



PALADIN
BLOCKCHAIN SECURITY

Smart Contract Security Assessment

Final Report

For unshETH

15 June 2023



paladinsec.co



info@paladinsec.co

Table of Contents

Table of Contents	2
Disclaimer	4
1 Overview	5
1.1 Summary	5
1.2 Contracts Assessed	6
1.3 Findings Summary	7
1.3.1 ERC20PermitPermissionedMint	8
1.3.2 unshETH	8
1.3.3 Darknet	8
1.3.4 LSDVault	9
1.3.5 RenouncedOwner	9
1.3.6 unshETHZap	10
1.3.7 vdAMM	10
1.3.8 Owned	10
2 Findings	11
2.1 ERC20PermitPermissionedMint	11
2.1.1 Privileged Functions	11
2.1.2 Issues & Recommendations	12
2.2 unshETH	16
2.2.1 Privileged Functions	16
2.2.2 Issues & Recommendations	17
2.3 Darknet	19
2.3.1 Privileged Functions	19
2.3.2 Issues & Recommendations	20
2.4 LSDVault	23
2.4.1 Privileged Functions	24
2.4.2 Issues & Recommendations	25
2.5 RenouncedOwner	39
2.5.1 Issues & Recommendations	40
2.6 unshETHZap	41
2.6.1 Issues & Recommendations	42
2.7 vdAMM	47

2.7.1 Privileged Functions	48
2.7.2 Issues & Recommendations	49
2.8 Owned	54
2.8.1 Privileged Functions	54
2.8.2 Issues & Recommendations	55



Disclaimer

Paladin Blockchain Security ("Paladin") has conducted an independent audit to verify the integrity of and highlight any vulnerabilities or errors, intentional or unintentional, that may be present in the codes that were provided for the scope of this audit. This audit report does not constitute agreement, acceptance or advocacy for the Project that was audited, and users relying on this audit report should not consider this as having any merit for financial advice in any shape, form or nature. The contracts audited do not account for any economic developments that may be pursued by the Project in question, and that the veracity of the findings thus presented in this report relate solely to the proficiency, competence, aptitude and discretion of our independent auditors, who make no guarantees nor assurance that the contracts are completely free of exploits, bugs, vulnerabilities or deprecation of technologies. Further, this audit report shall not be disclosed nor transmitted to any persons or parties on any objective, goal or justification without due written assent, acquiescence or approval by Paladin.

All information provided in this report does not constitute financial or investment advice, nor should it be used to signal that any persons reading this report should invest their funds without sufficient individual due diligence regardless of the findings presented in this report. Information is provided 'as is', and Paladin is under no covenant to the completeness, accuracy or solidity of the contracts audited. In no event will Paladin or its partners, employees, agents or parties related to the provision of this audit report be liable to any parties for, or lack thereof, decisions and/or actions with regards to the information provided in this audit report.

Cryptocurrencies and any technologies by extension directly or indirectly related to cryptocurrencies are highly volatile and speculative by nature. All reasonable due diligence and safeguards may yet be insufficient, and users should exercise considerable caution when participating in any shape or form in this nascent industry.

The audit report has made all reasonable attempts to provide clear and articulate recommendations to the Project team with respect to the rectification, amendment and/or revision of any highlighted issues, vulnerabilities or exploits within the contracts provided. It is the sole responsibility of the Project team to sufficiently test and perform checks, ensuring that the contracts are functioning as intended, specifically that the functions therein contained within said contracts have the desired intended effects, functionalities and outcomes of the Project team.

Paladin retains the right to re-use any and all knowledge and expertise gained during the audit process, including, but not limited to, vulnerabilities, bugs, or new attack vectors. Paladin is therefore allowed and expected to use this knowledge in subsequent audits and to inform any third party, who may or may not be our past or current clients, whose projects have similar vulnerabilities. Paladin is furthermore allowed to claim bug bounties from third-parties while doing so.

1 Overview

This report has been prepared for unshETH on the Ethereum network. Paladin provides a user-centred examination of the smart contracts to look for vulnerabilities, logic errors or other issues from both an internal and external perspective.

1.1 Summary

Project Name	unshETH
URL	https://unsheth.xyz/
Platform	Ethereum
Language	Solidity
Preliminary Contracts	https://github.com/UnshETH/unsheth-contracts-audit/tree/67f5071ef7365e01b85aea2748764c9c2945f82c
Resolution 1	https://github.com/UnshETH/contracts-audit/tree/fbeeb929eb52e72557ce2ed2a5ae689ae5bf3f70

1.2 Contracts Assessed

Name	Contract	Live Code Match
ERC20PermitPermissionedMint	Dependency	✓ MATCH
unshETH	0x0Ae38f7E10A43B5b2fB064B42a2f4514cbA909ef	✓ MATCH
Darknet	0x1a890EBCD20a9fB551C0440428805BC24eF62641	✓ MATCH
LSDVault	0x51A80238B5738725128d3a3e06Ab41c1d4C05C74	✓ MATCH
RenouncedOwner	0xb250216b5bcE306Fa37F7DE76A82409663eD52c3	✓ MATCH
unshETHZap	0xc258fF338322b6852C281936D4EdEff8Adff23eE	✓ MATCH
vdAMM	0x35636B85B68C1b4A216110fB3A5FB447a99DB14A	✓ MATCH
Owned	Dependency	✓ MATCH



1.3 Findings Summary

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
● High	0	-	-	-
● Medium	6	3	2	1
● Low	9	1	2	6
● Informational	17	2	-	15
Total	32	6	4	22

Classification of Issues

Severity	Description
● High	Exploits, vulnerabilities or errors that will certainly or probabilistically lead towards loss of funds, control, or impairment of the contract and its functions. Issues under this classification are recommended to be fixed with utmost urgency.
● Medium	Bugs or issues that may be subject to exploit, though their impact is somewhat limited. Issues under this classification are recommended to be fixed as soon as possible.
● Low	Effects are minimal in isolation and do not pose a significant danger to the project or its users. Issues under this classification are recommended to be fixed nonetheless.
● Informational	Consistency, syntax or style best practices. Generally pose a negligible level of risk, if any.

1.3.1 ERC20PermitPermissionedMint

ID	Severity	Summary	Status
01	LOW	The <code>minter_burn_from</code> function is redundant	ACKNOWLEDGED
02	INFO	Set of minters lacks proper enumerability	ACKNOWLEDGED
03	INFO	Typographical issues	ACKNOWLEDGED
04	INFO	Gas optimizations	ACKNOWLEDGED

1.3.2 unshETH

ID	Severity	Summary	Status
05	MEDIUM	Governance risk: Any of the registered minters can freely mint tokens	✓ RESOLVED
06	INFO	Typographical issue	ACKNOWLEDGED

1.3.3 Darknet

ID	Severity	Summary	Status
07	LOW	Contract design is extremely limited	✓ RESOLVED
08	INFO	Lack of validation	✓ RESOLVED
09	INFO	Typographical issues	✓ RESOLVED

1.3.4 LSDVault

ID	Severity	Summary	Status
10	MEDIUM	migrateVault contains an error which prevents this function from being called, preventing the team from ever executing migrations	ACKNOWLEDGED
11	MEDIUM	Governance risk: Timelock functionality can be bypassed by the unshETH team to migrate and drain the vault	✓ RESOLVED
12	MEDIUM	LSD array logic is flawed, allowing for the first LSD to be added and enumerability to be more limited than it needs to be	PARTIAL
13	MEDIUM	Lack of cross-contract reentrancy checks can lead to serious side-effects due to the usage of multiple contracts	✓ RESOLVED
14	LOW	LSDVault is not sufficiently redundant against compromised LSDs, allowing a hacked LSD to potentially drain the vault	ACKNOWLEDGED
15	LOW	Shangai time safeguard is futile as it can be called repeatedly	PARTIAL
16	INFO	Division is done before multiplication within exit, causing unnecessary precision loss	ACKNOWLEDGED
17	INFO	Typographical issues	ACKNOWLEDGED
18	INFO	Gas optimizations	ACKNOWLEDGED

1.3.5 RenouncedOwner

ID	Severity	Summary	Status
19	INFO	Typographical error	ACKNOWLEDGED



1.3.6 unshETHZap

ID	Severity	Summary	Status
20	MEDIUM	The external deposit_stEth function is broken as it does not pull in any stEth from the user	PARTIAL
21	LOW	deposit_1sd lacks a reentrancy guard	PARTIAL
22	LOW	Some interfaces do not match the actual function interface which may cause issues with future compiler versions	ACKNOWLEDGED
23	INFO	Gas optimizations	ACKNOWLEDGED
24	INFO	Typographical issues	ACKNOWLEDGED

1.3.7 vdAMM

ID	Severity	Summary	Status
25	LOW	1sds lacks a length function, making it tedious for the frontend to explore the array efficiently	ACKNOWLEDGED
26	LOW	withdrawStuckEth might not work with certain multi-signature wallets and timelocked owners	ACKNOWLEDGED
27	LOW	vdAMM lacks edge-case safeguards	ACKNOWLEDGED
28	INFO	Gas optimizations	ACKNOWLEDGED
29	INFO	Typographical issues	ACKNOWLEDGED

1.3.8 Owned

ID	Severity	Summary	Status
30	INFO	The zero address can claim ownership	ACKNOWLEDGED
31	INFO	Gas optimization	ACKNOWLEDGED
32	INFO	Typographical issue	ACKNOWLEDGED

2 Findings

2.1 ERC20PermitPermissionedMint

ERC20PermitPermissionedMint is the ERC20 implementation used for the unshETH token. It extends the standard OpenZeppelin token implementation with certain governance privileges:



- A set of minters can mint coins freely.
- A timelock and owner can freely add and remove minter addresses.

The OpenZeppelin Permit and Burnable extensions are also used, which means that the token is burnable by and for users, and that the tokens can be transferred from a user given that the user signs an approval, instead of submitting one.

2.1.1 Privileged Functions

- `minter_burn_from [minter]`
- `minter_mint [minter]`
- `addMinter [owner / timelock_address]`
- `removeMinter [owner / timelock_address]`
- `setTimelock [owner / timelock_address]`
- `nominateNewOwner [owner]`
- `acceptOwnership [nominated owner]`

2.1.2 Issues & Recommendations

Issue #01	The <code>minter_burn_from</code> function is redundant
Severity	 LOW SEVERITY
Description	<p>ERC20PermitPermissionedMint includes a <code>minter_burn_from</code> function. However, this function is identical to <code>burnFrom</code> except that an extra event is emitted (meaning this function now emits two events which is a waste of gas).</p>
Recommendation	<p>Consider whether the internal <code>_burn</code> function is intended to be used instead within the function body. This internal function does not require an allowance unlike the public <code>burnFrom</code> function.</p> <p>If this was not intended, consider simply removing the <code>minter_burn_from</code> function and sticking to the standard <code>burnFrom</code> function to keep the code simple.</p>
Resolution	 ACKNOWLEDGED

Issue #02**Set of minters lacks proper enumerability****Severity** INFORMATIONAL**Description**

The minters set is represented as an array and mapping. This is restrictive as it prevents minters from being removed cleanly from the array. Furthermore, there is no way to know how many minters there are except by trial-and-error through accessing each of the array indices until the function starts reverting.

Recommendation

Consider using the superior EnumerableSet structure introduced by OpenZeppelin. We recommend moving to a set of minters and adding the following UI methods:

- `minterAt(index)`
- `minterLength()`

Resolution ACKNOWLEDGED

No changes were made as this is a contract directly forked from Frax and the team wishes for the contract to be identical to the original.



Issue #03	Typographical issues
Severity	<div><div></div>INFORMATIONAL</div>
Description	<p><u>Line 46</u></p> <pre>require(minters[msg.sender] == true, "Only minters");</pre> <p><code>== true</code> is not necessary here since the first part of the statement is already a boolean.</p> <p><u>Line 78</u></p> <pre>require(minters[minter_address] == true, "Address nonexistant");</pre> <p><code>== true</code> is not necessary here since the first part of the statement is already a boolean. This error should also say "non-existent".</p> <p>It is an anti-pattern to use <code>snake_case</code> within Solidity. We recommend sticking to <code>camelCase</code> instead.</p>
Recommendation	Consider fixing the typographical issues.
Resolution	<div><div></div>ACKNOWLEDGED</div> <p>No changes were made as this is a contract directly forked from Frax and the team wishes for the contract to be identical to the original.</p>



Severity

 INFORMATIONAL

Description

Line 77

```
require(minter_address != address(0), "Zero address  
detected");
```

This check is not necessary as these addresses cannot be flagged as true within the minters mapping. This is a non-critical code location so this is provided purely informationally as gas optimizations are not so relevant here.

Line 84

```
for (uint i = 0; i < minters_array.length; i++){
```

Consider caching the length in a local variable to save significant amounts of storage access gas costs. This is a non-critical code location so this is provided purely informationally as gas optimizations are not so relevant here.

Recommendation

Consider implementing the gas optimizations mentioned above.

Resolution

 ACKNOWLEDGED

No changes were made as this is a contract directly forked from Frax and the team wishes for the contract to be identical to the original.

2.2 unshETH



unshETH is a critical token within the unshETH ecosystem. It represents a basket of LSD tokens and increases in value over time as these underlying tokens grow in value due to validator rewards and swaps and fees within the unshETH ecosystem.

This token is a simple implementation of the previously-mentioned ERC20PermitPermissionedMint and we highly recommend reading that description to better understand how the token works and who can mint it.



2.2.1 Privileged Functions

- `minter_burn_from [minter]`
- `minter_mint [minter]`
- `addMinter [owner / timelock_address]`
- `removeMinter [owner / timelock_address]`
- `setTimelock [owner / timelock_address]`
- `nominateNewOwner [owner]`
- `acceptOwnership [nominated owner]`

2.2.2 Issues & Recommendations

Issue #05	Governance risk: Any of the registered minters can freely mint tokens
Severity	 MEDIUM SEVERITY
Description	<p>ERC20PermitPermissionedMint allows any governance-registered minting wallet to freely mint tokens. This means that if just one of these registered wallets is compromised, the whole project might be considered compromised since this wallet can then mint and dump all tokens (or use them to drain the underlying reserves).</p> <p>It should be noted that both the registered <code>timelock_address</code> and owner can freely add new minter addresses. This means that they indirectly can freely mint tokens, alongside with any already registered minter addresses.</p>
Recommendation	<p>Consider extremely carefully restricting the parties who can mint these tokens. It may also make sense to add an hourly or daily cap on the percentage of tokens that can be minted (e.g. up to 10% of the total supply may be minted every day) as an extra safeguard. Adding such arbitrary safeguards can be extremely useful in reducing the impact of exploits/key compromise.</p>
Resolution	 RESOLVED

The client had apparently already resolved this issue in production way before the audit started. They have fully renounced the owner role of the token and have indicated that the vault is the only minter of the token. They have also indicated that no excess supply has been minted. We did not verify this personally and strongly urge large exposed users to ensure that no additional supply was minted before ownership was renounced.

Issue #06	Typographical issue
Severity	 INFORMATIONAL
Location	<u>Line 9</u> <code>address public LSDRegistryAddress;</code>
Description	This line is unused and the value is therefore always zero.
Recommendation	Consider fixing the typographical issue.
Resolution	 ACKNOWLEDGED



2.3 Darknet

Darknet represents the oracle hub for the unshETH codebase. It is the interface for the other contracts to discover how much each individual wrapped LSD is worth in ETH.



It should be noted that unlike a traditional oracle, this contract simply calls a configurable function on the LSD to let the LSD return how much ETH it is worth. This means that this contract does not rely on more complex architectures like ChainLink and is therefore less vulnerable to oracle attacks (unless the actual underlying LSD can be manipulated, which is a scenario which should not be excluded).



For each LSD contract, the Darknet owner can configure a function once which can be called to figure out the value of that LSD. This function must return the LSD value in ETH multiplied by $1e18$. Take wstETH for example, this would be the `stEthPerToken()` function which returns $1.126 * 1e18$ at the time of this audit, or a value of 1.126 ETH per wrapped LSD token.

2.3.1 Privileged Functions



- `nominateNewOwner [owner]`
- `acceptOwnership [nominated owner]`

2.3.2 Issues & Recommendations

Issue #07	Contract design is extremely limited
Severity	 LOW SEVERITY
Location	<u>Line 9</u> <i>//need to store 1. function signature 2. conversion (if any)</i>
Description	<p>This comment indicates that the contract does conversions of the returned ratio if necessary. E.g., if the ratio is in 1e4 instead of 1e18, this contract should be able to do that. This is however not implemented.</p> <p>This contract is extremely straightforward which will undoubtedly cause issues when the team wants to implement LSDs which do not have a function that returns this price in 1e18.</p>
Recommendation	<p>Consider relying on a meta Darknet implementation:</p> <pre>contract MetaDarknet is IDarknet, Owned { mapping(address => IDarknet) public darknets; event DarknetAdded(address lsd, IDarknet link); function checkPrice(address lsd) external view returns (uint256) { return darknets[lsd].checkPrice(lsd); } function addLSD(address _lsd, IDarknet _darknet) external onlyOwner { require(darknets[_lsd] == bytes4(0), "Darknet already exists"); darknets[_lsd] = _darknet; emit DarknetAdded(_lsd, _darknet); } }</pre> <p>Such an implementation provides great flexibility as it allows for configuring custom oracles for any LSD if necessary.</p>
Resolution	 RESOLVED
	<p>The client moved to a static design for the contract by simply redeploying it every time a new LSD is added. The code is quite trivial now and easily auditable. The addRouter function has been removed.</p>

Issue #08	Lack of validation
Severity	 INFORMATIONAL
Location	<u>Line 33</u> <code>function addRouter(address _lsd, bytes4 _link) external onlyOwner {</code>
Description	It might make sense to call checkPrice on the _lsd as a way to check that the function at least exists and returns a number. This way the configurational risk is reduced.
Recommendation	Consider validating the function parameters mentioned above.
Resolution	 RESOLVED The function has been removed.



Issue #09	Typographical issues
Severity	<div data-bbox="454 165 636 199">  INFORMATIONAL </div>
Description	<p data-bbox="454 248 564 282"><u>Line 17</u></p> <pre data-bbox="454 293 916 327">router[_lsds[i]] = _links[i];</pre> <p data-bbox="454 365 1123 398">This should probably emit RouterAdded events.</p> <p data-bbox="454 488 564 521"><u>Line 23</u></p> <pre data-bbox="454 533 1286 611">function checkPrice(address lsd) public view returns (uint256) {</pre> <p data-bbox="454 649 1046 683">This function can be marked as external.</p> <p data-bbox="454 772 564 806"><u>Line 34</u></p> <pre data-bbox="454 817 1414 851">require(router[_lsd] == bytes4(0), "Router already exists");</pre> <p data-bbox="454 889 1390 1064">This requirement seems restrictive as there is a very small chance that the zero selector is exactly the selector necessary to fetch the pricing data. Consider using an EnumerableMap by OpenZeppelin instead.</p>
Recommendation	Consider fixing the typographical issues.
Resolution	<div data-bbox="454 1178 595 1211">  RESOLVED </div> <p data-bbox="454 1234 975 1267">Most of these issues have been fixed.</p>



2.4 LSDVault

LSDVault manages and maintains all LSDs deposited within unshETH.

The owner of the contract can enlist new LSD tokens to the vault. For each of these tokens, the owner can configure a target LSD proportion for the vault, a relative cap (weight) and an absolute cap.

Users can, at any point, deposit through the configured zipper contract as long as deposits are not paused.

The owner is able to take out all assets of the vault and do other intrusive actions like replacing the oracle (Darknet) and the zipper, but this can only be done by going through a 3 day timelock. Larger depositors should therefore set up listeners on the owner address to monitor timelock transactions (`createTimelockProposal` calls).



Another critical feature of the vault is its ability to configure a swap AMM contract which has full liberty to take out LSD tokens. This means that this contract needs to be secure as well and must be audited whenever its adjusted through the timelock.



2.4.1 Privileged Functions

- `setUnshethZap [owner, callable once (already called)]`
- `setAdmin [owner]`
- `addLSD [owner, if paused]`
- `setLSDConfigs [owner, if paused]`
- `enableLSD [owner, if paused]`
- `enableAllLSDs [owner, if paused]`
- `disableLSD [owner]`
- `toggleWeightCaps [owner]`
- `toggleAbsoluteCaps [owner]`
- `toggleV1VaultAssetsForCaps [owner]`
- `unpauseDeposits [owner]`
- `deposit [zapper]`
- `setRedeemFee [owner]`
- `createTimelockProposal [owner]`
- `cancelTimelockProposal [owner]`
- `updateUnshethZapAddress [owner]`
- `updateUnshethZapAddress [owner]`
- `migrateVault [owner]`
- `setVdAmm [owner]`
- `updateShanghaiTime [owner]`
- `pauseDeposits [owner / admin]`
- `pauseWithdrawals [owner / admin]`
- `unpauseWithdrawals [owner / admin]`
- `disableVdAmm [owner / admin]`
- `depositNoCapCheck [zapper]`
- `nominateNewOwner [owner]`
- `acceptOwnership [nominated owner]`

2.4.2 Issues & Recommendations

Issue #10	migrateVault contains an error which prevents this function from being called, preventing the team from ever executing migrations
Severity	 MEDIUM SEVERITY
Location	<u>Line 493-495</u> <pre>unshETH.addMinter(proposedVaultAddress); unshETH.setTimelock(proposedVaultAddress); unshETH.removeMinter(address(this));</pre>
Description	<p>The vault contains an emergency function callable by the vault owner timelock which allows for the vault assets to be moved to a new address.</p> <p>By using the migrate function, the unshETH team can move out the vault assets in emergencies. This can be useful if they need to upgrade to a new vault contract or for any other reason (e.g. securing assets away from a vulnerability).</p> <p>However, there's a critical error within the migrate function. Part of the function transfers the mintership and timelock rights to the new contract. However, the following order of operations is used for this:</p> <ol style="list-style-type: none">1. addMinter (new address) [callable by timelock]2. setTimelock (new address) [callable by timelock]3. removeMinter (old address) [callable by timelock] <p>However, since the timelock was transferred in step 2, removeMinter fails and reverts. This is because the new address is now the timelock and the old one cannot call any functions like this anymore.</p> <p>The migrateVault function is thus broken.</p>
Recommendation	Consider inverting the two lines and adding test coverage to this function.
Resolution	 ACKNOWLEDGED <p>The team has indicated they were already aware of this from the start and have considered this a feature as it allows for their partners and users to be sure that the vault will never migrate.</p>

Severity

 MEDIUM SEVERITY

Description

It appears like the unshETH team has intentionally taken steps to prevent themselves from withdrawing any of the staked tokens out of the vault. Actions like migration, changing the oracle or zap are all timelocked.

However, through an advanced set of actions, a privileged exploit presently exists which allows the unshETH team to bypass this timelock.

To the best of our knowledge, the unshETH team was not aware of this advanced vector and immediately took steps to lock themselves out of this as soon as we disclosed it.

Bypassing the timelock is possible as adding new assets is possible. The team thought that adding new assets is not possible as the zapper needs to explicitly grant approval for these assets, which can only be done through a timelocked zapper upgrade.

However, the team did not realize that a bad asset can be added as a new vault asset which simply grants each user in existence infinite approval (e.g. never prevents transfers from occurring). This malicious asset bypasses the approval guards on the zap and can be freely deposited into the vault after it has been added.

The trick is then to assign an extremely high oracle value to this asset and iteratively minting unshETH by using it as collateral. This then allows the owner to fully drain all assets within the LSDVault.

Recommendation

Consider guarding the addLSD function behind the timelock as well — this function should not be callable instantly.

Alternatively, the owner address can be replaced by a timelock contract, which is owned by the multi-signature wallet. This way, the addLSD function cannot be called instantly either.

Resolution



The client has updated the Darknet implementation so that no new oracles can be added without going through the timelock. This effectively prevents the exploit and only makes it possible through undergoing the multi-day timelock.

Note that the client was not aware of this vector until we disclosed it. After disclosure, they immediately took steps to mitigate it, even before our audit was finalized. By the time we had finished the audit, it was already addressed.



Issue #12**LSD array logic is flawed, allowing for the first LSD to be added and enumerability to be more limited than it needs to be****Severity** MEDIUM SEVERITY**Location**Line 200-201

```
function addLSD(address _lsd) public onlyOwner
onlyWhenPaused {
    require(lsdIndex[_lsd] == 0, "Lsd has already been
added");
```

Description

The first LSD can be added twice due to a misconfigured requirement. This might seem innocuous at first but if the same LSD is added twice, it would cause critical vulnerabilities within the deposit and exit functions. The exit function would be especially problematic as it would grant the same payout twice for that token, draining the token balance over time.

The use of this array and mapping combination is quite restrictive as it is verbose and a length function is also not present.

Finally, setLSDConfigs can be called before the LSD is added.

Recommendation

Consider removing the lsdIndex, supportedLSDs and isEnabled variables and replacing them with an EnumerableSet supportedLSDs variable. Consider adding getter functions for this set. The isEnabled boolean should be incorporated in the LSDConfig instead.

```
struct LSDConfig {
    uint16 targetWeightBps;
    uint16 weightCapBps;
    uint216 absoluteCapEth;
    bool enabled;
}
```

It should be noted that we adjusted the type sizes for them to fit in a single storage slot which saves significant amounts of gas, as described in the gas optimization section of this audit.

Consider adding a requirement that the LSD is added within setLSDConfigs.

Resolution

 PARTIALLY RESOLVED

The client has taken note of this issue and acknowledged that it could be potentially abused to double withdraw the vault value without having to go through the timelock. They take this seriously and have added another party to their timelock to ensure that this will not happen. They have also indicated that once they do redeploy a new vault, they will definitely address this vector.



Issue #13**Lack of cross-contract reentrancy checks can lead to serious side-effects due to the usage of multiple contracts****Severity** MEDIUM SEVERITY**Description**

The AMM contract and LSD vault each have reentrancy guards on their functions. However, as these guards are isolated to each respective contract, one can reenter from the AMMs function into the LSDs functions and vice-versa.

A critical example (but not necessarily the only example) would be reentrancy on the following code-section within the AMM:

vdAMM::440-444

```
//Transfer amountIn from vdAMM to the vault
```

```
TransferHelper.safeTransfer(lsdIn, vaultAddress, amountIn);
```

```
//Transfer lsdOut from vault to vdAMM
```

```
TransferHelper.safeTransferFrom(lsdOut, vaultAddress,  
address(this), lsdAmountOutFromVault);
```

If a reentrancy occurs on the first transfer, the vault would have more assets than it actually should have at that time. In this case, an exit call would drain more assets than is adequate, potentially draining the vault.

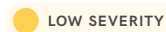
This issue has only been marked as medium severity as right now it is not exploitable as to the best of our knowledge none of the listed assets permit reentrancy. This could change as new assets are added.

Recommendation

Consider either sticking to checks-effects-interactions throughout the codebase, adding a global reentrancy lock or never listing anything which can potentially allow for reentrancy.

Resolution RESOLVED

The client has indicated that they will be diligent in not adding such tokens. Users should be careful and monitor upgrades to any and all collateral LSDs as some of these are upgradeable and may introduce new risk vectors. No changes were made.

Severity**Description**

Steps have been taken to protect the LSDVault against a hacked LSD. E.g. by restricting the percentage of the vault value which can be made up of any specific LSD.

By doing so, if an LSD gets hacked and its value becomes zero, the vault is only exposed to a limited percentage, effectively diversifying the underlying portfolio.

However, certain LSDs might get hacked in a way where the hacker is able to bypass these limits.

The most notable example is a hack where the hacker can freely adjust the "rate" function on the LSD. This is possible in upgradeable LSD contracts where the upgrade admin is compromised.

In this case, the hacker can potentially adjust the rate function in a way where the limits are fully bypassed as the rate is set extremely low during the cap checks, while it is set higher again during the actual deposit calculation.

Finally, `exit` is presently not safeguarded if the `transfer` function of a single LSD ever breaks. In this case, nobody can withdraw underlying tokens due to a single LSD breaking.

Recommendation

Consider rethinking the weight limiting mechanism to address fully compromised LSDs. The following extra safeguards might make sense:

- Hourly/daily limits on the amount of unshETH can be minted for any specific LSD.
- A non-decreasing check on the rate which reverts deposits and exits if detected (should be disableable/configurable by the team as some LSDs might allow for decreasing). This might not be an ideal safeguard as there might be genuine reasons to decrease the rate like slashing.
- Other creative extra layers of defense.

Consider either having a working migration function or adding a way to disable LSDs properly as well (this disabling should of course be timelocked).

Resolution

ACKNOWLEDGED

The team acknowledged this risk but also indicated its very unlikely to occur. Nevertheless, they do plan to add safeguards within the Darknet implementation to fully prevent this over time.

Issue #15

Shangai time safeguard is futile as it can be called repeatedly

Severity

LOW SEVERITY

Location

Line 523

```
require(_newTime < shanghaiTime + 4 weeks, "Cannot extend  
more than 4 weeks" );
```

Description

When updating shanghaiTime, it can be postponed by at most four weeks at a time. This is rather futile as this function can be called repeatedly.

Recommendation



This issue will be resolved on the note that this safeguard is only to prevent human error, as it does suffice in this case. If there are any other security business reasons, a stricter requirement should be added.

Additionally, the requirement should be adjusted to using \leq to match the description of the error. The following requirement should be adjusted respectively, as its semantically wrong as well.

Resolution

PARTIALLY RESOLVED

The team indicated that this code is in the past so it is no longer relevant. No changes were made.

Issue #16	Division is done before multiplication within exit, causing unnecessary precision loss
Severity	 INFORMATIONAL
Location	<u>Lines 445 and 450</u> <pre>uint256 shareOfUnsheth = 1e18*amount/ IERC20(unshETHAddress).totalSupply(); uint256 amountPerLsd = (shareOfUnsheth-fee)*lsdBalance/1e18;</pre>
Description	Division is performed before multiplication within the exit function which can cause loss of precision and less than expected receipt amounts for small withdrawals or when the total supply gets extremely large.
Recommendation	Consider doing multiplication before division instead. The fee logic would need to be adjusted as well.
Resolution	 ACKNOWLEDGED



DescriptionLine 6

```
import "communal/SafeERC20.sol";
```

This is a duplicate with the already imported TransferHelper.
Consider removing it and sticking to a single library.

Line 8

```
//import "forge-std/console.sol"
```

Consider removing this line.

Line 20

```
interface IunshETH {
```

Consider extending IERC20 to make the interface more complete.

Line 46

```
// address public admin;
```

Consider removing this line.

Line 48

```
address public constant v1VaultAddress =  
address(0xE76Ffee8722c21b390eebe71b67D95602f58237F);
```

The address wrapping is unnecessary.

Line 49

```
address public unshETHAddress;
```

Consider casting this to IunshETH to avoid casting it repeatedly
throughout the contract and make the contract more readable.

Line 53

```
address public darknetAddress;
```

Consider casting this to IDarknet to avoid casting it repeatedly throughout the contract and make the contract more readable.

Lines 84 and 129

```
bool public depositsPaused;  
.  
.  
.  
depositsPaused = true;
```

These two lines can be simplified by simply setting line 84 to true instead.

Line 98

```
event UnshethAddressSet(address unshethAddress);
```

This event is unused and can be removed.

Lines 134-136


```
useWeightCaps = false;  
useAbsoluteCaps = false;  
includeV1VaultAssets = false;
```

These lines are redundant as the variables are already set to false during initialization. We recommend removing these.

isLsdEnabled, getLsdIndex, remainingRoomToTargetInEthTerms, createTimeLockProposal and cancelTimeLockProposal can be marked as external to make the contract more readable.

Recommendation	Consider fixing the typographical issues.
-----------------------	-------------------------------------------

Resolution

 ACKNOWLEDGED

The LSDVault cannot be redeployed.

DescriptionLine 49

```
address public unshETHAddress;
```

Consider marking this variable as `immutable` which would save significant gas.

Line 58

```
struct LSDConfig {
```

This struct uses three storage slots while it only requires one. Additionally, the `enabled` boolean can be incorporated to reduce the usage of 4 storage slots to a single one. This would save significant amounts of gas:

```
struct LSDConfig {  
    uint16 targetWeightBps;  
    uint16 weightCapBps;  
    uint216 absoluteCapEth;  
    bool enabled;  
}
```

Line 73

```
struct TimelockProposal {
```

This struct uses two storage slots while it only requires one. Consider optimizing the types by setting the `unlockTime` to `uint96` which would save gas. This is not a critical portion of code so this is purely provided informationally.

Lines 182, 187, 213, 238, 243, 262, 437, 492, 502, 503, 526, 532, 541 and 552

```
emit UnshethZapAddressSet(unshethZapAddress);
emit AdminSet(admin);
emit LSDConfigSet(_lsd, lsdConfigs[_lsd]);
emit WeightCapsToggled(useWeightCaps);
emit AbsoluteCapsToggled(useAbsoluteCaps);
emit DepositPauseToggled(depositsPaused);
emit RedeemFeeUpdated(redeemFee);
IunshETH unshETH = IunshETH(unshETHAddress);
if(swapperAddress != address(0)) {
    _setApprovals(swapperAddress, 0);
    emit ShanghaiTimeUpdated(shanghaiTime);
    emit DepositPauseToggled(depositsPaused);
    emit WithdrawalsPaused(withdrawalUnpauseTime);
    emit VdAmmDisabled(swapperAddress);
```

Lines 224, 253, 326, 448, 488 and 514

```
for(uint256 i=0; i<supportedLSDs.length; i=unchkIncr(i)) {
```

All of these lines use more gas than necessary as they access contract storage on each iteration. We recommend caching the array length in a locally scoped variable to save significant gas in all of these locations.

Line 225

```
enableLSD(supportedLSDs[i]);
```

This function calls an external function with redundant modifiers, causing gas to be wasted due to repetitive checks. This is a non-critical code-section so this has been provided informationally.

Lines 254, 256, 327 and 328

```
uint256 targetWeightBps =
lsdConfigs[supportedLSDs[i]].targetWeightBps;
require(isEnabled[supportedLSDs[i]], "Need to enable LSD
with non-zero target weight");
uint256 rate = getEthConversionRate(supportedLSDs[i]);
underlyingBalance += rate
*IERC20(supportedLSDs[i]).balanceOf(address(this))/1e18;
```

supportedLSDs[i] can be cached to save gas.

Line 410 and 412

```
uint256 rate = getEthConversionRate(1sd);  
return  
remainingRoomToCap(1sd,marginalDeposit)*getEthConversionRate  
(1sd)/1e18;
```

getEthConversionRate(1sd) can be cached. This also reduces the risk of side-effects as the function could theoretically return a different value for each call.

Within balanceInUnderlying, darknetAddress is fetched from storage many times. It would save significant gas to only fetch it once.

Recommendation	Consider implementing the gas optimizations mentioned above.
-----------------------	--------------------------------------------------------------

Resolution	
-------------------	--

 ACKNOWLEDGED

The LSDVault cannot be redeployed.



2.5 RenouncedOwner

RenouncedOwner is a trivial and empty contract with a single function: to call `acceptOwnership` on any contract.

Once ownership is accepted from a contract, that ownership is effectively renounced. This is because this contract cannot call any owner functions on the contract except for `acceptOwnership` which would revert.



2.5.1 Issues & Recommendations

Issue #19	Typographical error
Severity	INFORMATIONAL
Location	<u>Lines 9-10</u> event OwnershipRenounced(address ownedContract); function acceptAndRenounce(address _contract) external {
Description	The contract address can be provided as the IOwned address to avoid casting it later on and making the type more explicit.
Recommendation	Consider fixing the typographical errors.
Resolution	ACKNOWLEDGED



2.6 unshETHZap



unshETHZap is the core contract used by users to deposit LSDs into the unshETH vault in return for unshETH tokens. It contains various functions which allow users to enter into the vault with USDT, ETH or any one of the LSDs. The zipper can take a variety of routes to deposit into the vault.



A deposit fee may be levied based on the dynamic fee from deviating away from the weight target.



2.6.1 Issues & Recommendations

Issue #20	The external <code>deposit_stEth</code> function is broken as it does not pull in any <code>stETH</code> from the user
Severity	 MEDIUM SEVERITY
Location	<p>Lines 153-160</p> <pre>function deposit_stEth(uint256 stETHAmount) external nonReentrant { // Deposit stETH into wstETH IWStETH(wstETHAddress).wrap(stETHAmount); // Get the wrapped balance uint256 wstETHAmount = IERC20(wstETHAddress).balanceOf(address(this)); // Deposit into lsd vault _deposit_lsd(wstETHAddress, wstETHAmount); }</pre>
Description	<p>The contract contains a function which wraps Lido staked ETH into wstETH and deposits it into the vault.</p> <p>However, this function misses a critical functionality as it does not transfer in the staked Ethereum from the user, causing this function to revert unless the contract already contains stETH (which would likely be stolen due to MEV).</p>
Recommendation	Consider adding a line which transfers in the staked ETH.
Resolution	 PARTIALLY RESOLVED
	The client indicated they do not use this function and will remove it in a further iteration. No change has been made.

Issue #21 deposit_1sd lacks a reentrancy guard	
Severity	 LOW SEVERITY
Description	The deposit_1sd function lacks a reentrancy guard. This is inconsistent with the rest of the external functions and could cause issues if a reentrancy token is ever added.
Recommendation	Consider adding a reentrancy guard.
Resolution	 PARTIALLY RESOLVED The client indicated they will not support any reentrancy assets. No change was made.

Issue #22 Some interfaces do not match the actual function interface which may cause issues with future compiler versions	
Severity	 LOW SEVERITY
Location	<u>Lines 25, 29 and 50</u> <pre>function submitAndDeposit(address recipient) payable external; function deposit(uint256 assets, address receiver) external; function wrap(uint256 _stETHAmount) external;</pre>
Description	All of these functions return a number. However, the interface does not reflect this. Even though the compiler might check the return size length, it might in future versions (as it does when you add a return element).
Recommendation	Consider updating these interfaces to the correct ones.
Resolution	 ACKNOWLEDGED

Severity

 INFORMATIONAL

Description

Line 199

```
if(depositFee > 0) {
```

It would be more correct and optimal to check for unshETHFee instead.

Lines 277, 301 and 377

```
deadline: block.timestamp + 3600,
```

The time addition is unnecessary here given that transactions execute in a fixed timestamp.

Recommendation

Consider implementing the gas optimizations mentioned above.

Resolution

 ACKNOWLEDGED

DescriptionLine 7

```
import "communal/TransferHelper.sol";
```

This appears to be a duplicate of the SafeERC20 import. We believe it's sufficient to only use the latter and delete the TransferHelper.

Line 12

```
function depositNoCapCheck(address lsd, uint256 amount)
external;
```

Line 32

```
interface RETH {
```

The above are unused and can be removed.

Line 57

```
address public constant wstETHAddress =
0x7f39C581F595B53c5cb19bD0b3f8dA6c935E2Ca0;
```

This can be retyped to IWStETH to avoid casting later on.

Line 58

```
address public constant frxETHMinterAddress =
0xbAFA44EFE7901E04E39Dad13167D089C559c1138;
```

This can be retyped to FRXETH to avoid casting later on.

Line 60

```
address public constant sfrxETHAddress =
0xac3E018457B222d93114458476f3E3416Abbe38F;
```

This can be retyped to SFRXETH to avoid casting later on.

Line 68

```
address public constant rocketSettingsAddress =
0x781693a15E1fA7c743A299f4F0242cdF5489A0D9;
```

This can be retyped to IRocketSettings to avoid casting later on.

Line 70

```
address public immutable lsdVaultAddressV2;
```

This can be retyped to ILSDVaultV2 to avoid casting later on.

Line 71

```
address public immutable unshETHAddressV2;
```

This can be retyped to IunshETH to avoid casting later on.

Line 73

```
ISwapRouter public uniswapRouterV3 =  
ISwapRouter(address(0xE592427A0AEce92De3Edee1F18E0157C058615  
64));
```

The address cast can be removed from this line.

Line 76

```
address public vdAmmAddress;
```


This can be retyped to IVdAmm to avoid casting later on.

Some of the return statements contain unnecessary brackets.

Several token definitions can be retyped to IERC20.

Recommendation	Consider fixing the typographical issues.
-----------------------	-------------------------------------------

Resolution

 ACKNOWLEDGED

2.7 vdAMM

vdAMM is a highly efficient interface for users to swap the underlying vault LSDs. It allows for near slippage free swaps between any of the supported assets.

Instead of working with an IL curve, vdAMM is able to swap without direct impermanent loss as it uses the oracle rates of the LSDs for the swap rate. For every swap, two fees are levied: a constant fee and a dynamic fee. Both of these fees can be configured freely though the constant base fee can be configured up to 0.1% while the maximum dynamic fee can reach up to 2%. A portion of the fees are sent to the governance while another portion of the fees are sent to the vault, increasing its value. This share can be configured. At most 50% of the fees can be kept by the unshETH team.

The dynamic fee is based on how far the swap takes either of the tokens away from the target weight. If the tokens move closer to their target weight, no dynamic fee is levied. If they however move away from the target, a dynamic fee is levied based on the distance the fee is away from the target, relative to the target (eg. 20% away from the target). This fee grows exponentially with the distance away from the target. An extra small fee is levied if people swap to WETH as it is expected that these swaps are more beneficial to the users given that exits take time.








2.7.1 Privileged Functions

- `setBaseFee [owner]`
- `setDynamicFeeSlopes [owner]`
- `setUnshethFeeShare [owner]`
- `setInstantRedemptionFee [owner]`
- `toggleDepositFee [owner]`
- `togglePaused [owner]`
- `withdrawTokens [owner]`
- `withdrawStuckEth [owner]`
- `updateDarknetAddress [owner]`
- `approveNewLsd [owner]`
- `nominateNewOwner [owner]`
- `acceptOwnership [nominated owner]`



2.7.2 Issues & Recommendations

Issue #25	1sds lacks a length function, making it tedious for the frontend to explore the array efficiently
Severity	 LOW SEVERITY
Description	<p><u>Line 56</u></p> <pre>address[] public 1sds;</pre> <p>The 1sds array has a public index function. However, there is presently no way for the frontend to figure out what the size of this array is (e.g. which indices are permissible) without trial-and-error.</p> <p><u>Line 62</u></p> <pre>struct AmmFee {</pre> <p>This struct can be removed as it appears to be unused.</p>
Recommendation	Consider adding a length function.
Resolution	 ACKNOWLEDGED
	The team is aware of this issue and agrees a length function would be useful. It will as such be added in a future upgrade.
Issue #26	withdrawStuckEth might not work with certain multi-signature wallets and timelocked owners
Severity	 LOW SEVERITY
Description	<p>withdrawStuckEth requires an owner that has a receive function. However, not all multi-signature wallets and timelock contracts have this. An ownership transfer would be required whenever this function needs to be used.</p>
Recommendation	Consider adding a to parameter to allow the owner to set the recipient of the ETH.
Resolution	 ACKNOWLEDGED
	The team plans to add this in a future upgrade.

Issue #27**vdAMM lacks edge-case safeguards****Severity** LOW SEVERITY**Description**

vdAMM has full approval from the LSDVault to access and transfer tokens freely from the vault at any point in time and in any quantity.

Though this appears to be consistent with the security parameters of the overall system within the current version of the AMM, this poses an unnecessary risk to the vault, which might eventually cause it to be drained simply due to suboptimal risk management.

If vdAMM is ever upgraded to a version with a bug within it, this bug might be exploited to drain the vault through the approvals. In our opinion, this is an unnecessary risk.


Recommendation

Consider doing the actual transfer logic within the vault instead of giving the zap vdAMM approval. The zap can then do a constant sum invariant check to ensure that the reserves never decrease in ETH terms. This would require a zap upgrade.

Resolution ACKNOWLEDGED

The client indicated that unshETH v3 which is currently being developed handles input validation checks in the LSDVault. The current system as stated is designed in this modular manner and the LSDVault is not designed to be upgradeable.

Severity

 INFORMATIONAL

Description

Line 58

```
ILSDVault public vault;
```

This value can be marked as `immutable` to save gas whenever its used.

Line 68

```
struct AMMFeeConfig {
```

The values within this struct can be significantly optimized to fit in just one or two storage slots.

Lines 179, 190, 208

```
emit DepositFeeToggled(depositFeeEnabled);
```

```
emit PauseToggled(ammPaused);
```

```
emit DarknetAddressUpdated(darknetAddress);
```

The emitted variable has already been fetched from storage, consider caching it first.

Recommendation

Consider implementing the gas optimizations mentioned above.

Resolution

 ACKNOWLEDGED

The client indicates these will be fixed in future versions.



DescriptionLine 7

```
import "communal/TransferHelper.sol";
```

This import appears to serve the same functionality as SafeERC20 and can therefore be removed in favor of SafeERC20.

Line 9

```
//import "forge-std/console.sol";
```

This line can be removed.

Line 51

```
address public constant wethAddress =  
0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2;
```

This address could be cast to IWETH to avoid casting it later on.

Line 54

```
address public darknetAddress;
```

This address should be cast as IDarknet to avoid casting it later on.

Line 93

```
event BaseFeeUpdated(uint256 _baseFeeBps);
```

This event is unused, consider using it within the setter function.

Line 119

```
ammPaused = true;
```

This line can be removed as the variable can be directly initialized on line 60.

Line 126

```
TransferHelper.safeApprove(unshethAddress, vaultAddress,  
type(uint256).max);
```

This line does not appear to serve any purpose, consider removing it.

Line 140

```
require(ammPaused == true, "AMM must be paused");
```

`== true` can be removed as the variable is already a boolean.

Line 163

```
require(_unshethFeeShareBps >= minUnshethFeeShareBps,  
"unshETH fee share must be greater than min")
```

The error seems to indicate that the requirement must use a strictly greater than statement. Consider adjusting the error message to indicate it can be greater than or equal to min.

Line 193

```
function withdrawTokens(address tokenAddress) external  
onlyOwner {
```

`tokenAddress` can be cast as `IERC20` to avoid casting it later on. Additionally, when `SafeERC20` is used, a cast is not required for the transfer.

Lines 240-241

```
require(vault.isEnabled(lsdIn), "lsdIn not enabled");  
require(vault.isEnabled(lsdOut), "lsdOut is not enabled");
```

These errors are inconsistent, consider being consistent in the grammar.

`unshethFeeShareBps` can be marked as `external`.

Recommendation	Consider fixing the typographical issues.
-----------------------	-------------------------------------------

Resolution

 ACKNOWLEDGED

The client indicates these will be fixed in future versions.

2.8 Owned

Owned is a dependency used within the unshETH ecosystem to manage the owner privilege of each contract. It differs from the traditional Ownable contract from OpenZeppelin in that the Owned contract adds another layer of protection by requiring the new owner to accept ownership on transfers of said ownership.



It is a near perfect fork of the Owned contract within Synthetix.

2.8.1 Privileged Functions

- `nominateNewOwner [owner]`
- `acceptOwnership [nominated owner]`



2.8.2 Issues & Recommendations

Issue #30	The zero address can claim ownership
Severity	 INFORMATIONAL
Location	<u>Lines 20-21</u> <pre>function acceptOwnership() external { require(msg.sender == nominatedOwner, "You must be nominated before you can accept ownership");</pre>
Description	The default value for nominatedOwner is address(0). This means if the zero address private key is ever discovered by a third party (eg. post quantum computing), this entity can claim ownership of all unshETH contracts if no owner is nominated at that time.
Recommendation	Consider adding a non-zero requirement on the address.
Resolution	 ACKNOWLEDGED The client has indicated this is forked from Frax and they would prefer not to make changes as such.

Severity

 INFORMATIONAL

Location

Lines 20-25

```
function acceptOwnership() external {  
    require(msg.sender == nominatedOwner, "You must be  
    nominated before you can accept ownership");  
    emit OwnerChanged(owner, nominatedOwner);  
    owner = nominatedOwner;  
    nominatedOwner = address(0);  
}
```

Description

nominatedOwner can be cached within this function to reduce the storage access count of that slot from 3 to 1. Alternatively, msg.sender can be used within the second and third mention of this variable, as it should have even lower gas cost than a memory access. This issue is informational as this function is in no way critical with regards to gas usage.



Recommendation

Consider implementing the gas optimization mentioned above.

Resolution

 ACKNOWLEDGED

The client has indicated this is forked from Frax and they would prefer not to make changes as such.

Issue #32	Typographical issue
Severity	 INFORMATIONAL
Location	<u>Line 9</u> constructor (address _owner) public {
Description	The public modifier is unnecessary and ignored. Consider removing it. This issue is also present within the reentrancy guard implementation which is not within scope.
Recommendation	Consider fixing the typographical issue.
Resolution	 ACKNOWLEDGED The client has indicated this is forked from Frax and they would prefer not to make changes as such.





PALADIN
BLOCKCHAIN SECURITY