



Three Sigma Labs

Code Audit



Syrup Lending Protocol

Disclaimer

Code Audit

Syrup Lending Protocol

Disclaimer

The ensuing audit offers no assertions or assurances about the code's security. It cannot be deemed an adequate judgment of the contract's correctness on its own. The authors of this audit present it solely as an informational exercise, reporting the thorough research involved in the secure development of the intended contracts, and make no material claims or guarantees regarding the contract's post-deployment operation. The authors of this report disclaim all liability for all kinds of potential consequences of the contract's deployment or use. Due to the possibility of human error occurring during the code's manual review process, we advise the client team to commission several independent audits in addition to a public bug bounty program.

Table of Contents

Code Audit

Syrup Lending Protocol

Table of Contents

Disclaimer	3
Summary	7
Scope	9
Methodology	11
Project Dashboard	13
Code Maturity Evaluation	16
Findings	19
3S-Sy-L01	19
3S-Sy-L02	20
3S-Sy-N01	21
3S-Sy-N02	22
3S-Sy-N03	23

Summary

Code Audit

Syrup Lending Protocol

Summary

Three Sigma Labs audited Syrup in a 2 days engagement. The audit was conducted from 21-05-2024 to 22-05-2024.

Protocol Description

The Syrup platform, built by Maple Labs, enables users permissionless access to secured, institutional lending for the first time. By depositing USDC into the platform, users receive LP tokens (syrupUSDC) and begin earning yield immediately. All of the yield generated by Syrup is sourced from secured loans to the largest institutions in crypto, fully collateralized with digital assets.

Scope

Code Audit

Syrup Lending Protocol

Scope

SyrupRouter.sol
SyrupRateProvider.sol

Assumptions

The scope of the audit was carefully defined to include the contracts at the lowest level of the inheritance hierarchy, as these are the ones that will be deployed to the mainnet. The libraries used in the implementation of these contracts are internal contracts that have been previously audited and shown to be secure.

Methodology

Code Audit

Syrup Lending Protocol

Methodology

To begin, we reasoned meticulously about the contract's business logic, checking security-critical features to ensure that there were no gaps in the business logic and/or inconsistencies between the aforementioned logic and the implementation. Second, we thoroughly examined the code for known security flaws and attack vectors. Finally, we discussed the most catastrophic situations with the team and reasoned backwards to ensure they are not reachable in any unintentional form.

Taxonomy

In this audit we report our findings using as a guideline Immunefi's vulnerability taxonomy, which can be found at immunefi.com/severity-updated/. The final classification takes into account the severity, according to the previous link, and likelihood of the exploit. The following table summarizes the general expected classification according to severity and likelihood; however, each issue will be evaluated on a case-by-case basis and may not strictly follow it.

Severity / Likelihood	LOW	MEDIUM	HIGH
NONE	None		
LOW	Low		
MEDIUM	Low	Medium	Medium
HIGH	Medium	High	High
CRITICAL	High	Critical	Critical

Project Dashboard

Code Audit

Syrup Lending Protocol

Project Dashboard

Application Summary

Name	Syrup
Commit	bd74ca9 and 4aab188
Language	Solidity
Platform	Ethereum

Engagement Summary

Timeline	21 to 22 May, 2024
Nº of Auditors	2
Review Time	2 days

Vulnerability Summary

Issue Classification	Found	Addressed	Acknowledged
Critical	0	0	0
High	0	0	0
Medium	0	0	0
Low	2	2	0
None	3	1	2

Category Breakdown

Suggestion	1
Documentation	0
Bug	2
Optimization	0
Good Code Practices	2

Code Maturity Evaluation

Code Audit

Syrup Lending Protocol

Code Maturity Evaluation

Code Maturity Evaluation Guidelines

Category	Evaluation
Access Controls	The use of robust access controls to handle identification and authorization and to ensure safe interactions with the system.
Arithmetic	The proper use of mathematical operations and semantics.
Centralization	The presence of a decentralized governance structure for mitigating insider threats and managing risks posed by contract upgrades
Code Stability	The extent to which the code was altered during the audit.
Upgradeability	The presence of parameterizations of the system that allow modifications after deployment.
Function Composition	The functions are generally small and have clear purposes.
Front-Running	The system's resistance to front-running attacks.
Monitoring	All operations that change the state of the system emit events, making it simple to monitor the state of the system. These events need to be correctly emitted.
Specification	The presence of comprehensive and readable codebase documentation.
Testing and Verification	The presence of robust testing procedures (e.g., unit tests, integration tests, and verification methods) and sufficient test coverage.

Code Maturity Evaluation Results

Category	Evaluation
Access Controls	Satisfactory. The codebase has a strong access control mechanism.
Arithmetic	Satisfactory. The codebase uses Solidity version >0.8.0 as well as takes the correct measures in rounding the results of arithmetic operations.
Centralization	Weak. Users of the Syrup Router may be DoSed at any time by the permissions admin.
Code Stability	Satisfactory. The code was stable during the audit.
Upgradeability	Moderate. Certain smart contract implementations can be modified after deployment, albeit with proper timelocks and functional upgradeability patterns.
Function Composition	Satisfactory. Certain components are similar, and the codebase would benefit from increased code reuse.
Front-Running	Moderate. Some front-running issues were found with permits, but they are not significant.
Monitoring	Satisfactory. Events are correctly emitted and the Syrup Router keeps track of on chain activity.
Specification	Satisfactory. In-depth and well structured high-level specification as well as codebase documentation.
Testing and Verification	Satisfactory. Extensive test code coverage as well as usage of tools and different test methods.

Findings

Code Audit

Syrup Lending Protocol

01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00
00 00 00 00 3B A3 ED FD 7A 7B 12 B2 7A C7 2C 3E
67 76 8F 61 7F C8 1B C3 88 8A 51 32 3A 9F B8 AA
4B 1E 5E 4A 29 AB 5F 49 FF FF 00 1D 1D AC 2B 7C
01 01 00 00 00 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00
00 00 00 00 00 FF FF FF FF 4D 04 FF FF 00 1D
01 04 45 54 68 65 20 54 69 6D 65 73 20 30 33 2F
4A 61 6E 2F 32 30 30 39 20 43 68 61 6E 63 65 6C
6C 6F 72 20 6F 6E 20 62 72 69 6E 6B 20 6F 66 20
73 65 63 6F 6E 64 20 62 61 69 6C 6F 75 74 20 66
6F 72 20 62 61 6E 6B 73 FF FF FF FF 01 00 F2 05
2A 01 00 00 00 43 41 04 67 8A FD B0 FE 55 48 27
19 67 F1 A6 71 30 B7 10 5C D6 A8 28 E0 39 09 A6
79 62 E0 EA 1F 61 DE B6 49 F6 BC 3F 4C EF 38 C4
F3 55 04 E5 1E C1 12 DE 5C 38 4D F7 BA 0B 8D 57
8A 4C 70 2B 6B F1 1D 5F AC 00 00 00 00
30 31 30 30 30 30 30 30 30 36 66 65 32 38 63 30 61
61 65 63 66 37 34 66 39 33 31 65 38 33 36 35
65 31 35 61 30 38 39 63 36 38 64 36 31 39 30 30
30 31 30 30 30 30 30 30 30 39 38 32 30 35 31 66 64
31 66 65 63 31 34 36 37 37 62 61 31 61 33 63 33
35 34 30 62 66 37 62 31 63 64 62 36 30 36 65 38
35 37 32 33 33 65 30 65 36 31 62 63 36 36 34 39
66 66 66 66 30 30 31 64 30 31 65 33 36 32 39 39
30 31 30 31 30 30 30 30 30 30 30 30 31 30 30 30
30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30
30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30
30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30
30 30 30 30 30 30 30 30 30 30 30 30 66 66 66 66
66 66 66 66 30 37 30 34 66 66 66 66 30 30 31 64
30 31 30 34 66 66 66 66 66 66 66 66 30 31 30 30
66 32 30 35 32 61 30 31 30 30 30 30 30 30 30 30
34 31 30 34 39 36 62 35 33 38 65 38 35 33 35 31
39 63 37 32 36 61 32 63 39 31 65 36 31 65 63 31
31 36 30 30 61 65 31 33 39 30 38 31 33 61 36 32
37 63 36 36 66 62 38 62 65 37 39 34 37 62 65 36
33 63 35 32 64 61 37 35 38 39 33 37 39 35 31 35
64 34 65 30 61 36 30 34 66 38 31 34 31 37 38 31
65 36 32 32 39 34 37 32 31 31 36 36 62 66 36 32
31 65 37 33 61 38 32 63 62 66 32 33 34 32 63 38
35 38 65 65 61 63 30 30 30 30 30 30 30 30 30 30
30 31 30 30 30 30 30 30 34 38 36 30 65 62 31 38

Findings

3S-Sy-L01

Attacker may frontrun **SyrupRouter::depositWithPermit()** call and use a different **depositData_** as it is not signed

Id	3S-Sy-L01
Classification	Low
Severity	Low
Likelihood	Low
Category	Bug
Status	Addressed in #b64472a .

Description

SyrupRouter::depositWithPermit() accepts a permit signature and a **depositData_** as extra data to be emitted. An attacker may spot the signature and frontrun the transaction, but with a different **depositData_**, griefing the user.

Recommendation

A simple fix is not possible as the permit function does not accept any extra data. It should be possible to send the transaction through flashbots to mitigate this issue if needed.

3S-Sy-L02

SyrupRouter::depositWithPermit() may be DoSed by frontrunning **ERC20::permit()**

Id	3S-Sy-L02
Classification	Low
Severity	Medium
Likelihood	Low
Category	Suggestion
Status	Addressed in #b64472a .

Description

`SyrupRouter::depositWithPermit()` uses the permit functionality of the `asset` by calling `IERC20Like(asset).permit(owner_, address(this), amount_, deadline_, v_, r_, s_)`. Some griefer may spot the `SyrupRouter::depositWithPermit()` transaction, frontrun it and call the `IERC20Like::permit()` function directly, spending the nonce of the signature and DoSing the router.

Recommendation

An attacker has no profit from executing so it is not expected to happen, but some measures could still be taken. Some examples are:

1. Get the allowance and only call `IERC20Like::permit()` if it is smaller than `amount_`.
2. Use `try/catch`.
3. Flashbots.

3S-Sy-N01

Signatures in the **SyrupRouter** are not fully compatible with EIP712

Id	3S-Sy-N01
Classification	None
Category	Bug
Status	Acknowledged

Description

EIP712 defines the digest as `keccak256("\x19\x01" || domainSeparator || hashStruct(message))`.

`domainSeparator` is `keccak256(abi.encode(TYPE_HASH, _hashedName, _hashedVersion, block.chainid, address(this)))`.

`TYPE_HASH` is `keccak256("EIP712Domain(string name,string version,uint256 chainId,address verifyingContract)");`

`hashStruct(message)` is dependent on the application, **ERC20Permit** is a good example. In this case it would be something like:

```
AUTH_TYPEHASH = keccak256("Auth(address lender,uint256 nonce,uint256 bitmap,uint256 deadline)");
bytes32 structHash = keccak256(abi.encode(AUTH_TYPEHASH, msg.sender,
nonces[msg.sender]++, bitmap, deadline));
```

And the final digest is `keccak256(abi.encodePacked("\x19\x01", domainSeparator, structHash))`.

Recommendation

To be strictly compliant, the structure above should be applied. However, signatures are signed by Maple so it has no impact on users.

3S-Sy-N02

Hardcoded **1e18** in **SyrupRateProvider**

Id	3S-Sy-N02
Classification	None
Category	Good Code Practices
Status	Addressed in #e8054a4 .

Description

SyrupRateProvider hardcodes the **shares** when **calling**
IPoolLike(pool).convertToExitAssets(1e18);.

Recommendation

Consider using a constant state value instead along with a comment for better readability.

3S-Sy-N03

owner in **SyrupRouter** also requires **transfer** permission

Id	3S-Sy-N03
Classification	None
Category	Good Code Practices
Status	Acknowledged

Description

SyrupRouter:::_deposit() checks for **owner** permission to deposit, but due to **transferring** the shares later in **ERC20Helper::transfer()**, it also requires **P:transfer** permission.

Recommendation

According to the specifications, **P:transfer** and **P:transferFrom** should be permissionless, so this is not a problem. However, consider adding a comment in the code.