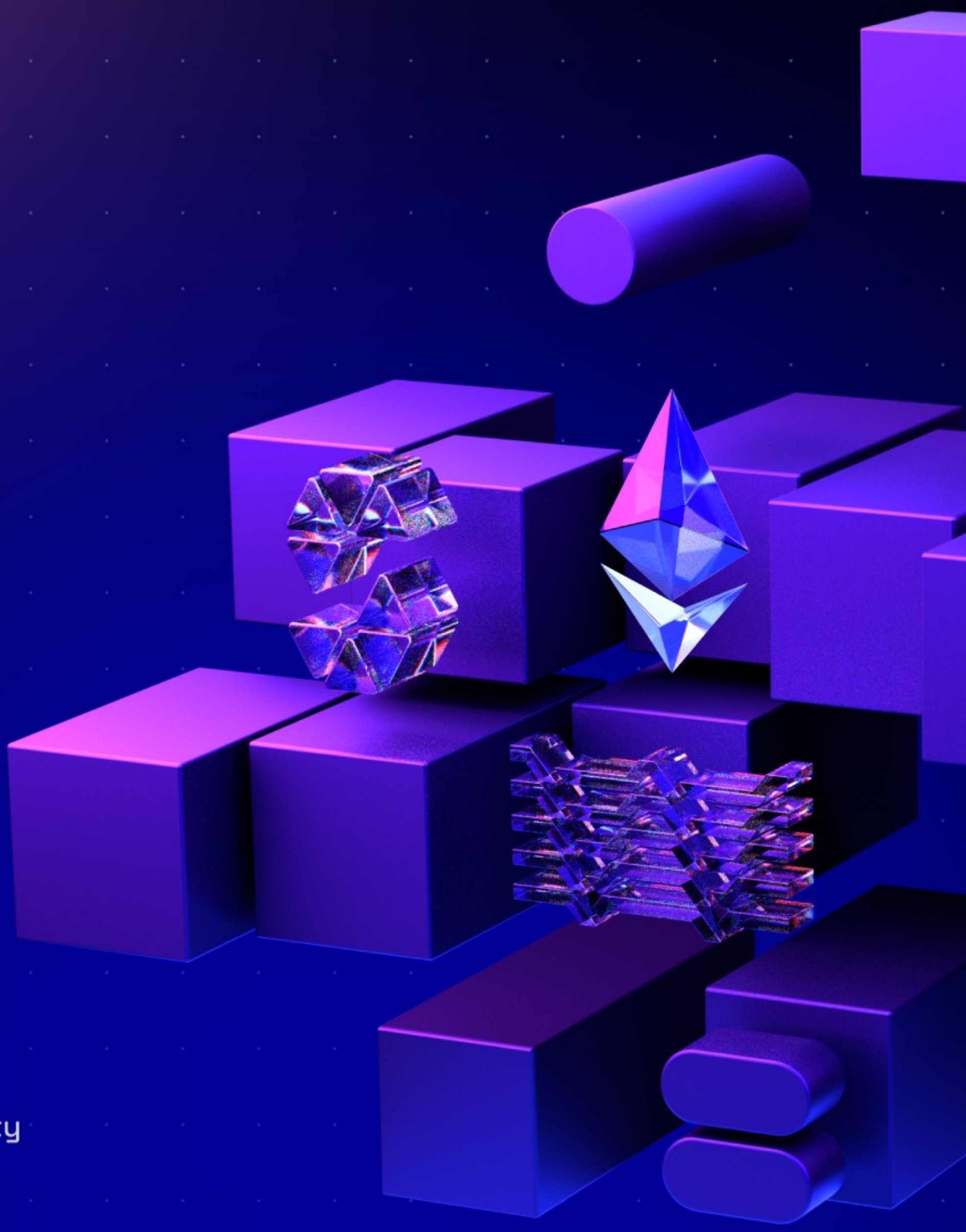
Vfat

Farm Strategies

24.6.2025



Ackee Blockchain Security

Contents

1. Document Revisions

1.0-draft	Draft Report	20.01.2025
1.1	Final Report	24.06.2025

2. Overview

This document presents our findings in reviewed contracts.

2.1. Ackee Blockchain Security

Ackee Blockchain Security is an in-house team of security researchers performing security audits focusing on manual code reviews with extensive fuzz testing for Ethereum and Solana. Ackee is trusted by top-tier organizations in web3, securing protocols including Lido, Safe, and Axelar.

We develop open-source security and developer tooling Wake for Ethereum and Trident for Solana, supported by grants from Coinbase and the Solana Foundation. Wake and Trident help auditors in the manual review process to discover hardly recognizable edge-case vulnerabilities.

Our team teaches about blockchain security at the Czech Technical University in Prague, led by our co-founder and CEO, Josef Gattermayer, Ph.D. As the official educational partners of the Solana Foundation, we run the School of Solana and the Solana Auditors Bootcamp.

Ackee’s mission is to build a stronger blockchain community by sharing our knowledge.

Ackee Blockchain a.s.

Rohanske nabrezi 717/4

186 00 Prague, Czech Republic

https://ackee.xyz

hello@ackee.xyz

2.2. Audit Methodology

1. **Verification of technical specification**

   The audit scope is confirmed with the client, and auditors are onboarded to the project. Provided documentation is reviewed and compared to the audited system.

2. **Tool-based analysis**

   A deep check with Solidity static analysis tool Wake in companion with Solidity (Wake) extension is performed, flagging potential vulnerabilities for further analysis early in the process.

3. **Manual code review**

   Auditors manually check the code line by line, identifying vulnerabilities and code quality issues. The main focus is on recognizing potential edge cases and project-specific risks.

4. **Local deployment and hacking**

   Contracts are deployed in a local Wake environment, where targeted attempts to exploit vulnerabilities are made. The contracts' resilience against various attack vectors is evaluated.

5. **Unit and fuzz testing**

   Unit tests are run to verify expected system behavior. Additional unit or fuzz tests may be written using Wake framework if any coverage gaps are identified. The goal is to verify the system's stability under real-world conditions and ensure robustness against both expected and unexpected inputs.

2.3. Finding Classification

A *Severity* rating of each finding is determined as a synthesis of two sub-ratings: *Impact* and *Likelihood*. It ranges from *Informational* to *Critical*.

If we have found a scenario in which an issue is exploitable, it will be assigned an impact rating of *High*, *Medium*, or *Low*, based on the direness of the consequences it has on the system. If we haven't found a way, or the issue is only exploitable given a change in *configuration* (system settings or parameters, such as deployment scripts, compiler configurations, using multi-signature wallets for owners, etc.) or given a change in the codebase, then it will be assigned an impact rating of *Warning* or *Info*.

Low to *High* impact issues also have a *Likelihood*, which measures the probability of exploitability during runtime.

The full definitions are as follows:

Severity

		Likelihood			
		High	**Medium**	**Low**	**N/A**
	High	Critical	High	Medium	-
	Medium	High	Medium	Low	-
Impact	**Low**	Medium	Low	Low	-
	Warning	-	-	-	Warning
	Info	-	-	-	Info

Table 1. Severity of findings

Impact

- **High** - Code that activates the issue will lead to undefined or catastrophic consequences for the system.
- **Medium** - Code that activates the issue will result in consequences of serious substance.
- **Low** - Code that activates the issue will have outcomes on the system that are either recoverable or don’t jeopardize its regular functioning.
- **Warning** - The issue cannot be exploited given the current code and/or *configuration*, but could be a security vulnerability if these were to change slightly. If we haven’t found a way to exploit the issue given the time constraints, it might be marked as a ”Warning” or higher, based on our best estimate of whether it is currently exploitable.
- **Info** - The issue is on the borderline between code quality and security. Examples include insufficient logging for critical operations. Another example is that the issue would be security-related if code or *configuration* was to change.

Likelihood

- **High** - The issue is exploitable by virtually anyone under virtually any circumstance.
- **Medium** - Exploiting the issue currently requires non-trivial preconditions.
- **Low** - Exploiting the issue requires strict preconditions.

2.4. Review Team

The following table lists all contributors to this report. For authors of the specific revision, see the “Revision team” section in the respective “Report revision” chapter.

Member’s Name	Position
Dmytro Khimchenko	Lead Auditor
Naoki Yoshida	Auditor
Josef Gattermayer, Ph.D.	Audit Supervisor

2.5. Disclaimer

We’ve put our best effort to find all vulnerabilities in the system, however our findings shouldn’t be considered as a complete list of all existing issues. The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them.

3. Executive Summary

Vfat is a yield aggregator, utilizing Sickle smart contract wallet for yield farming. It reduces complex operations such as entering/exiting positions, compounding, or rebalancing into single transactions.

Revision 1.0

Vfat engaged Ackee Blockchain Security to perform a security review of Vfat Farm Strategies with a total time donation of 12 engineering days in a period between May 19 and June 3, 2025, with Dmytro Khimchenko as the lead auditor.

The audit was performed on the commit `d85b2cd`[1] in the contracts repository and the scope was the following:

- `contracts/connectors/uniswap/UniswapV3Connector.sol`
- `contracts/connectors/velodrome/SlipstreamGaugeConnector.sol`
- `contracts/connectors/velodrome/SlipstreamNftConnector.sol`
- `contracts/connectors/velodrome/VelodromeGaugeRegistry.sol`
- `contracts/strategies/FarmStrategy.sol`
- `contracts/strategies/MultiFarmStrategy.sol`
- `contracts/strategies/NftFarmStrategy.sol`
- `contracts/strategies/SweepStrategy.sol`
- `contracts/libraries/ZapLib.sol`
- `contracts/libraries/NftZapLib.sol`.

The focus of this audit was to review the integration of the protocol with external protocols Uniswap and Velodrome.

We began our review using static analysis tools, including Wake. We then took a deep dive into the logic of the contracts. For testing and fuzzing, we have involved Wake testing framework. During the review, we paid special attention to:

- ensuring the arithmetic of the system is correct;
- detecting possible reentrancies in the code;
- integration with external protocols is correct;
- the code is consistent and follows the best practices;
- ensuring access controls are not too relaxed or too strict; and
- looking for common issues such as data validation.

Our review resulted in 12 findings, ranging from Info to Medium severity. The most severe one was M1, which is a front-running issue due to which users' funds can be stolen by a malicious actor; however, the likelihood of this is low. Most findings are related to violations of best practices, code quality issues, and the trust model.

Ackee Blockchain Security recommends Vfat:

- validate `approved` argument in `deposit` functions if it equals to `Sickle.approved`;
- make trust model more permissionless;
- use `prices` function instead of `getPoolPrice` for price calculation;
- read and review the complete audit report; and
- address all identified issues.

See Report Revision 1.0 for the system overview and trust model.

Revision 1.1

The review was done on the given commit `e5ff820`[2]. The scope of the fix review was limited to issues found in the previous revision and no other code changes were audited. 5 issues were fixed, 7 issues acknowledged by the client.

[1] full commit hash: `d85b2cd89cf5d6c76b92e3545b71ab0be71c08f5`

[2] full commit hash: `e5ff820b7218103404dc46286d8f1216e961b19b`

4. Findings Summary

The following section summarizes findings we identified during our review. Unless overridden for purposes of readability, each finding contains:

- *Description*
- *Exploit scenario* (if severity is low or higher)
- *Recommendation*
- *Fix* (if applicable).

Summary of findings:

Critical	High	Medium	Low	Warning	Info	Total
0	0	1	1	6	4	12

Table 2. Findings Count by Severity

Findings in detail:

Finding title	Severity	Reported	Status
M1: Front-run of `Sickle` deployment gives an opportunity for attacker to specify arbitrary `approved` and `referralCode` arguments	Medium	1.0	Acknowledged
L1: The charge fee can be bypassed for several functions	Low	1.0	Acknowledged
W1: Withdrawal of funds can be blocked by `Collector` contract by not accepting tokens	Warning	1.0	Acknowledged

Finding title	Severity	Reported	Status
W2: Connectors are single point of failure	Warning	1.0	Acknowledged
W3: Usage of function with `inplace=True` argument always fails in gauges that use NFTs	Warning	1.0	Acknowledged
W4: Missing `CompoundFor` fee calculation	Warning	1.0	Fixed
W5: `block.timestamp` is used for swap deadline	Warning	1.0	Acknowledged
W6: Incorrect price calculation	Warning	1.0	Acknowledged
I1: Missing NatSpec comments	Info	1.0	Fixed
I2: Potential incorrect fee calculation	Info	1.0	Fixed
I3: Unexpected revert in `increase` function	Info	1.0	Fixed
I4: Missing events in `MultiFarmStrategy`	Info	1.0	Fixed

Table 3. Table of Findings

Report Revision 1.0

Revision Team

Member's Name	Position
Dmytro Khimchenko	Lead Auditor
Naoki Yoshida	Auditor
Josef Gattermayer, Ph.D.	Audit Supervisor

System Overview

The protocol is a farming automation system where users receive unique `Sickle` instances deployed deterministically through the `SickleFactory` using CREATE2. Approved Automators can execute compound, harvest, and rebalance operations on users' behalf. Position settings are managed through the `PositionSettingsRegistry` for standard positions and the `NftSettingsRegistry` for NFT positions. The protocol features a fee system capped at 5% and specialized operation libraries. Protocol governance is controlled through a `SickleMultisig` contract with configurable thresholds and signer management. Integration with DeFi protocols is handled through an updatable Connector registry system. Connectors are responsible for interactions with external projects such as Uniswap and Velodrome, enabling depositing, withdrawing, swapping of funds, and creating yield positions

Trust Model

The protocol requires users to trust administrators who control critical parameters (fees, whitelists, Connector updates) and Automators who execute operations on their behalf. While users control their Sickle instances and position settings, the system maintains centralized control points. Trust risks are partially mitigated through hardcoded limits and multisig requirements; however, users must accept risks of centralized control and potential transaction manipulation by Automators who can control transaction timing.

Findings

The following section presents the list of findings discovered in this revision. For the complete list of all findings, Go back to Findings Summary

M1: Front-run of `Sickle` deployment gives an opportunity for attacker to specify arbitrary `approved` and `referralCode` arguments

Medium severity issue

Impact:	High	Likelihood:	Low
Target:	StrategyModule.sol, NftFarmStrategy.sol, FarmStrategy.sol	Type:	Front-running

Description

The protocol allows users to use `deposit` functions via `FarmStrategy` and `NftFarmStrategy` contracts without having the user's `Sickle` wallet deployed. If the `deposit` or `simpleDeposit` function is called by a user and they do not have a `Sickle` wallet deployed, the `FarmStrategy` or `NftFarmStrategy` will deploy the `Sickle` wallet for the user through the `getOrDeploySickle` function. Otherwise, the `FarmStrategy` or `NftFarmStrategy` will use the existing `Sickle` wallet connected to the user's address.

Listing 1. Excerpt from FarmStrategy.deposit

```
function deposit(
    DepositParams calldata params,
    PositionSettings calldata positionSettings,
    address[] calldata sweepTokens,
    address approved,
    bytes32 referralCode
) public payable {
    Sickle sickle = getOrDeploySickle(msg.sender, approved, referralCode);
```

Listing 2. Excerpt from FarmStrategy.simpleDeposit

```
function simpleDeposit(
    SimpleDepositParams calldata params,
```

```
    PositionSettings calldata positionSettings,
    address approved,
    bytes32 referralCode
) public payable {
    Sickle sickle = getOrDeploySickle(msg.sender, approved, referralCode);
```

Listing 3. Excerpt from NftFarmStrategy.deposit

```
function deposit(
    NftDeposit calldata params,
    NftSettings calldata settings,
    address[] calldata sweepTokens,
    address approved,
    bytes32 referralCode
) external payable {
    if (params.increase.zap.addLiquidityParams.tokenId != 0) {
        revert PleaseUseIncrease();
    }
    INftLiquidityConnector liquidityConnector = INftLiquidityConnector(
        connectorRegistry.connectorOf(address(params.nft))
    );
    uint256 initialSupply =
        liquidityConnector.totalSupply(address(params.nft));

    Sickle sickle = getOrDeploySickle(msg.sender, approved, referralCode);
```

Listing 4. Excerpt from NftFarmStrategy.simpleDeposit

```
function simpleDeposit(
    NftPosition calldata position,
    bytes calldata extraData,
    NftSettings calldata settings,
    address approved,
    bytes32 referralCode
) public {
    Sickle sickle = getOrDeploySickle(msg.sender, approved, referralCode);
```

However, a malicious actor can front-run the `deposit` function by calling the `getOrDeploySickle` function to deploy the victim's `Sickle` wallet with their own `approved` and `referralCode` arguments. In this case, the user's `deposit` transaction will not fail and will be executed, and the `approved` and `referralCode` arguments will be set to the malicious actor's values.

Listing 5. Excerpt from StrategyModule

```
function getOrDeploySickle(
    address owner,
    address approved,
    bytes32 referralCode
) public returns (Sickle) {
    return
        Sickle(payable(factory.getOrDeploy(owner, approved, referralCode)));
}
```

To exploit this vulnerability, the following low-likelihood conditions must be met:

1. A user starts interaction with the `Vfat` protocol by calling the `deposit` or `simpleDeposit` function and not by deploying the `Sickle` wallet; and
2. A user specifies flags `PositionSettings.autoExit` or `NftSettings.autoExit` to `true`.

Exploit scenario

Alice, a user, wants to use the `Vfat` protocol. Bob, a malicious actor, monitors the mempool.

1. Alice calls the `deposit` function with `PositionSettings.autoExit` set to `true` and valid parameters in `PositionSettings.exitConfig`.
2. Bob front-runs Alice's transaction by calling the `getOrDeploySickle` function to deploy a `Sickle` wallet for Alice with his own `approved` and `referralCode` arguments.
3. Alice's `deposit` transaction executes successfully, and she does not notice that the `Sickle.approved` state variable is set to another value, and not to the automation contract address.

4. Time passes, and conditions specified in `PositionSettings.exitConfig` are met.
5. Bob creates a `FAKE` token.
6. Bob creates a `FAKE/USDC` pool on `Velodrome`.
7. Bob adds liquidity to the `FAKE/USDC` pool.
8. Bob calls the `exitFor` function to exit Alice's position, and swaps all Alice's tokens to `FAKE` tokens through `Velodrome`.
9. Bob mints enough `FAKE` tokens to himself to completely drain the `FAKE/USDC` pool.
10. Bob swaps all `FAKE` tokens to `USDC` tokens provided by Alice through `Velodrome`.

As a result, all Alice's funds are converted to `FAKE` tokens and have no value.

Recommendation

Verify that the passed `approved` and `referralCode` arguments to the `deposit` or `simpleDeposit` functions equal the `Sickle.approved` and `Sickle.referralCode` state variables.

Acknowledgment 1.1

The issue was acknowledged by the Sickle team with the following comment:

> *This will be corrected in a future Sickle version but does not affect current vfat.io users as we deploy a Sickle contract before making any deposits.*
>
> — Vfat

[Go back to Findings Summary]

L1: The charge fee can be bypassed for several functions

Low severity issue

Impact:	Low	Likelihood:	Medium
Target:	FarmStrategy.sol, NFTFarmStrategy.sol, MultiFarmStrategy.sol	Type:	Logic error

Description

In `FarmStrategy`, the `withdraw` function charges fees from `WithdrawParams.tokensOut` tokens. However, this variable is not verified anywhere in the function logic. Users can set a different token or an empty array, thereby bypassing the fee.

This issue affects the following arrays:

- `HarvestParams.tokensOut`;
- `WithdrawParams.tokensOut`; and
- `CompoundParams.rewardTokens`.

The same issue exists in `NFTFarmStrategy` and `MultiFarmStrategy`.

Listing 6. Excerpt from FarmStrategy._harvest

```
address farmConnector =
    connectorRegistry.connectorOf(farm.stakingContract);
targets[0] = farmConnector;
data[0] = abi.encodeCall(IFarmConnector.claim, (farm, params.extraData));

targets[1] = address(swapLib);
data[1] = abi.encodeCall(ISwapLib.swapMultiple, (params.swaps));

targets[2] = address(feesLib);
```

```
data[2] = abi.encodeCall(
    IFeesLib.chargeFees, (strategyAddress, fee, params.tokensOut)
);

targets[3] = address(transferLib);
data[3] =
    abi.encodeCall(ITransferLib.transferTokensToUser, (sweepTokens));
```

Users can also bypass fees in `simpleHarvest` because `SimpleWithdrawParams.amountOut` in `simpleWithdraw` can be zero. Users can set any token at `SimpleWithdrawParams.lpToken` to sweep tokens without incurring fees.

Listing 7. Excerpt from FarmStrategy._simpleWithdraw

```
address farmConnector =
    connectorRegistry.connectorOf(farm.stakingContract);
targets[0] = farmConnector;
data[0] = abi.encodeCall(
    IFarmConnector.withdraw, (farm, params.amountOut, params.extraData)
);

targets[1] = address(transferLib);
data[1] =
    abi.encodeCall(ITransferLib.transferTokenToUser, (params.lpToken));
```

Exploit scenario

1. Alice, a user, wants to exit from her position but does not want to pay fees.
2. Alice sends a `FarmStrategy.exit` transaction with empty `HarvestParams.tokensOut` and `WithdrawParams.tokensOut`.
3. Alice successfully exits the position without being charged fees.

Recommendation

Implement functionality to recognize the reward tokens and charge fees from them.

Acknowledgment 1.1

The issue was acknowledged by the Sickle team with the following comment:

> *Users can also bypass fees by forking the contracts so changing this would not be enough of a deterrent.*
>
> — Vfat

Go back to Findings Summary

W1: Withdrawal of funds can be blocked by `Collector` contract by not accepting tokens

Impact:	Warning	Likelihood:	N/A
Target:	FeesLib.sol	Type:	Trust model

Description

The protocol charges fees for deposit and withdrawal of funds. If the fee token is specified as `0x0000000000000000000000000000000000000000` or `0xEeeeeEeeeEeEeeEeEeEeeEEEeeeeEeeeeeeeEEeE` (ETH), the protocol sends the fee to the `Collector` contract via the `safeTransferETH` function. However, the `Collector` contract can be configured to not accept ETH. As a result, all withdrawals with the specified fee token will fail.

Listing 8. Excerpt from FeesLib

```
if (feeToken == ETH || feeToken == UNISWAP_ETH) {
    SafeTransferLib.safeTransferETH(
        registry.collector(), amountToCharge
    );
} else {
```

Alice, the protocol admin, configures the `Collector` contract to reject ETH transfers. Bob, a user, attempts to withdraw funds with ETH specified as the fee token. The withdrawal transaction fails because the `Collector` contract refuses to accept the ETH fee payment, effectively blocking Bob's withdrawal.

Recommendation

Implement error handling to ensure withdrawals succeed even when fee collection fails. Add a fallback mechanism that allows the transaction to proceed if the `Collector` contract rejects the fee transfer.

Acknowledgment 1.1

The issue is acknowledged by the Sickle team with the following comment:

> *Noted, collector is currently an EOA.*
>
> — Vfat

Go back to Findings Summary

W2: Connectors are single point of failure

Impact:	Warning	Likelihood:	N/A
Target:	*FarmStrategy.sol	Type:	Trust model

Description

The `Vfat` protocol uses connectors to interact with other protocols. However, if a specific connector fails, transactions will fail or the protocol behavior may become unpredictable.

For example, if a user wants to withdraw funds from a pool on Velodrome but the Velodrome connector fails, the user will be unable to withdraw the funds.

Listing 9. Excerpt from VelodromeRouterConnector.swapExactTokensForTokens

```
function _withdrawNft(
    Sickle sickle,
    NftPosition calldata position,
    bytes calldata extraData
) private {
    address[] memory targets = new address[](1);
    bytes[] memory data = new bytes[](1);

    address farmConnector =
        connectorRegistry.connectorOf(position.farm.stakingContract);

    targets[0] = farmConnector;
    data[0] =
        abi.encodeCall(INftFarmConnector.withdrawNft, (position,
extraData));
```

Recommendation

Implement alternative pathways when primary connectors fail.

Acknowledgment 1.1

The issue was acknowledged by the Sickle team with the following comment:

> *Acknowledged, in the case of Velodrome there is no great way to provide a secondary withdrawal option. The current connectors work and there is a 24 hour timelock for changing them. If it were to fail due to our error we will amend accordingly then.*
>
> — Vfat

Go back to Findings Summary

W3: Usage of function with `inplace=True` argument always fails in gauges that use NFTs

Impact:	Warning	Likelihood:	N/A
Target:	NftFarmStrategy.sol, MultiFarmStrategy.sol	Type:	Logic error

Description

The `NftFarmStrategy` and `MultiFarmStrategy` contracts provide the ability to increase or compound NFT positions by calling the `increase` or `compound` functions in-place, meaning the NFT does not need to be withdrawn from the gauge. However, to perform these operations, the contract or EOA must own the NFT. As a result, every `increase` or `compound` function call with the `inplace=True` argument will fail with the error `NG`.

Recommendation

Remove the `inplace=True` argument option for all functions that interact with NFTs.

Acknowledgment 1.1

The issue is acknowledged by the Sickle team with the following comment:

> *inPlace=true is a configuration option depending on the contract one wants to interact with, it can be used for Uniswap V3 but not for Velodrome.*
>
> — Vfat

[Go back to Findings Summary]

W4: Missing CompoundFor fee calculation

Impact:	Warning	Likelihood:	N/A
Target:	FarmStrategy.sol	Type:	Logic error

Description

Listing 10. Excerpt from FarmStrategy.compoundFor

```
_compound(sickle, params, sweepTokens);
```

The `_compound` function is used for both the `compound` and `compoundFor` functions.

However, there is no option to specify whether the fee type is `compound` or `compoundFor` in the `_compound` function.

Listing 11. Excerpt from FarmStrategy._compound

```
function _compound(
    Sickle sickle,
    CompoundParams calldata params,
    address[] calldata sweepTokens
) private {
    address[] memory targets = new address[](5);
    bytes[] memory data = new bytes[](5);

    address farmConnector =
        connectorRegistry.connectorOf(params.claimFarm.stakingContract);

    targets[0] = farmConnector;
    data[0] = abi.encodeCall(
        IFarmConnector.claim, (params.claimFarm, params.claimExtraData)
    );

    targets[1] = address(feesLib);
    data[1] = abi.encodeCall(
        IFeesLib.chargeFees,
        (strategyAddress, FarmStrategyFees.Compound, params.rewardTokens)
```

```
656     );
```

This implementation is inconsistent with the `NftFarmStrategy` contract.

Recommendation

Add a parameter to the `_compound` function to specify the fee type. Set this parameter appropriately in the `compound` and `compoundFor` functions.

Fix 1.1

The issue was fixed in the commit `d05cdd9`[1] by adding a parameter to the `_compound` function to specify the fee type.

Go back to Findings Summary

W5: `block.timestamp` is used for swap deadline

Impact:	Warning	Likelihood:	N/A
Target:	VelodromeRouterConnector.sol	Type:	Logic error

Description

The `block.timestamp` is used as the deadline for the swap operation.

However, the deadline should be provided as a user parameter.

Listing 12. Excerpt from VelodromeRouterConnector.swapExactTokensForTokens

```
IRouter(swap.router).swapExactTokensForTokens(
    swap.amountIn,
    swap.minAmountOut,
    _extraData.routes,
    address(this),
    block.timestamp
);
```

Slippage is already managed by `minOutAmount`.

Recommendation

Add a parameter for the user to set the deadline.

Acknowledgment 1.1

The issue was acknowledged by the Sickle team with the following comment:

> *Acknowledged, will be amended if there is user demand.*
>
> — Vfat

Go back to Findings Summary

W6: Incorrect price calculation

Impact:	Warning	Likelihood:	N/A
Target:	FarmStrategy.sol	Type:	Logic error

Description

The `VelodromeRouterConnector.getPoolPrice` function calculates the price using information from the pool.

However, the calculation may return incorrect values due to precision loss, causing the view function to return incorrect values.

Additionally, the `VelodromeRouterConnector.getPoolPrice` function uses the `getAmountOut` function from Velodrome, which is potentially vulnerable to price manipulation.

Listing 13. Excerpt from VelodromeRouterConnector.getPoolPrice

```
    address token0 = ICLPool(lpToken).token0();
    address token1 = ICLPool(lpToken).token1();

    uint256 amountOut0 = ICLPool(lpToken).getAmountOut(1, token0);
    if (amountOut0 > 0) {
        price = amountOut0 * 1e18;
    } else {
        uint256 amountOut1 = ICLPool(lpToken).getAmountOut(1, token1);
        if (amountOut1 == 0) {
            revert InvalidPrice();
        }
        price = 1e18 / amountOut1;
    }

    if (price == 0) {
        revert InvalidPrice();
    }

    if (baseTokenIndex == 1) {
        price = 1e36 / price;
    }
```

```
}
```

Listing 14. Excerpt from PositionSettingsRegistry.validateExitFor

```
uint256 price = connector.getPoolPrice(
    address(settings.pool),
    config.baseTokenIndex,
    config.quoteTokenIndex
);

bool priceBelowRange = price < config.triggerPriceLow;

bool priceAboveRange = price > config.triggerPriceHigh;
```

This function is used in the `exitFor` function in `FarmStrategy`.

Listing 15. Excerpt from FarmStrategy.exitFor

```
function exitFor(
    Sickle sickle,
    Farm calldata farm,
    HarvestParams calldata harvestParams,
    address[] calldata harvestSweepTokens,
    WithdrawParams calldata withdrawParams,
    address[] calldata withdrawSweepTokens
) external override onlyApproved(sickle) {
    positionSettingsRegistry.validateExitFor(
        PositionKey({
            sickle: sickle,
            stakingContract: farm.stakingContract,
            poolIndex: farm.poolIndex
        })
    );
```

Exploit scenario

The attack can be combined with front-running of deployment at deposit. In this scenario:

1. Alice, a user, starts using the protocol with `deposit`. Alice sets

`PositionSettings`. Alice sets `ExitConfig` with the pool and certain `triggerPriceLow` and `triggerPriceHigh` values. These values, with the condition at `validateExitFor`, should not be met at this moment.

2. Bob, an attacker, observes this transaction.
3. Bob sends a transaction that creates a Sickle with `getOrDeploySickle` for Alice, setting himself as the approved automator.
4. Alice's transaction succeeds.
5. Bob performs a flashloan to manipulate the price in the pool and sends `exitFor` that withdraws Alice's tokens and swaps with a fake token.
6. Alice observes that the approved address differs from her intention. Alice sends a transaction to change the approved address; however, Bob's `exitFor` transaction succeeds first and Alice loses the value of her funds.

Recommendation

Use an anti-manipulation price function, for example, the `quote` function from Velodrome.

Acknowledgment 1.1

The issue is acknowledged by the Sickle team with the following comment:

> - *Precision loss: Noted, this is acceptable at present, may be looked into in detail in the future.*
> - *Price manipulation: Not possible on any of the networks Velodrome is deployed due to centralized sequencers.*
>
> — Vfat

[Go back to Findings Summary](#)

I1: Missing NatSpec comments

Impact:	Info	Likelihood:	N/A
Target:	SickleFactory.sol	Type:	Code quality

Description

The `SickleFactory` contract lacks NatSpec comments for the `approved` parameter in the `deploy` and `getOrDeploy` functions.

Listing 16. Excerpt from SickleFactory

```
/// @notice Deploys a new Sickle contract for a specific user
/// @dev Sickle contracts are deployed with create2, the address of the
/// admin is used as a salt, so all the Sickle addresses can be pre-computed
/// and only 1 Sickle will exist per address
/// @param referralCode Referral code for the user
/// @return sickle Address of the deployed Sickle contract
function deploy(
    address approved,
    bytes32 referralCode
) external returns (address sickle) {
```

Listing 17. Excerpt from SickleFactory

```
/// @notice Deploys a new Sickle contract for a specific user, or returns
/// the existing one if it exists
/// @param admin Address receiving the admin rights of the Sickle contract
/// @param referralCode Referral code for the user
/// @return sickle Address of the deployed Sickle contract
function getOrDeploy(
    address admin,
    address approved,
    bytes32 referralCode
) external returns (address sickle) {
```

Recommendation

Add NatSpec comments for the `approved` parameter in the `deploy` and `getOrDeploy` functions.

Fix 1.1

The issue was fixed in the commit `359fab7`[2] by adding NatSpec comments for the `approved` parameter in the `deploy` and `getOrDeploy` functions.

[Go back to Findings Summary]

I2: Potential incorrect fee calculation

Impact:	Info	Likelihood:	N/A
Target:	TransferLib.sol	Type:	Logic error

Description

The `TransferLib` contains functionality responsible for transferring tokens from a user to their Sickle wallet. However, fee miscalculation can occur when a user has additional tokens in their Sickle wallet.

First, the internal function `_transferTokenFromUser` makes a `SafeTransferLib.safeTransferFrom` call transferring funds from the user to the corresponding Sickle address.

Listing 18. Excerpt from TransferLib

```
SafeTransferLib.safeTransferFrom(
    tokenIn,
    Sickle(payable(address(this))).owner(),
    address(this),
    amountIn
);
```

Second, it calls `FeesLib.chargeFee` passing a value of 0 for the `feeBasis` argument.

Listing 19. Excerpt from TransferLib

```
bytes memory result = _delegateTo(
    address(feesLib),
    abi.encodeCall(
        IFeesLib.chargeFee, (strategy, feeSelector, tokenIn, 0)
    )
);
```

Based on the core logic of the `FeesLib.chargeFee` function, it calculates fees from the Sickle balance rather than the transfer amount. The project is not designed to hold funds in the Sickle account, but if funds somehow remain there, it will lead to incorrect fee calculation and potentially unintended loss of funds from the Sickle balance to the collector.

Listing 20. Excerpt from TransferLib

```
    }
}

/// @dev Transfers all balances of {tokens} and/or ETH from the contract
/// to the sickle owner
/// @param tokens An array of token addresses
function transferTokensToUser(
    address[] memory tokens
) external payable checkTransfersTo(tokens) {
    for (uint256 i; i != tokens.length;) {
        transferTokenToUser(tokens[i]);
```

Recommendation

Calculate fees based on the transfer amount rather than the Sickle balance.

Fix 1.1

The issue was fixed in the commit `a721aeb`[3] by calculating fees based on the transfer amount rather than the Sickle balance.

[Go back to Findings Summary]

I3: Unexpected revert in `increase` function

Impact:	Info	Likelihood:	N/A
Target:	NftFarmStrategy.sol	Type:	Logic error

Description

When a user calls the `NftFarmStrategy.increase` function with a `tokenId` value of `0`, the transaction reverts with the error `NftFarmStrategy.NftSupplyChanged`, which does not clearly communicate what should be changed for the transaction to succeed.

Listing 21. Excerpt from NftFarmStrategy

```
function increase(
    NftPosition calldata position,
    NftHarvest calldata harvestParams,
    NftIncrease calldata increaseParams,
    bool inPlace, // Increase without withdrawing
    address[] calldata sweepTokens
) external payable nftSupplyUnchanged(position.nft) {
    Sickle sickle = getSickle(msg.sender);

    if (!inPlace) {
        _harvest(
            sickle, position, harvestParams, NftFarmStrategyFees.Harvest
        );
        _withdrawNft(sickle, position, increaseParams.extraData);
    }

    _transferInTokens(sickle, increaseParams);

    _zapIn(sickle, increaseParams.zap);
```

However, when a user calls the `NftFarmStrategy.deposit` function and specifies a `tokenId` value other than `0`, the transaction reverts with the clear error `NftFarmStrategy.PleaseUseIncrease`.

Listing 22. Excerpt from NftFarmStrategy

```
    NftDeposit calldata params,
    NftSettings calldata settings,
    address[] calldata sweepTokens,
    address approved,
    bytes32 referralCode
) external payable {
    if (params.increase.zap.addLiquidityParams.tokenId != 0) {
        revert PleaseUseIncrease();
    }
    INftLiquidityConnector liquidityConnector = INftLiquidityConnector(
```

Recommendation

Implement consistent and descriptive error messages for both the `increase` and `deposit` functions to clearly indicate the required conditions for successful execution.

Fix 1.1

The issue was fixed in the commit `31e1440`[4] by adding a check in the `increase` function to revert if `tokenId` is `0`.

[Go back to Findings Summary]

I4: Missing events in `MultiFarmStrategy`

Impact:	Info	Likelihood:	N/A
Target:	MultiFarmStrategy.sol	Type:	Logging

Description

The `NftFarmStrategy` contract emits events in the `_depositNft` and `_withdrawNft` functions. However, the `MultiFarmStrategy` contract does not emit any events despite having similar logic. This inconsistency exists between the `NftFarmStrategy` and `MultiFarmStrategy` contracts.

Listing 23. Excerpt from NftFarmStrategy

```
targets[0] = farmConnector;
data[0] = abi.encodeCall(
    INftFarmConnector.depositExistingNft, (position, extraData)
);

sickle.multicall(targets, data);

emit SickleDepositedNft(
    sickle,
    position.nft,
    position.tokenId,
    position.farm.stakingContract,
    position.farm.poolIndex
);
```

Listing 24. Excerpt from NftFarmStrategy

```
targets[0] = farmConnector;
data[0] =
    abi.encodeCall(INftFarmConnector.withdrawNft, (position, extraData));

sickle.multicall(targets, data);

emit SickleWithdrewNft(
    sickle,
    position.nft,
```

```
    position.tokenId,
    position.farm.stakingContract,
    position.farm.poolIndex
);
```

Listing 25. Excerpt from MultiFarmStrategy

```
targets[0] = farmConnector;
data[0] = abi.encodeCall(
    INftFarmConnector.depositExistingNft, (position, extraData)
);

sickle.multicall(targets, data);
```

Listing 26. Excerpt from MultiFarmStrategy

```
targets[0] = farmConnector;
data[0] =
    abi.encodeCall(INftFarmConnector.withdrawNft, (position, extraData));

sickle.multicall(targets, data);
```

Recommendation

Emit the same events in the `_depositNft` and `_withdrawNft` functions in the `MultiFarmStrategy` contract as in the `NftFarmStrategy` contract.

Fix 1.1

The issue was fixed in commits `1073175`[5] and `c50a800`[6] by adding events to the `MultiFarmStrategy` contract.

[Go back to Findings Summary](#)

[1] full commit hash: `d05cdd95f02c3967a02975fb23e130c7c2b6360a`

[2] full commit hash: `359fab7f0b2d8e58031a56e1bc3d5ed6dd726d15`

[3] full commit hash: `a721aeb595e1c6376eaf22fb23c6436178466163`

[4] full commit hash: `31e14403c07a99b8d3d7d008e3d593b358f863d6`

[5] full commit hash: `1073175c6f4041f3358ed35499955df4bb100969`

[6] full commit hash: `c50a8001cbfebf162438c16324fe739173be1398`

Appendix A: How to cite

Please cite this document as:

Ackee Blockchain Security, Vfat: Farm Strategies, 24.6.2025.

Appendix B: Wake Findings

This section lists the outputs from the Wake framework used for testing and static analysis during the audit.

B.1. Detectors

ackee
blockchain security

Thank You

Ackee Blockchain a.s.

Rohanske nabrezi 717/4
186 00 Prague
Czech Republic

hello@ackee.xyz