906b0cf31e371b0a74
EIP	vendor/proxy/EIP173Proxy.sol	0c25fc7fc1ac037b8c282b07faabe603f431b05c31e69b501b8f7fa0da6edb5c
EIW	vendor/proxy/EIP173ProxyWithReceive.sol	993ceb7c82e51b7592ad24b8626aa6b2b0cb35eb0ce4da9c6aab3008d79edd20
PGG	vendor/proxy/Proxied.sol	839aa562acde3a189d1d112a98e85d1580c9495d794b52bfb99d5da2d3393604
PGC	vendor/proxy/Proxy.sol	d6bd8b23d0d2a12be5baf48fb1fd394a3034449d9b0d2f2ac230a45061ee64b6
FMG	vendor/uniswap/FullMath.sol	b9599739d2d78cc85f70bdd424acf880df74afca09b16a5101cd3a7148722002
LAG	vendor/uniswap/LiquidityAmounts.sol	686c7c721ccb9836893c8d44c866610d0f60a92de446487ab38c4a3e5d303d8f
TMG	vendor/uniswap/TickMath.sol	46f37028f974ebbd848b52729c30c5328aa21d5c61e2ed0058fad96dcd13c11b
GUG	GUniFactory.sol	90454468cd9eb5954ae0be618b0008bca9b6360f7d350356ca33ff7fa396cdb1
GUC	GUniPool.sol	e0bb57a74821910337b91ba0806998389f20c2a33a29c7ec06645def316c2af0

There are a few depending injection contracts or addresses in the current project:

`factory`, `IERC20`, `tokenA` and `tokenB` for contract `GUniFactory` and `GUniPool`;

We assume these contracts or addresses are valid and non-vulnerable actors and implementing proper logic to collaborate with the current project.

To set up the project correctly, improve overall project quality and preserve upgradability, the following roles, are adopted in the codebase:

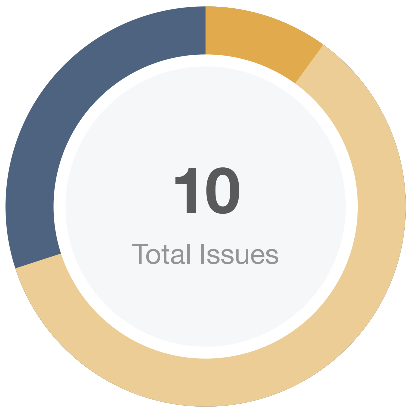
`onlyManager`, is adopted to manage pools and update configurations in contract `GUniFactory`;

`onlyManager`, is adopted to rebalance in contract and update configurations `GUniPool`;

`gelatofy`, is adopted to withdraw fees in contract `GUniFactory`;

To improve the trustworthiness of the project, dynamic runtime updates in the project should be notified to the community. Any plan to invoke the aforementioned functions should be also considered to move to the execution queue of `Timelock` contract.

Findings



🟥	**Critical**	**0** (0.00%)
🟧	**Major**	**0** (0.00%)
🟨	**Medium**	**1** (10.00%)
🟨	**Minor**	**6** (60.00%)
🟦	**Informational**	**3** (30.00%)
🟩	**Discussion**	**0** (0.00%)

ID	Title	Category	Severity	Status
GUC-01	Lack of Error Message	Coding Style	● Informational	✓ Resolved
GUC-02	Lack of Return Value Handling	Logical Issue	● Informational	ⓘ Acknowledged
GUC-03	Centralization Risk	**Centralization / Privilege**	● **Minor**	ⓘ **Acknowledged**
GUC-04	Uncollected Fee While Withdrawing Tokens	Logical Issue	● Minor	✓ Resolved
GUF-01	Centralization Risk	**Centralization / Privilege**	● **Minor**	ⓘ **Acknowledged**
GUG-01	Centralization Risk	**Centralization / Privilege**	● **Minor**	ⓘ **Acknowledged**
GUP-01	Lack of Specified Fee Range Restriction	**Centralization / Privilege**	● **Minor**	ⓘ **Acknowledged**
GUP-02	Lack of Event Emissions for Significant Transaction	Logical Issue	● Informational	✓ Resolved
GUP-03	Centralization Risk	**Centralization / Privilege**	● **Minor**	ⓘ **Acknowledged**
LAG-01	Incorrect Comparison	Volatile Code	● Medium	✓ Resolved

GUC-01 | Lack of Error Message

Category	Severity	Location	Status
Coding Style	● Informational	GUniPool.sol: 59, 71, 120, 177	✓ Resolved

Description

The `require` statements at the aforementioned lines do not have proper error messages implemented. Proper error messages in the following `require` check can indicate the desired operation failure to users or relay essential warnings:

```
    require(msg.sender == address(pool))
```

Recommendation

We recommend adding proper error message strings for the aforementioned `require` statements.

Alleviation

The client heeded our advice and resolved the issues by adding proper error messages for the `require` statements at the aforementioned lines. The changes are reflected in the commit `56a8e5403ba1823db0f108d4e85ad5d26699c0bb`.

GUC-02 | Lack of Return Value Handling

Category	Severity	Location	Status
Logical Issue	● Informational	GUniPool.sol: 145, 561~567, 600~606, 688~694	ⓘ Acknowledged

Description

Functions `IUniswapV3Pool.mint()` and `IUniswapV3Pool.collect()` are not void-returning functions. Ignoring their return values, especially when the return value represents the execution result, might cause unexpected exceptions.

Recommendation

We recommend handling return values of functions `IUniswapV3Pool.mint()` and `IUniswapV3Pool.collect()` at the aforementioned lines before continuing processing.

Alleviation

N/A

GUC-03 | Centralization Risk

Category	Severity	Location	Status
Centralization / Privilege	● Minor	GUniPool.sol: 235	ⓘ Acknowledged

Description

The `manager` role of the contract `GUniPool` has the following privilege by calling the listed function:

1. `GUniPool.executiveRebalance()`: Calculate the new concentration liquidity range and rebalance the liquidity. In addition, the liquidity concentration range is manually inputted by the `manager` role. As the calculation of the new concentration range is not reflected in the codebase, we have to address that any incorrect input would lead to severe loss in capital.

Any compromise to the `manager` account may allow the hacker to manipulate the project through these functions.

Recommendation

We recommend carefully managing the `manager` account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol to be improved via a decentralized mechanism or via smart-contract-based accounts with enhanced security practices, e.g. Multisignature wallets.

Indicatively, here are some feasible solutions that would also mitigate the potential risk:

- Time-lock with reasonable latency, e.g. 48 hours, for awareness on privileged operations;
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key;
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.

Alleviation

N/A

GUC-04 | Uncollected Fee While Withdrawing Tokens

Category	Severity	Location	Status
Logical Issue	● Minor	GUniPool.sol: 579	✓ Resolved

Description

According to the code behavior of function `GUniPool._withdrawExact()`, an `adminFee` should be deducted while withdrawing tokens. The following code snippet deducts the `adminFee`:

```
594    (fee0, fee1) = _subtractAdminFees(fee0, fee1);
```

Since there is no fee collection code found in the current codebase, the `adminFee` will remain in the contract. We do not believe that the current logic is desired and the fee should be collected during token withdrawal.

Recommendation

We recommend collecting fees in the function `GUniPool._withdrawExact()`.

Alleviation

The client heeded our advice and resolve the issue by implementing a new `_withdraw` function to replace the `_withdrawExact()` function. The change is reflected in the commit `f6250c9b2a520b166b68027df4b624709604b082`.

GUF-01 | Centralization Risk

Category	Severity	Location	Status
Centralization / Privilege	● **Minor**	abstract/GUniFactoryStorage.sol: 54, 62	ⓘ **Acknowledged**

Description

The manager of the contract `GUniFactoryStorage` has the following privileges by calling the listed functions:

1. `GUniFactoryStorage.setPoolImplementation()`: Set the pool implementation;
2. `GUniFactoryStorage.setGelatoDeployer()`: Set the address of `gelatoDeployer`.

Any compromise to the `manager` account may allow the hacker to manipulate the project through these functions.

Recommendation

We recommend carefully managing the `manager` account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol to be improved via a decentralized mechanism or via smart-contract-based accounts with enhanced security practices, e.g. Multisignature wallets.

Indicatively, here are some feasible solutions that would also mitigate the potential risk:

- Time-lock with reasonable latency, e.g. 48 hours, for awareness on privileged operations;
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key;
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.

Alleviation

N/A

GUG-01 | Centralization Risk

Category	Severity	Location	Status
Centralization / Privilege	● Minor	GUniFactory.sol: 84, 90, 103	ⓘ Acknowledged

Description

The manager of the contract `GUniFactory` has the following privileges by calling the listed functions:

1. `GUniFactory.upgradePools()`: Upgrade the `poolImplementation` for every pool;
2. `GUniFactory.upgradePoolsAndCall()`: Upgrade the address of `poolImplementation` and invoke pool functions;
3. `GUniFactory.makePoolsImmutable()`: Transfer the ownership of certain pools to zero address to make these pools immutable.

Any compromise to the `manager` account may allow the hacker to manipulate the project through these functions.

Recommendation

We recommend carefully managing the `manager` account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol to be improved via a decentralized mechanism or via smart-contract-based accounts with enhanced security practices, e.g. Multisignature wallets.

Indicatively, here are some feasible solutions that would also mitigate the potential risk:

- Time-lock with reasonable latency, e.g. 48 hours, for awareness on privileged operations;
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key;
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.

Alleviation

N/A

GUP-01 | Lack of Specified Fee Range Restriction

Category	Severity	Location	Status
Centralization / Privilege	● **Minor**	abstract/GUniPoolStorage.sol: 90, 129~131, 152~154	ⓘ **Acknowledged**

Description

The role `manager` can set the following state variables in functions `GUniPoolStorage.initialize()`, `GUniPoolStorage.updateGelatoParams()` and `GUniPoolStorage.initializeManagerFee()` up to 100% of the current pool balance:

- `managerFeeBPS`
- `gelatoWithdrawBPS`
- `gelatoRebalanceBPS`
- `gelatoSlippageBPS`
- `gelatoSlippageInterval`

Recommendation

We recommend setting specified ranges and check the following input variables in functions `GUniPoolStorage.initialize()`, `GUniPoolStorage.updateGelatoParams()` and `GUniPoolStorage.initializeManagerFee()`:

- `_managerFeeBPS`
- `newWithdrawBPS`
- `newRebalanceBPS`
- `newSlippageBPS`
- `newSlippageInterval`

Alleviation

[Gelato Team]: These are 100% of the fees earned but not the principle. Also, there may be some nonstandard reason why one would want a G-UNI pool where all the fees go to the manager, for instance, if G-UNI were combined with another incentive scheme.

GUP-02 | Lack of Event Emissions for Significant Transaction

Category	Severity	Location	Status
Logical Issue	● Informational	abstract/GUniPoolStorage.sol: 156	✓ Resolved

Description

The function `GUniPoolStorage.initializeManagerFee()` updates `managerFeeBPS`, which is crucial parameter of the contract. An event should be emitted to log these updates.

Recommendation

We recommend emitting an event to log the update of `managerFeeBPS` when calling the function `GUniPoolStorage.initializeManagerFee()`.

Alleviation

The client heeded our advice and resolve the issue by emitting a new `SetManagerFee` event in the function `GUniPoolStorage.initializeManagerFee()`. The change is reflected in the commit `56a8e5403ba1823db0f108d4e85ad5d26699c0bb`.

GUP-03 | Centralization Risk

Category	Severity	Location	Status
Centralization / Privilege	● Minor	abstract/GUniPoolStorage.sol: 122, 150, 159	ⓘ Acknowledged

Description

The manager of the contract `GUniPoolStorage` has the following privileges by calling the listed functions:

1. `GUniPoolStorage.updateGelatoParams()`: Update the rebalance, fee and slippage configuration;
2. `GUniPoolStorage.initializeManagerFee()`: Initialize the uninitialized `managerFeeBPS`;
3. `GUniPoolStorage.renounceOwnership()`: Renounce the ownership, set the balances of `manager` role and `manageFeeBPS` to zero, and set the address of `managerTreasury` to zero address.

Any compromise to the `manager` account may allow the hacker to manipulate the project through these functions.

Recommendation

We recommend carefully managing the `manager` account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol to be improved via a decentralized mechanism or via smart-contract-based accounts with enhanced security practices, e.g. Multisignature wallets.

Indicatively, here are some feasible solutions that would also mitigate the potential risk:

- Time-lock with reasonable latency, e.g. 48 hours, for awareness on privileged operations;
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key;
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.

Alleviation

N/A

LAG-01 | Incorrect Comparison

Category	Severity	Location	Status
Volatile Code	● Medium	vendor/uniswap/LiquidityAmounts.sol: 74, 149	✓ Resolved

Description

In commit b6b885, Uniswap releases a patch for `LiquidityAmounts` to fix the `if` statements at the aforementioned lines. The incorrect `if` statements might lead to calculation errors.

Recommendation

We recommend changing `<` at the aforementioned lines to `<=`.

Alleviation

The client heeded our advice and resolved the issue in their codebase by replacing `<` at the aforementioned lines to `<=`. The changes are reflected in the commit `56a8e5403ba1823db0f108d4e85ad5d26699c0bb`.

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.

Logical Issue

Logical Issue findings detail a fault in the logic of the linked code, such as an incorrect notion on how block.timestamp works.

Volatile Code

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

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.

CERTIK