Contract Audit: dETH Smart Contracts

Preamble

This audit report was undertaken by @adamdossa for the purpose of providing feedback to the Foundry / DAIHard team. It has been written without any express or implied warranty.

This audit was done on the code in Github at commit:
https://github.com/team-toast/dEth/commit/7f5d7edccd9633ce5a73c8272b156a84b4134110

Disclosure

Classification

- **Comment** - A note that highlights certain decisions taken and their impact on the contract.
- **Minor** - A defect that does not have a material impact on the contract execution and is likely to be subjective.
- **Moderate** - A defect that could impact the desired outcome of the contract execution in a specific scenario.
- **Major** - A defect that impacts the desired outcome of the contract execution or introduces a weakness that may be exploited.
- **Critical** - A defect that presents a significant security vulnerability or failure of the contract across a range of scenarios.

Audit

This audit focussed on the dETH contract. The dETH contract aims to provide a tokenised CDP position, where the position is automatically maintained via DeFiSaver.

The dETH contract therefore has many dependencies on third party contracts. These are partially represented by the contract parameters to the dETH constructor, with those contracts in turn relying on other contracts.

This audit aimed to understand the intended behaviour of those underlying contracts and protocols, but did not verify the behaviour of those contracts directly.

Summary

Overall the contract looks safe to me, given the caveats below and the intention to move to a time-locked approach for administration via the Foundary DAO. The issues found during audit were either directly mitigated or analysed and deemed non-material or transparent to users.

Contract Behaviour

The dETH contract is an ERC20 token (dETH) and also a DSProxy that controls a specific CDP.

The contract initialisation is done by passing an existing CDP into the contract, with the contract minting an initial dETH balance equal to the excess collateral in the CDP (denominated in ETH).

The dETH (DSProxy) contract is then passed ownership of the CDP. This allows it to schedule DeFiSaver automation and to add and free collateral from the CDP.

Additional dETH is then minted by taking a ratio of amount of new ETH being collateralised against the amount of existing excess collateral in ETH, as a percentage of the current total supply. It is redeemed in a similar fashion.

The deployed contract has a `giveCDPToDSProxy` function that would allow an authorised address to hand control of the CDP owned by the dETH to another arbitrary DSProxy contract. This function is expected to be removed at some point in the future, but allows an authorised address to rug pull the CDP from the dETH token holders until that point.

dETH Contract

Oracles

The dEth contract uses two chainlink oracles (ETH/USD and DAI/USD) to calculate the value of the CDPs debt (denominated in DAI), in terms of ETH.

Looking at these two chainlink oracles, the ETH/USD oracle seems to update every 5 - 30 minutes, and the DAI/USD oracle only updates every few hours, which seems inline with the expected behaviour. The oracles are updated when prices move more than a defined percentage, rather than on a regular cadence. This means their cross-rate should represent a reasonably accurate ETH / DAI price, although there may be some slippage as the USD / DAI oracle only updates on a 2% price change.

There is also a direct ETH / DAI oracle:
https://data.chain.link/dai-eth although it's trigger deviation threshold is 1% vs. 0.5% for the ETH / USD oracle.

It is possible there could be some basis between the chainlink oracle prices, and the real cost of converting DAI to ETH on-chain via a DEX - possibly some back-testing of oracle data vs uniswap pricing might add additional confidence here. Overall, using the chainlink oracle data seems to resolve the previous audit issue related to mis-pricing of excess collateral.

One point to note is that the conditions under which the oracle falls back to the previous makerDAO feed may be the conditions under which ETH (or DAI) are most volatility, and therefore exacerbate the previously identified issue where users can pay a lower than fair price to enter the contract (or receive a higher than fair price to exit the contract).

Mitigation

Back testing has been done by the Foundary team on divergence between the Maker and Chainlink oracles. This analysis shows that these oracles very rarely diverge by a large amount.

Automation

The automation fee is set to 1% on initialisation, but can be modified by an authorised user at a later date.

This allows the authorised user to front-run a deposit to the dETH contract and steal the users deposited ETH.

The attacker can monitor the transaction pool, and when it sees a large deposit (i.e. squanderMyEthForWorthlessBeans) front-run the transaction and set the automation fee to (100% - PROTOCOL_FEE_PERC). This means the user will receive no issued tokens for their deposit, and existing token holders get the benefit of the increased excess collateral. This could be resolved via a time-lock on changing this value.

It may also be worth considering making the fee a fixed value as that better aligns with the gas costs to interact with the contract.

Mitigation

The intention is to modify this authorised access to be via the Foundary DAO with time-locked access once the contract has stabalised.

Comment - Separation of dETH ERC20 and CDP controller

Currently the dETH contract is both the DSProxy that controls a CDP, and the dETH ERC20 token contract.

It may be worth considering moving the dETH ERC20 token to a separate contract, with the dETH token controller being the DSProxy that owns the CDP.

This would help provide a clearer separation in code, and also reduce attack surfaces due to delegate call functionality of the DSProxy.

NB - I can't identify any attack vectors, but this seems an unnecessarily complex way to implement this functionality.

Minor - Assumptions on underlying dependencies to calculate issuance / redemptions

The dETH contract uses third-party contracts to perform analysis on its underlying CDP when issuing or redeeming dETH.

If the behaviour or returned values from this function - `saverProxy.getCdpDetailedInfo` - changes, then these calculations may become incorrect.

The dETH contract also uses its `DSProxy::execute` function to execute various third-party logic through `freeETH`, `subscribe`, `give`, `lockETH`.

If these underlying contracts are immutable then this is fine, but if their logic can be modified in any way by a third-party this presents a risk.

I'd note that for the `DSProxy::execute` I think you could just directly execute the logic (from the third-party contracts) in the dETH contract itself, which would reduce gas costs and remove this risk.

Comment - Dependency on non-chain logic to maintain collateralisation ratio

The dETH contract relies on DeFiSaver to prevent liquidiation (and to increase leverage when the price of ETH increases). The DeFiSaver protocol is not enforced on-chain, and DeFiSaver may either choose not to, or be unable to, execute repay actions to prevent liquidation (for example if there is congestion on the chain).

In this instance, the CDP would be liquidated. From what I can tell, if this is the case, token holders will still be able to withdraw any remaining collateral that remains once the debt position has been auctioned off, and makerDAO fees applied. However a detailed analysis of makerDAO behaviour in this circumstances is outside the scope of this audit, so this behaviour should ideally be tested.

Notes

I analysed the code by eye rather than using automated tools and this audit should be considered in this context.