Security Audit Report

Plenny Smart Contracts BETA Release

Community self-audit conducted with tools openly available through OpenZeppelin, Slither and MythX™

Plenny Community | dplenny.crypto | March 2022

CONTENTS

1. Introduction 1
2. Security Tools Used 1
3. Risk Classification 3
4. Audit Scope 4
5. Nucleus Tests 5
6. Alpha Release Live Tests 5
7. Alpha Release Findings 6
7.1 Alpha Low Severity Issues 6
7.2 Alpha Medium Severity Issues 6
7.3 Alpha High Severity Issues 6
7.4 Alpha Release Bug Fixes 9
8. BETA Release Key Findings 10
8.1 BETA Low Severity Issues 10
8.2 BETA Medium Severity Issues 12
8.3 BETA High Severity Issues 12
8.4 BETA Release Security Audit Results 13
9. Dependencies 15
10. Proof of Security Audit 15
11. Safeguard Measures 16
12. Risk Level Conclusion 16
13. Intellectual Property 17
14. Disclaimer 18

1. Introduction

This security analysis report on Plenny's self-audit is both an educational and a technical resource. The content is designed to be equally useful for both technical and non-technical audiences.

The security audit report on Plenny's smart contracts relates to the Working Paper available on Plenny.crypto. Plenny is an online field experiment to research the interoperability of the Bitcoin Lightning Network with the Ethereum ecosystem. The Dapp (decentralized application) provides a capacity market and an oracle network for lightning nodes.

Continuous cybersecurity tests and iterative quality assurance procedures have been conducted by multiple engineers. The security tools described were used sequentially over an 11-month period of time. During the audit phase, multiple audit sessions were conducted to achieve the highest security for the smart contracts of the Dapp and to ensure the safety of users' tokens.

2. Security Tools Used

In March 2022, Plenny completed an extensive security self-audit. The audit was community-driven and conducted using the following tools:

- OpenZeppelin software
- Slither software
- MythX™ software

The OpenZeppelin ERC (Ethereum Request for Comment) standards were applied from the beginning and throughout the development process. These are standards for writing secure blockchain applications. Plenny's Dapp was programmed using pre-audited templates and secure libraries based on OpenZeppelin.

Slither is an open-source static analysis framework for Solidity smart contracts, enabling developers to find vulnerabilities in solidity smart contracts code. Slither was used for code review and optimization prior to the security audit using MythX™.

MythX™ by ConsenSys Software Inc™ is a smart contract security analysis service for Ethereum. The self-audit was performed with the comprehensive range of analysis techniques of MythX™, including static analysis, dynamic analysis, and symbolic execution to accurately detect security vulnerabilities and provide an in-depth analysis report.

In summary, using these three security tools, multistage scrutiny was carried out to assess and identify security flaws, as well as uncover vulnerabilities and bugs in the development of the smart contracts fundamental to the functionalities of the Plenny Dapp.

The security audit included both the smart contracts, known as Alpha release (Sept/Oct 2021), and the current version, referred to as BETA release (Feb 2022).

3. Risk Classification

The findings originating from MythX™ reports are classified as follows:

- **Low Severity Issues**

  This category of issues is considered to be of low severity. Such problems represent a breach of best practices.

- **Medium Severity Issues**

  The category of medium security issues is considered to be moderately severe. Such problems pose moderate risk and could represent a potential vulnerability. Their remediation is strongly recommended to ensure the secure functionality of the smart contracts of the Dapp. The fix is urgently needed to secure users' tokens in general.

- **High Severity Issues**

  This category of issues is considered critical because they pose a serious and immediate risk of exploitation. The MythX™ report strongly advises against releasing code with serious issues, as the consequences could be fatal for the smart contracts of the Dapp. The security of users' tokens would also be at risk. Fixing high severity issues is necessary and a top priority.

All three of these classifications may include *false positive results* depending on the application of smart contracts within the ecosystem.

4. Audit Scope

Originally, Plenny consisted of 26 smart contracts (Nucleus release). However, the BETA version consists of 20 smart contracts and 2 library contracts for a total of 22 audited smart contracts supported by Plenny.

Specifically, the smart contracts in use can be divided into different categories. There are so-called Plenny native smart contracts, interface contracts, library contracts, and storage contracts, all of which are based on OpenZeppelin standards. In addition, there is a set of external dependency contracts.

The security audit was conducted on the smart contract implementations (i.e. smart contracts and libraries, but not on interface contracts and not on storage contracts or external dependency contracts such as the external libraries by Uniswap, Sushi, and Arbitrum).

The 15 interface contracts were not subject to the audit as they don't involve implementation functions and serve only as plain functions such as using code splitting and inheritance patterns. In addition, the 12 storage contracts were also not included as they only define variables and variable types.

In summary, the scope of the audit report and the analysis it describes is limited to a review of the code of the given 22 smart contracts.

5. Nucleus Tests

In the interest of transparency, it is worth mentioning that the Nucleus release was not in the scope of the security audit.

During this starting phase of the project, smart contracts for generating tokens were tested once for experimental purposes (known among community members as “Operation Goldfinger”). The total supply for this trial was completely burned and can be verified via Etherscan.

6. Alpha Release Live Tests

The smart contracts were tested live in production after the Alpha release. During this experimental phase, tokens were allocated gradually through two reward contracts for interaction with Arbitrum, Sushi, and Uniswap.

This allocation of seed inventory and the subsequent allocation of reserves to smart contracts that programmatically provide token rewards is known as “Operation GoldenEye” among community members.

It should be noted that no tokens were lost, and no hacks took place during the initial phases. This track record refers to the Alpha version, but is also true for the Nucleus version.

7. Alpha Release Findings

As a result, the test operation of the Dapp live in production showed that some of the smart contracts have security weaknesses. In addition, it became apparent that a few contracts required improvements with regards to the token economy.

7.1 Alpha Low Severity Issues

Numerous issues of low and least severity were discovered during the initial security audit. The members of the community have recognized these minor issues and have decided to gradually remedy the corresponding problems as the development work progresses. In general, these low severity issues are considered a breach of best practice but do not represent a security threat. A change is not necessarily mandatory as the security of smart contracts is neither affected nor compromised in practice.

7.2 Alpha Medium Severity Issues

No medium severity issues were found.

7.3 Alpha High Severity Issues

In the final analysis, two high severity issues were found, along with one *false positive result*.

- **Alert I: Integer Overflow and Underflow**

  Recommendation: It is recommended to use safe math libraries for arithmetic operations.

  Description: An integer overflow/underflow occurs when an arithmetic operation attempts to create a numeric value that is outside of the range that can be represented with a given number of digits, meaning either higher than the maximum or lower than the minimum representable value.

  Remediation: To mitigate the integer overflow or underflow issues, the Plenny community has re-implemented the arithmetic operations using safe math libraries. The above issue was resolved in several of the Plenny's smart contracts.

- **Alert II: Reentrancy**

  Recommendation: It is recommended to perform any state change before the call is executed. This is known as Checks-Effects-Interactions pattern. It is also recommended to use a reentrancy lock (i.e. OZ's ReentrancyGuard).

  Description: A reentrancy attack is a recursive call attack, a malicious contract calls back into the calling contract before the first invocation of the function is finished.

  Remediation: To mitigate reentrancy issues, two security enhancements were made prior to the BETA release:

  First, by using OpenZeppelin's ReentrancyGuardUpgradeable contract, inheriting a non-reentrant modifier was implemented. This feature prevents a contract from calling itself, directly or indirectly. Secondly, by following best practices, the "Checks Effects Interactions" pattern was implemented so a function cannot be called again, thereby preventing a reentrancy attack.

- **Alert III: Use of Low-Level Functionality**

  Recommendation: To prevent this problem, the use of low-level operations and assemblers should be avoided.

  Description: This issue refers to PlennyTreasury.sol. The problem originates from OpenZeppelin's utility contract called "EnumerableSetUpgradeable." The report states that this dependency could potentially allow callers to redirect the execution. It appears to be possible to redirect the control flow to arbitrary bytecode locations. OpenZeppelin's pre-audited smart contract may allow an attacker to bypass security controls or manipulate the logic of the smart contract. The function returns the number of values in a set (its length), meaning the calls return the count of addresses with the passed role.

  Remediation: Contrary to the report, a closer examination of this issue revealed that trying to exploit this feature cannot result in the tokens being drained from the treasury. In particular, the function that calls .length, meaning "getRoleMemberCount()", is declared as a "public view" that cannot change the contract state.

  In addition, intensive live testing in production did not uncover any such security issues, leading to the conclusion that this problem refers to a *false positive result.*

  Moreover, regarding the transfer function of the treasury smart contract, the security analysis has not reported any kind of security issue.

  Finally, after fixing other vulnerabilities subject to the Alpha security audit, a new analysis report did not report any high severity issue for the treasury contract. The reason for the all-clear signal appears to be related to the remediation of bugs in other smart contracts. The root cause could also refer to some security definitions updates by MythX™ and/or OpenZeppelin, or the earlier check may have been accidentally performed with an outdated version of the smart contract.

7.4 Alpha Release Bug Fixes

To improve security and maintainability, OpenZeppelin's extending contracts and proxy patterns were introduced, providing a certain degree of mutability for bug fixing or potential service enhancements.

To fix the bugs that appeared in the course of the Alpha release, a number of smart contracts were replaced with new implementations and revised versions were subsequently introduced in the BETA release. Consequently, the relevant legacy contracts from the Alpha release were retired and are no longer available.

8. BETA Release Key Findings

The following sections list the findings that came to light through the security audit report prior to the BETA release.

8.1 BETA Low Severity Issues

Low severity issues occur multiple times for some of the smart contracts. The specific status messages are provided and the security implications are explained below:

- **Alert I: "A floating pragma is set"**
  Recommendation: Use a fixed version of Solidity.
  Description: The programming applied is common in practice and widely used in the industry.
  Remediation: This issue is considered a *false positive result*. No action has been taken.

- **Alert II: "Multiple calls are executed in the same transaction"**
  Recommendation: It is recommended to avoid combining multiple calls in a single transaction; it is also recommended to always assume that external calls can fail and implement the contract logic to handle failed calls.
  Description: A call is executed following another call within the same transaction. So it is possible that the call is never executed if a prior call fails permanently.
  Remediation: No action has been taken because if implemented differently, the additional transaction costs are neither justified nor

practical. Moreover, the calls are trusted as they are part of Plenny's own codebase.

- **Alert III: "Potential use of block.number as source of randomness"**

  Recommendation: It is recommended to use external sources of randomness via oracles (this approach requires trusting an oracle, thus it may be reasonable to use multiple oracles). According to the SWC Registry for Smart Contract Weakness Classification and Test Cases, it is recommended to use Bitcoin block hashes because this blockchain requires a greater commitment.

  Description: The block.number is returned to determine in which block a particular action or event occurred (e.g. the time when the payment channel was added).

  Remediation: No action has been taken because the block number contains more verifiable information than a simple timestamp. Moreover, the Ethereum blockchain is recognized as a trusted network.

- **Alert IV: "Requirement violation"**

  Recommendation: If the required logical condition is too strong, it should be weakened to allow all valid external inputs.

  Description: This issue occurs when a validation is performed, which is intended behavior. The only difference is that this particular validation uses nested calls.

  Remediation: No action has been taken because although the implementation does not follow best practice, the security of smart contracts is not compromised in practice.

- **Alert V: "A control flow decision is made based on the environment variable block.number"**
  Recommendation: It is recommended to use external sources of randomness via oracles (this approach requires trusting an oracle, thus it may be reasonable to use multiple oracles). According to the SWC Registry for Smart Contract Weakness Classification and Test Cases, it is recommended to use Bitcoin block hashes because this blockchain requires a greater commitment.
  Description: The block.number is used in conditional or required statements, thus controlling the execution flow. For example, it is used to check when a user is able to collect token rewards, when a certain community trigger can be called.
  Remediation: No action has been taken because the block number is considered sufficiently tamper-proof. Moreover, the Ethereum blockchain is recognized as a trusted network.

8.2 BETA Medium Severity Issues

No issues were found.

8.3 BETA High Severity Issues

No issues were found.

8.4 BETA Release Security Audit Results

The following sections provide a summary of the key findings that surfaced during the security audit before the BETA release.

Counting	**Solidity Programming Language Source File**	**Low Severity**	**Medium Severity**	**High Severity**
1	PlennyERC20V2.sol	Yes	No	No
2	PlennyArbERC20.sol	Yes	No	No
3	PlennyBase.sol	Yes	No	No
4	PlennyBaseERC20.sol	Yes	No	No
5	PlennyBasePausableV2.sol	Yes	No	No
6	PlennyBaseUpgradableV2.sol	Yes	No	No
7	PlennyContractRegistry.sol	Yes	No	No
8	PlennyCoordinatorV2.sol	Yes	No	No
9	PlennyDappFactoryV2.sol	Yes	No	No
10	PlennyDistribution.sol	Yes	No	No
11	PlennyLiqMining.sol	Yes	No	No
12	PlennyOceanV2.sol	Yes	No	No
13	PlennyOracleValidatorV2.sol	Yes	No	No
14	PlennyRePLENishmentv3.sol	Yes	No	No

15	PlennyReward.sol	Yes	No	No
16	PlennyStaking.sol	Yes	No	No
17	PlennyTreasury.sol	Yes	No	No
18	PlennyValidatorElectionV2.sol	Yes	No	No
19	RewardLibV2.sol	Yes	No	No
20	ExtendedMathLib.sol	Yes	No	No
21	PlennyLocking.sol*	Yes	No	No
22	PlennyDao.sol*	Yes	No	No

*Release to a closed user group (i.e. early adopters) for testing purposes on Arbitrum Rinkeby (Test Network), further release to new community members will occur in due course.

9. Dependencies

No instances of integer overflow and underflow vulnerabilities or back door entry were found in the smart contracts of the decentralized application.

For the sake of overall security, it must be stated that Plenny's smart contracts rely on external dependencies provided by OpenZeppelin, Arbitrum, Sushi, and Uniswap. The security of these pre-audited smart contracts with inherent functions was not subject to this security audit.

Any Solidity code poses unique and unquantifiable risks because the Solidity language itself is still under development and subject to unknown risks and flaws. The review does not extend to other areas outside of the specified code that could represent security risks. Cryptographic tokens are emerging technologies that carry a high degree of technical risk and uncertainty.

10. Proof of Security Audit

Plenny is in the position to provide evidence that security audit reports have been prepared. Analysis reports created with MythX™ are available upon request. The content contained in these analysis reports is current as of the point of time stated therein and is subject to change without notice, unless indicated otherwise by Plenny.

Technical details are shared on a peer-to-peer basis. The original documentation may not be published without express written permission. Access to reports cannot be granted for commercial purposes.

11. Safeguard Measures

Securing smart contracts is a multi-step and progressive process. Plenny continuously compiles security analysis reports according to best practices. The community is committed to producing new security audit reports prior to each new release. In addition, any critical issues or other known vulnerabilities that pose a risk are published on an ad hoc basis.

Plenny makes security reviews available over the Internet, and hopes that by making these analyses publicly available, it can help the blockchain ecosystem develop technical best practices in this rapidly evolving area of innovation.

White hat hackers are eligible to receive a bug bounty for reporting undiscovered vulnerabilities.

12. Risk Level Conclusion

Plenny is aware of the risks and consequences that cyber threats bring and firmly believes in the maxim *security first*. In all material respects, the smart contracts of the Dapp are well written and adhere to guidelines.

Overall, the risk level is **LOW**.

13. Intellectual Property

The intellectual property (IP) released by Plenny is licensed under the Creative Commons copyright license (2022).

The licensing models of Uniswap, Sushi, Arbitrum, Etherscan and Arbiscan are available online and are subject to the respective terms and conditions of these decentralized applications.

All logos, product and company names are trademarks of their respective holders. Use of them does not imply any affiliation with or endorsement by them. The following software security auditing tools were used by independently contributing bona fide members of the Plenny community for educational purposes only:

- MythX software is licensed under the MIT license. MythX™ and ConsenSys Software Inc™ are trademarks of ConsensSys Inc. in New York City, USA and/or ConsenSys AG in Zug, Switzerland. © 2020
- Slither software is licensed and distributed under the AGPLv3 license from Trailofbits.com. The Slither logo has been released as open source by Trail of Bits, Inc. based in New York, USA, and/or its affiliates (2018). Paper published by Josselin Feist, Gustavo Grieco, Alex Groce (2019).
- OpenZeppelin secure smart contract software is licensed under the MIT license. OpenZeppelin is a trademark of zOS Global Limited, Cayman Island and/or its affiliates. © 2017

14. Disclaimer

This audit report is a not a security warranty or approval of the Plenny-Dapp, since it does not provide a smart contract code faultlessness guarantee. The statements made in this document should not be interpreted as investment advice, nor should the community be held accountable for decisions made based on this security audit report.