Summary

Auditor: vnmrtz (Victor Martinez)

Client: Unhosted Wallet

Report Delivered: November, 2023

Protocol Summary

Protocol Name	Unhosted Wallet Modules
Language	Solidity
Codebase	Wallet Modules
Commit	a9469c0e3979ba3efc07c31d5be06e8512b60041
Previous Audits	No

About vnmrtz

Victor Martinez, or **vnmrtz**, is an independent smart contract security researcher. Having found numerous security vulnerabilities in various protocols, he does his best to contribute to the blockchain ecosystem and its protocols by putting time and effort into security research & reviews. Reach out on Twitter @vn_martinez_ or on Telegram @Vicrocvn.

Audit Summary

Unhosted engaged **vnmrtz** to review the security of its wallet modules. From the 6th of November to the 19th of November, vnmrtz reviewed the source code in scope. At the end, there were 18 issues identified. All findings have been recorded in the following report. Notice that the examined smart contracts are not resistant to internal exploit. For a detailed understanding of risk severity, source code vulnerability, and potential attack vectors, refer to the complete audit report below.

Vulnerability Summary

Severity	Total	Pending	Acknowledged	Par. resolved	Resolved
HIGH	2	0	0	0	2
MEDIUM	5	0	2	0	3
LOW	3	0	0	0	3
INFO	8	0	3	0	5

Audit Scope

ID	File Path
FACTORY	defi-strategies\contracts\StrategyFactory.sol
MODULE	defi-strategies\contracts\StrategyModule.sol
PROXY	defi-strategies\contracts\Proxy.sol
BASE_HANDLER	defi-strategies\contracts\handlers\BaseHandler.sol
AAVEV2_HANDLER	defi-strategies\contracts\handlers\aavev2\AaveV2H.sol
CALLBACK_HANDLER	defi-strategies\contracts\handlers\aavev2\ CallbackHandler.sol
DATATYPES	defi-strategies\contracts\handlers\aavev2\libraries\ DataTypes.sol
COMPOUND_HANDLER	defi-strategies\contracts\handlers\compoundv3\ CompoundV3H.sol
LIDO_HANDLER	defi-strategies\contracts\handlers\lido\ LidoH.sol
UNISWAPV3_HANDLER	defi-strategies\contracts\handlers\uniswapV3\ UniswapV3H.sol
BYTESLIB	defi-strategies\contracts\handlers\uniswapV3\libraries\ BytesLib.sol

Severity Classification

Severity	Classification
HIGH	Exploitable, causing loss/manipulation of assets or data.
MID	Risk of future exploits that may or may not impact the smart contract execution.
LOW	Minor code errors that may or may not impact the smart contract execution.
INF	No impact issues. Code improvement

Methodology

The auditing process pays special attention to the following considerations:

- **Testing** the smart contracts against both **common** and **uncommon attack vectors**.
- **Assessing the codebase to ensure compliance** with current **best practices** and industry **standards**.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross-referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- **Thorough line-by-line manual review** of the entire codebase by industry experts.

Findings and Resolutions

ID	Description	Severity	Status
1	Malicious beneficiary can DOS strategies	HIGH	Fixed
2	Cross contract transaction replay	HIGH	Fixed
3	A malicious handler could modify enabled modules through selfcall	MID	Acknowledged
4	Missing access control checks on flash loan callback contract	MID	Fixed
5	Avoid using a pricefeed oracle to fetch gas price	MID	Fixed
6	Transaction can be replayed after hard forks	MID	Acknowledged
7	Solidity version not supported on some chains	MID	Fixed
8	Do not hardcode contract addresses	LOW	Fixed
9	Include uniswap v3 swap deadline in signed tx params	LOW	Fixed
10	FACTORY `baseImplementation` address cannot be updated	LOW	Fixed
11	Missing admin module approval	INFO	Acknowledged
12	Consider implementing withdraw on HLIDO handler	INFO	Acknowledged
13	High dev fees on gas-used during handler execution	INFO	Fixed
14	Use OZ's forceApprove instead of custom `_tokenApprove`	INFO	Fixed
15	Consider implementing an AAVE-V3 handler	INFO	Acknowledged
16	Handler naming conventions do not match	INFO	Fixed
17	Use custom errors on _requireMsg	GAS	Fixed
18	Optimize for loops across the codebase	GAS	Fixed

1 | Malicious beneficiary can DOS strategies

Severity	Category	Status
HIGH	Design error	**Fixed**

Description of the issue

The existing beneficiary fee mechanism in **MODULE** transfers a proportion of the gas utilized within the strategy to the deploying developer. The drawback of this "push" model is the introduction of vulnerabilities that could potentially render the strategy susceptible to a Denial-of-Service (DoS) state.

A malicious developer could exploit this by designating a malicious contract as the beneficiary. This malicious contract, exemplified by the `Beneficiary` contract below, executes a revert upon receiving Ether, thereby enabling the malicious developer to obstruct the execution of the strategy at their discretion.

```
contract Beneficiary() {
    receive() {
        revert("");
    }
}
```

Consequently, any invocation of the `execStrategy` function in this scenario would result in a revert, impeding the normal operation of the strategy.

Recommendation

It is recommended to transition from the current push mechanism to a push-pull model within the existing system. In the proposed push-pull model, beneficiary fees are accumulated without immediate transfer, affording the beneficiary the ability to claim these fees through a designated function at their discretion.

Resolution

Commit: f1af1bbd08e230af108be90667a90127c52046b4

Team Response: Resolved by introducing a claim mechanism for the beneficiary address and decoupling the process of sending fees from the sender to the userSA. This separation ensures that executing the strategy is independent of the userSA needing to execute it for paying the developer fee.

2 | Cross contract transaction replay

Severity	Category	Status
HIGH	Logic error	**Fixed**

Description of the issue

The **MODULE** component incorporates EIP-1271 signature verification to confirm that the transaction

signer is the owner of the smart account. This verification precedes the execution of handler code, preventing arbitrary users from utilizing the module as an entry point for interacting with the owner's smart account.

The signature itself encompasses various elements, including the smart account, transaction data, a unique nonce, EXECUTE_STRATEGY_TYPEHASH, handler, transaction value, and the domain separator. According to the EIP-712 specification, the domain separator is calculated as the hash of DOMAIN_SEPARATOR_TYPEHASH, nameHash, versionHash, CHAINID, and the address of the verifying contract.

A potential issue arises from the fact that, in this system, the **MODULE** is the entity constructing the transaction hash for verification, while the smart account is responsible for the verification itself. Consequently, the address of the **MODULE** that constructs the hash should be incorporated into the **DOMAIN SEPARATOR**. Failure to include this address could permit another **MODULE**, with a different beneficiary but the same handler (thus deployed to a distinct address), to replay all transactions processed by the original one.

Despite users having control over adding modules to the wallet, if they add two modules targeting the same handler, all transactions from one module could be replayed on the second one. To mitigate this, it is recommended to include the address of the **MODULE** in the **DOMAIN SEPARATOR**, ensuring proper differentiation between modules and preventing replay attacks.

Recommendation

Follow the EIP-712 guidelines for domain separator building and include both smart account and **MODULE** addresses in the signed data.

Resolution

Commit: 4cf58a2fd1de7d6bf6b5b57001507c2379f19b83

Team Response: Resolved by updating the verifying contract to use the module address for constructing the domain separator and incorporating the smart account address as a salt. This modification ensures that each handler and module has a specific identification, mitigating the risk of replay attacks.

3 | A malicious handler could modify enabled modules calling the smart account through a selfcall

Severity	Category	Status
MID	Missing check	**Acknowledged**

Description of the issue

The issue arises from a handler being able to bypass the enableModule() function access control checks, which increments the enabledModuleCount of a smart account, and register a new module without being the smart account owner. Given that the enableModule() function allows calls from the entrypoint or the wallet itself, two potential approaches can address this concern:

Approach 1: Restricting Module Management to Smart Account Owner

Make sure only the owner of the smart account and not the smart account itself can add / remove a module. Consider overriding and changing the checks on the following SmartAccount.sol functions

```
function enableModule(address module) external virtual override {
    _requireFromEntryPointOrSelf();
    _enableModule(module);
}

function setupAndEnableModule(
       address setupContract,
       bytes memory setupData
) external virtual override returns (address) {
       _requireFromEntryPointOrSelf();
       return _setupAndEnableModule(setupContract, setupData);
}

function disableModule(address prevModule, address module) public virtual {
       _requireFromEntryPointOrSelf();
       _disableModule(prevModule, module);
}
```

Approach 2:

To ensure that a malicious handler cannot add a non-factory deployed module to the smart account, which could grant them unlimited future power, a check is introduced before calling `execTransactionFromModuleReturnData`. This check compares the hash of the modules before the transaction with the hash after the call:

```
// Approach 2: Hash Check
(address[] memory modulesBefore,) = safe.getModulesPaginated(SENTINEL_OWNERS,
enabledModuleCount);
_existingModulesHash = keccak256(abi.encode(modulesBefore));

// ... (execute transaction)

(address[] memory modulesAfter,) = safe.getModulesPaginated(SENTINEL_OWNERS,
enabledModuleCount + 1);
if (keccak256(abi.encode(modulesAfter)) != _existingModulesHash) {
       revert SignersCannotChangeModules();
}
```

Recommendation

In summary, it is strongly advised to implement measures that restrict the ability of handlers to register new modules in the smart account system.

Resolution

Team Response: Acknowledged.

4 | Missing access control checks on flash loan callback contract

Severity	Category	Status
MID	Missing checks	**Fixed**

Description of the issue

The **CALLBACK_HANDLER** component, functioning as a blueprint for custom implementations, should come with two default checks by default: msg.sender == AAVEv2 pool and initiator address == smart wallet. These checks, while fundamental for the secure operation of the smart account, might be overlooked by developers if not included in the template. The absence of these checks in custom implementations built upon this contract could potentially jeopardize the integrity of the smart account.

Recommendation

Add the following checks to the implementation of **CALLBACK_HANDLER's** `executeOperation`:

```
function executeOperation(
        address[] calldata,
        uint256[] calldata,
        uint256[] calldata,
        address,
        bytes calldata
) external virtual returns (bool) {
        _requireMsg(
            msg.sender ==
                ILendingPoolAddressesProviderV2(provider).getLendingPool(),
            "executeOperation",
            "invalid caller"
        );

        _requireMsg(
            initiator == address(this),
            "executeOperation",
            "not initiated by the proxy"
        );
        // execute logic on flashloan receive
        return true;
}
```

Resolution

Commit: c4a5355900bde545d11e0e15850077601cacf9ba

Team Response: Resolved by incorporating the check `msg.sender == initiator` into the executeOperation function. This adjustment is made considering that the callback handler is invoked by the wallet and not directly by the AAVE pool, ensuring the necessary security verification.

5 | Avoid using a pricefeed oracle to fetch gas price

Severity	Category	Status
MID	Business logic error	**Fixed**

Description of the issue

The **MODULE** component calculates the gas price by utilizing a Chainlink price feed. Generally using oracles is not advisable when alternative methods are available to fetch the data. Since gas price is being calculated in ETH, a more straightforward approach would be to rely on `tx.gasprice` for the gas price calculation in ETH. This avoids the use of an external oracle, which could introduce various attack vectors and vulnerabilities, including the risk of stale prices specially on L2s.

Recommendation

Opt for simplicity and gas efficiency by utilizing `tx.gasprice` for gas price retrieval, as opposed to the more complex alternative of fetching data from an external oracle such as `AggregatorV3Interface(_gasFeed).latestRoundData()`. This not only achieves notable gas savings but also eliminates an entire category of potential bugs, contributing to a more straightforward and reliable implementation.

Resolution

Commit: 5f1f0615c3596beeff0cb347fdd13bc36d31d478

Team Response: Resolved by substituting the gas feed with tx.gasprice for gas estimation in the module component.

6 | Transaction can be replayed after hardforks

Severity	Category	Status
MID	Business logic error	**Acknowledged**

Description of the issue

The **MODULE** contract initializes the `domainSeparator` (containing the chain ID) in the constructor and always uses this separator. However, this approach can pose issues in the event of a hard fork that alters the chain ID (e.g., Ethereum -> Ethereum Classic) after the contract deployment. All transactions will then be usable on both chains, although it is only intended for one of them. Furthermore, replaying transactions on different chains.

Recommendation

Consider using the OpenZeppelin's `EIP712` contract or caching the original chain ID and returning the cached `DOMAIN_SEPARATOR` if the chain id has not changed. This solution aligns with OpenZeppelin's approach, providing a gas-efficient resolution specifically required during hard forks when the chain ID undergoes modification.

Resolution

Team Response: Acknowledged for optimization reasons, the domain separator in the module

contract, being dependent on a salt and not immutable, prompts consideration. Checking the chainId every time for executing incurs additional gas usage.

7 | Solidity version not supported on some chains

Severity	Category	Status
MID	Design Error	**Fixed**

Description of the issue

Solidity version compatibility issues exist on certain chains due to the utilization of the PUSH0 instruction, which pushes the constant value 0 onto the stack. Notably, this opcode is not supported on various chains, including Arbitrum, and may pose challenges for projects compiled with Solidity versions equal to or greater than 0.8.20, where PUSH0 was introduced.

Recommendation

Since Unhosted Wallet is supposed to be deployed across a huge variety of chains, its essential to make sure it will be supported in all of them. To prevent unexpected issues, consider using a Solidity version lower than 0.8.20, such as 0.8.19, where the problematic opcode is not utilized. This adjustment helps ensure broader support and avoids potential complications across different blockchain networks.

Resolution

Commit: 3dccb1e97e0a623c52d65cdc3396291f8db8a761

Team Response: Resolved by updating the factory and implementation version to 0.8.19, while maintaining the rest of the handlers at version 0.8.20. This adjustment accommodates customization by developers, also ensuring compatibility with the latest version of the OpenZeppelin v5 contracts.

8 | Do not hardcode contract addresses

Severity	Category	Status
LOW	Design Error	**Fixed**

Description of the issue

The contract **UNISWAPV3_HANDLER** has the ROUTER address hard coded as a constant. This is not advisable since contracts do not always live at the same address across all chains, for example the Cello chain has the uniswap v3 router deployed on a different address than the rest of supported chains.

Not only for this contract but for future handler implementations it is advisable to pass addresses as params instead of hardcoding them.

Recommendation

Pass the router address as a parameter in the constructor and store it in an immutable variable like wrappedNativeTokenUniV3.

Resolution

Commit: db9ad8246f757226aefff9e2c9b0950f41ca7432

Team Response: Resolved by making the router address initializable in the constructor, providing flexibility and adaptability for the UNISWAPV3_HANDLER contract across different chains.

9 | Include uniswap v3 swap deadline in signed tx params

Severity	Category	Status
LOW	Business logic error	**Fixed**

Description of the issue

To facilitate swaps on Uniswap, one of the parameters required is the deadline, ensuring that the order won't be processed after a certain timestamp. Currently, the **UNISWAPV3_HANDLER** sets the deadline to the current block.timestamp every time it configures swap parameters. However, this approach renders the deadline useless since it is not included in the signature, making the swap transaction perpetually valid in the future since the deadline will always be set when executing the transaction.

Recommendation

To address this issue, it is recommended to enhance user control over the execution time frame of their swaps. This can be achieved by including a deadline parameter within the signed transaction data. This modification ensures that users can specify the deadline for their swaps and that it is a part of the transaction signature, providing more effective control over the execution time frame.

Resolution

Commit: 862c146b775a8e111e250d29b7afcc58cec2354a

Team Response: Resolved by modifying the functions to receive the deadline as parameters, thereby providing users with the ability to specify the deadline for their swaps and ensuring it is included in the transaction signature for effective control over the execution time frame.

10 | FACTORY baseImplementation address cannot be updated

Severity	Category	Status
LOW	Design Error	**Fixed**

Description of the issue

The **FACTORY** contract currently employs a basicImplementation variable during the creation of new **MODULE** clones. Despite the presence of versioning within **MODULE**, setting this variable as

immutable makes it impossible to deploy new clones pointing to the new version of the implementation of the **MODULE** contract.

Recommendation

To address this concern, it is advisable to make the `basicImplementation` variable mutable. Additionally, it is recommended to implement an admin or DAO-controlled setter function that allows for the dynamic adjustment of this variable.

Resolution

Commit: cdf8002064ead9e0f14f80c5896b0b81e32b6409

Team Response: Resolved by introducing an updating implementation mechanism that is access controlled by the owner, ensuring flexibility in adjusting the basicImplementation variable within the FACTORY contract.

11 | Missing admin module approval

Severity	Category	Status
INFO	Missing checks	**Acknowledged**

Description of the issue

As per the specification in the defi-strategies GitHub repository, the **FACTORY** component should feature an admin or ownership function dedicated to the approval and registration of new modules. Given the inherent trustless nature of the system, it is imperative to establish a mechanism enabling users to identify modules that have been deployed with verified or audited handlers, distinguishing them from modules characterized as entirely trustless.

A noteworthy observation is that, while the factory contract inherits OpenZeppelin's `Ownable` contract, the `onlyOwner` modifier remains unutilized within the codebase.

Recommendation

To address this, it is strongly recommended to incorporate an admin access control mechanism for the deployment of new modules. This can be achieved by either applying the onlyOwner modifier to `deployStrategyModule` functions or by implementing a registry featuring a mapping that whitelists forthcoming module addresses. Given the deterministic nature of these addresses, facilitated by the use of the create2 opcode, the latter approach would enhance transparency and facilitate a more robust control structure for module deployment.

Resolution

Team Response: Acknowledged.

12 | Consider implementing withdraw on HLIDO handler

Severity	Category	Status
INFO	Improvement	**Acknowledged**

Description of the issue

The **LIDO_HANDLER** currently only implements a function for depositing on LIDO, lacking support for withdrawals. Providing both deposit and withdrawal actions within the handler would enhance the user experience.

Recommendation

To address this, it is recommended to implement the LIDO withdrawal functionality in the **LIDO_HANDLER**. By extending the handler to support both deposit and withdrawal actions, users will have a more comprehensive and convenient interface for interacting with the LIDO protocol.

The LIDO docs explain how to implement this.

Resolution

Team Response: Acknowledged.

13 | High dev fees on gas-used during handler execution

Severity	Category	Status
INFO	Design Error	**Fixed**

Description of the issue

The **BASE_HANDLER** currently imposes a 50% fee factor on gas, which might be perceived as relatively high when utilizing a developer's strategy.

Recommendation

It is advisable to review the feasibility of reducing the beneficiary fee to a lower percentage, ideally around 20%. This adjustment aims to strike a balance between compensating the developer and ensuring a more reasonable fee structure for users employing a developer's strategy.

Resolution

Commit: 3b57909527a72b9cedb66bfea7f797523b949e35

Team Response: Fee factor reduced to 10%.

14 | Use OZ's `forceApprove` instead of custom `_tokenApprove`

Severity	Category	Status
INFO	Improvement	**Fixed**

Description of the issue

Use the cleaner and more efficient `forceApprove` function from Open Zeppelin's SafeERC20 lib instead of using the custom `_tokenApprove` and `_tokenApproveZero` functions.

Recommendation

Change `_tokenApprove` and `_tokenApproveZero` occurrences through the codebase to forceApprove.

Resolution

Commit: 4b8d54a23c3d879b11332e0042416cb213589da1

Team Response: Resolved by replacing instances of `_tokenApprove` and `_tokenApproveZero` with the more efficient forceApprove function from OpenZeppelin's SafeERC20 library.

15 | Consider implementing an AAVE-V3 handler

Severity	Category	Status
INFO	Improvement	**Acknowledged**

Description of the issue

Given that AAVE-v3 represents the latest version of the protocol, boasting increased liquidity, a larger user base, and a broader range of features, it is advisable to consider implementing an AAVE-v3 handler.

This implementation can attract more users to the wallet and aligns with the ongoing protocol migrations towards the v3 version. Supporting AAVE-v3 ensures compatibility with the latest features and contributes to the long-term viability of the wallet within the evolving AAVE ecosystem.

Resolution

Team Response: Acknowledged

16 | Handler naming conventions do not match

Severity	Category	Status
INFO	Typo	**Fixed**

Description of the issue

All handlers' `getContractName` functions implementations return the name of the protocol that they integrate preceded by the letter "H", except **UNISWAPV3_HANDLER** which returns "UniswapV3H".

Recommendation

Consider fixing the typo, in order to maintain consistency on the naming of all handlers.

Resolution

Commit: 84d87f3ff1534aa6b104271da8d5be99d65d4683

Team Response: Resolved by renaming and matching all handlers with their respective contract names to ensure consistency.

17 | Use custom errors on _requireMsg

Severity	Category	Status
GAS	Optimization	**Fixed**

Description of the issue

The **BASE_HANDLER** utilizes `_requireMsg` for error handling, which involves passing strings as parameters. This approach is considered inefficient, and there is also an absence of custom errors, potentially impacting gas efficiency.

Recommendation

To enhance gas efficiency, it is recommended to eliminate a level of abstraction by performing checks directly inside the functions instead of passing strings as parameters to `_requireMsg`. Consider refactoring the errors to custom errors too. This will save a significant amount of gas.

Resolution

Commit: aa0f73e70e4e92a595f5a5e544bea010ac2f67c5

Team Response: Resolved by incorporating custom error handling and removing the use of the `_requireMsg` function.

18 | Optimize for loops across the codebase

Severity	Category	Status
GAS	Optimization	**Fixed**

Description of the issue

Inside for loops across the codebase do not initialize `i` variables to 0, and consider performing increments inside an `unchecked` block as well. This reduces gas costs.

```
for (uint256 i; i< length;) {
	// ...
	unchecked{
		++i;
	}
}
```

Resolution

Commit: f95d54f8a3c4e204005d6feeade2e40266b1fbf2

Team Response: Resolved by enhancing loop gas efficiency.

DISCLAIMER

This report is not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any “product” or “asset” created by any team or project that contracts Víctor Martínez to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intended to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology. Note that blockchain technology and cryptographic assets present a high level of ongoing risk.

My position is that each company and individual are responsible for their own due diligence and continuous security. My goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze. Therefore, I do not guarantee the explicit security of the audited smart contract, regardless of the verdict.