OPTIMUM
BLOCKCHAIN SECURITY

Defi Saver - TxSaver

Smart contract
Security Assessment

July, 2024

Table Of Contents

Disclaimer

This report should not be considered as a security guarantee, investment advice, endorsement or disapproval of any specific project or team. The report makes no claim that the code being reviewed is completely free of vulnerabilities, bugs or potential exploits. Additionally, the report does not assess the financial risk of any asset. Therefore, it is not intended for any third party to make any decisions to buy or sell any asset or product based on this report.

It is important to note that ensuring the security of code is an ongoing process that requires multiple measures. Therefore, it is highly recommended that best coding practices, comprehensive testing, internal audits and bug bounty programs be implemented in addition to this report.

It is the responsibility of the project team to ensure that the code being reviewed is functioning as intended, and that the recommendations provided in this report are thoroughly tested before deployment.

Overview Page

Summary

Project name	Defi Saver
URL	https://defisaver.com
Code	https://github.com/defisaver/defisaver-v3-contracts/tree/bca090682796fb14d2fd459ec3fcb3d75c88f2a2
Commit hash	bca090682796fb14d2fd459ec3fcb3d75c88f2a2
Mitigations commit hash	
Language	Solidity

The security assessment was made by one researcher over a period of five full days.

Contracts Assessed

Contract name
TxSaverExecutor.sol
BotAuthForTxSaver.sol
TxSaverBytesTransientStorage.sol
SafeModulePermission.sol
RecipeExecutor.sol
DFSExchangeWithTxSaver.sol
DFSSell.sol
LlamaLendSwapper.sol

Findings Summary

Severity	Found	Resolved	Partially resolved	Acknowledged
Critical	0	0	0	0
High	0	0	0	0
Medium	0	0	0	0
Low	6	2	2	2
Informational	2	1	0	1
Total	8	3	2	3

Classification of issues

Severity	
Critical	Issues that may directly result in loss of funds, and thus require an urgent fix.
High	Issues that may not be directly exploitable, or with a limited impact, are still required to be fixed.
Medium	Issues that are not necessarily security vulnerabilities, that are required to be fixed unless there is a clear reason not to.
Low	Subjective issues with a negligible impact.
Informational	Subjective issues or observations with negligible or no impact.

Findings

Issue #01	The *BotAuthForTxSaver* contract is missing events emissions in state changing functions
Severity	Low
Location	BotAuthForTxSaver.sol#L18-L26
Description	The *BotAuthForTxSaver* contract is used to manage approved callers for the *TxSaverExecutor* contract. However, state changing functions like *addCaller* and *removeCaller* do not emit any events which makes it harder to track off-chain changes made.
Recommendation	Consider adding two events that represent the changes made to the state in both these functions.
Resolution	Acknowledged.

Issue #02	Users may trick the DFS bot that executes transactions through the *TxSaverExecutor* contract to spend gas they will not cover
Severity	Low
Location	
Description	The TxSaver service allows DFS users to submit their digital signatures, which represent their intended operations within the system, to the DFS backend. The backend then sends on-chain transactions on their behalf. The gas costs for these transactions are typically covered by the user's Safe during the transaction execution. However, users can cause transactions to revert, leading the bot to incur gas costs that will not be reimbursed. For example, a user with a *Safe* owned by a single owner might withdraw all the funds intended to cover the gas costs, causing the transaction to revert. Despite this, the bot will still incur the gas costs. It's important to note that transaction success can vary between simulations and actual on-chain execution, sometimes due to factors such as slippage checks, among others, even if the user has no intention of causing a revert.
Recommendation	After discussing the issue with the team, they explained the backend mechanisms designed to prevent or mitigate it. The main mechanisms are: 1. The backend simulates transactions and only submits those that do not revert in the simulation. 2. The backend uses MEV private RPC. While these measures provide partial mitigation, they do not completely resolve the issue. It is still possible for a simulation to succeed, but if a user's transaction to withdraw the funds intended for covering gas costs is inserted immediately afterward, the

	on-chain transaction may fail. Although this scenario is unlikely, it remains a potential risk. To better mitigate this issue, we propose monitoring transactions that revert and block users who perform such activities from using the TxSaver service.
Resolution	Acknowledged.

Issue #03	**Potential temporary denial of service in case the *TxSaver* bot is a contract that batches calls to *TxSaverExecutor.executeTx***
Severity	**Low**
Location	[TxSaverExecutor.sol#L103-L105]()
Description	The *TxSaver* service offers DFS users to submit their digital signatures that represent their intents on operations in the system to the DFS backend which in turn sends on-chain transactions on their behalf through the *TxSaverExecutor* contract. A potential issue arises since these digital signatures can be submitted to the *Safe* owned by the signer directly as well. As discussed with the team, in the current version of the system, the bot that is supposed to call *TxSaverExecutor.executeTx* is an EOA which prevents this potential issue. However, if in the future the DFS team decides that the bot will be a contract that performs batching of calls to *TxSaverExecutor.executeTx* in a single transaction, then this entire batch might revert due to one (or more) calls to *TxSaverExecutor.executeTx* that have failed due to a front-running attack in which the digital signature is sent directly to one of the users *Safe* contracts. Full scenario:

1. Backend batches multiple digital signatures and runs a simulation that succeeds to call a function that batches calls to *TxSaverExecutor.executeTx*.
2. Attackers front-run the batch transaction with a call to an individual *Safe* contract with a user signature that was also used in the batch call made by the bot.
3. The bot transaction reverts and thus other users' transactions are not being submitted on-chain.

In theory, as long as there is more than a single call to *TxSaverExecutor.executeTx* in a batch, users can be denied service continuously.

Recommendation

As mentioned in Issue #02, the fact that the bot transactions will be relayed via private RPC services may partially mitigate this issue since potential attackers won't have access to the digital signatures that can be used for the attack. But the users themselves do know the digital signatures that are about to be batched and therefore can be the ones launching this attack.

In case the DFS team is willing to implement the described batching functionality in the future, we propose implementing it in a way that will not revert the entire transaction in case some calls have failed inside the loop.

Resolution

The issue is partially mitigated and is not relevant for the current version of the code, as the bot used is an EOA. However, the issue is noted for future versions.

Issue #04	Recipes meant to be executed via the *TxSaverExecutor* should not have more than one sell order in case *txSaverData.shouldTakeFeeFromPosition=true*
Severity	Low
Location	DFSExchangeWithTxSaver.sol#L25
Description	The DFS TxSaver system allows users to cover the bot's gas fees with a token payment taken from the *Safe's* source token right before selling in case *txSaverData.shouldTakeFeeFromPosition* is set to *true*. The issue however, is that the fee will be taken more than once for recipes with multiple sell orders with the same source token or will revert for other sell orders where the source token is different from the expected fee token.
Recommendation	As discussed with the team, they are aware of this issue, and it is currently solved off-chain by validating that a recipe can not have more than one sell order (including *LlamaLend* actions as well). We encourage the DFS team to add a warning about this edge case to the DFS docs so that potential integrators that do not use the DFS frontend will be aware of this limitation as well.
Resolution	Solved on the backend level which is out of scope for this audit, as described above.

Issue #05	Users can skip covering the bot’s gas payments by not including a sell order inside their recipe
Severity	Low
Location	RecipeExecutor.sol#L167
Description	In the current implementation, users sign an order that consists of the field *_txSaverData.shouldTakeFeeFromPosition*. This field is later used to determine whether or not fess should be taken from the position (source token) right before the swap to cover the gas costs paid by the DFS TxSaver bot. The issue, however, is that users may opt to set *_txSaverData.shouldTakeFeeFromPosition* to *true* but never include any sell order in their recipe and by doing this effectively avoid paying the fees.
Recommendation	As discussed with the DFS team, this issue will be handled by the backend, the bot will simulate any transaction before submitting and will drop transactions where fees are not paid as well as relay its transactions via private RPC.
Resolution	Solved on the backend level which is out of scope for this audit, as described above.

Issue #06	*LlamaLendSwapper*: State changing functions should be reentrancy protected
Severity	Low
Location	LlamaLendSwapper.sol#L146
Description	The *LlamaLendSwapper* contract includes several callback functions invoked by the *LlamaLend* contracts as part of the code flows initiated in the DFS *LlamaLend* wrapper contracts located in *contracts/actions/llamalend/advanced/*. Additionally, the *LlamaLendSwapper* contract has a function named *withdrawAll* that allows anyone to withdraw the entire balance of *debtToken* and *collToken* from the contract. The callback functions utilize external calls that might be untrusted. In certain scenarios, this could allow attackers to steal users' funds if they manage to hijack the call flow and invoke *withdrawAll* during the execution of a callback function.
Recommendation	Consider adding a *nonReentrant* modifier to all state changing functions in the *LlamaLendSwapper* contract.
Resolution	Fixed in 582648e94af88e383252cbd0e14846673629daec by implementing the auditor's recommendation.

Issue #07	Potential points of concern around the usage of transient storage in the system
Severity	Informational
Description	Although no concrete issues were identified with the usage of transient storage contracts in the current implementation, it's important to highlight a few potential edge cases that may arise in future versions of the code. The *TxSaverBytesTransientStorage* contract can act as a shared resource within the system, as values written to its transient storage are not tied to a specific user operation. In fact, the *TxSaverBytesTransientStorage* is one of the few contracts that function as shared resources, whereas most other contracts are designed to be user-specific. During the security review, we searched for potential attack vectors, specifically reentrancy attacks that could alter the transient storage of *TxSaverBytesTransientStorage* right before its usage. Currently, such attacks are impossible because the TxSaver bot is an Externally Owned Account (EOA), preventing reentrant calls. However, as discussed with the DFS team, future TxSaver bots might be contract-based. Another concern is the possibility of invoking the *TxSaverExecutor.executeTx* code path in addition to a code path that directly interacts with a user's Safe. This could result in reading "dirty" values from the *TxSaverBytesTransientStorage* contract, potentially causing unintended consequences. Therefore, we recommend reexamining the code (with a focus on these concerns) before designing and implementing such bot contracts.

Issue #08	Consider checking the support of EVM opcodes for different chains
Severity	Informational
Description	The codebase consists of newly introduced opcodes that are not necessarily supported in different EVM chains like *TSTORE* and *TLOAD* for example. The Ethereum hard fork known as Shanghai introduced several new features to the chain, including the *PUSH0* opcode. By default, Solidity contracts with a version >= 0.8.20 are compiled for the *shanghai* EVM version thus incorporating this new opcode into the bytecode. However, the *PUSH0* opcode is not fully supported by all EVM chains, such as Linea. The current codebase compiler version is *0.8.24* which renders it incompatible to be deployed on this set of chains.
Recommendation	Consider either downgrading the Solidity version to <0.8.20 or setting the EVM version of the compiler to *paris* in case support for Linea is needed, and in general check the compatibility and change the code in accordance.
Resolution	Acknowledged.

Gas optimizations

Optimization #01	***SafeModulePermission.disableModule*** **should fetch 10 modules only in the worst case**
Description	In the current version of the code, *SafeModulePermission.disableModule* fetches the first 10 modules from the *Safe* contract which are concluded in 10 different SLOAD operations. However, since the expected behavior in most transactions is that the desired module to disable will be among the first two modules.
Recommendation	We propose that instead of always fetching the first 10 modules, the function will be changed to fetch only the first item, compare it to the *_moduleAddr* and continue to do the same for the second item. Only in case it was not found in the two first items, fetch the rest. This way you will be able to reduce gas costs for the best case in cases of *Safe* contracts with more than 2 modules.
Resolution	Fixed in 859889dcbf8467fbaf1b8fb3a31eaf7e9889efcb by implementing the auditor's recommendation.

General Recommendations

- Ensure that state-changing functions emit events that accurately reflect the changes made to the contract state.
- Remove any unused or test code before deploying the smart contract to minimize security risks and improve its performance.
- Consider adding reentrancy guards to state changing functions that contain external calls.
- While dealing with external token contracts, conduct a thorough due diligence before adding external token contracts to the platform. Utilize secure wrapper libraries such as SafeERC20 for added protection.
- While dealing with upgradeable contracts, consider using a plugin to verify implementation contracts are upgrade safe. openzeppelin-upgrades is highly recommended for this.
- Implement a thorough testing process in a testnet before deploying code to mainnet to minimize risks and ensure stability.
- Consider introducing a contingency plan. The plan should include steps to mitigate risks and address potential issues in the event of unforeseen circumstances. It may include the following elements:
    - Regular code audits and security assessments to identify potential vulnerabilities and improve the robustness of the smart contract code.
    - Monitoring the smart contract's deployment and usage, with automated alerts in place to identify and respond to potential incidents quickly.
    - Establishing a clear process for updating and upgrading the smart contract code, in case of a critical bug.
    - Having a plan in place to limit the impact of an attack, such as implementing a fail-safe mechanism or shutting down the smart contract temporarily until the issue can be resolved.

- Implementing a bug bounty program to incentivize security researchers to report any security issues.
- Establishing clear communications with users and stakeholders, including regular updates on the performance of the smart contract and any potential risks.

OPTIMUM

BLOCKCHAIN SECURITY