Gamma

Date	February 2022
Auditors	Sergii Kravchenko, David Oz Kashi

1 Executive Summary

This report presents the results of our engagement with **Gamma** to review its smart contracts.

The initial review was conducted over two weeks, from **January 31, 2022** to **February 11, 2022** by **Sergii Kravchenko** and **David Oz Kashi**. A total of 20 person-days were spent. Mitigations review was conducted over additional two weeks, from **March 14, 2022** to **March 25, 2022**. A total of 10 person-days were spent on the mitigations review.

The inital review of was conducted on a best-effort basis, the code was not production ready. Later, the **Gamma** team introduced fixes to the issues mentioned in the original report. Mitigations were reviewed by us, all issues except from improving the test coverage were addressed.

2 Scope

Our review focused on the commit hash `41fd4abf79864478523e87924d4e80d80df04879` . Mitigations review focused on the commit hash `9a7a3dd88e8e8b106bf5d0e4c56e879442a72181` The list of files in scope can be found in the Appendix.

2.1 Objectives

Together with the **Gamma** team, we identified the following priorities for our review:

1. Ensure that the system is implemented consistently with the intended functionality, and without unintended edge cases.
2. Identify known vulnerabilities particular to smart contract systems, as outlined in our Smart Contract Best Practices, and the Smart Contract Weakness Classification Registry.

2.2 Discussion

- During the review we have discovered that the system is heavily parameterized by the owners of `UniProxy` and `Hypervisor` . We recommend implementing a time-lock that informs users of planned changes and gives them sufficient time to react to an unwanted change. It is also recommended to use a multisig contract or other transparent governance mechanisms to initiate changes, and ensure that private keys are managed securely.
- The ownership transfer is one-step which might come with a significant risk of losing access to the contract.
- Token contracts have to be carefully vetted for compatibility with Gamma. Bugs, privileges of operators, non-standard or weird behavior, and more or less accepted features like blacklisting, and upgradeability obviously can have an impact on the protocol and cause lost or stuck funds.

3 Findings

Each issue has an assigned severity:

- Minor issues are subjective in nature. They are typically suggestions around best practices or readability. Code maintainers should use their own judgment as to whether to address such issues.
- Medium issues are objective in nature but are not security vulnerabilities. These should be addressed unless there is a clear reason not to.
- Major issues are security vulnerabilities that may not be directly exploitable or may require certain conditions in order to be exploited. All major issues should be addressed.
- Critical issues are directly exploitable security vulnerabilities that need to be fixed.

3.1 The `Hypervisor.deposit` function does not check the `msg.sender` Critical ✓ Fixed

Resolution

Partially fixed in GammaStrategies/hypervisor@ `9a7a3dd` , by allowing only `whitelistedAddress` to call `deposit` , or anyone if `whitelisted = false` (currently it is set to `true` by default).

Description

`Hypervisor.deposit` pulls pre-approved ERC20 tokens from the `from` address to the contract. Later it mints shares to the `to` address. Attackers can determine both the `from` and `to` addresses as they wish, and thus steal shares (that can be redeemed to tokens immediately) from users that pre-approved the contract to spend ERC20 tokens on their behalf.

Recommendation

As described in issue 3.5, we recommend restricting access to this function only for `UniProxy`. Moreover, the `UniProxy` contract should validate that `from == msg.sender`.

3.2 `UniProxy.depositSwap` - Tokens are not approved before calling `Router.exactInput` Major

✓ Fixed

Resolution
Fixed in GammaStrategies/hypervisor@`9a7a3dd` by deleting the `depositSwap` function.

Description

the call to `Router.exactInput` requires the sender to pre-approve the tokens. We could not find any reference for that, thus we assume that a call to `UniProxy.depositSwap` will always revert.

Examples

code/contracts/UniProxy.sol:L202-L234

```
router = ISwapRouter(_router);
uint256 amountOut;
uint256 swap;
if(swapAmount < 0) {
    //swap token1 for token0

    swap = uint256(swapAmount * -1);
    IHypervisor(pos).token1().transferFrom(msg.sender, address(this), deposit1+swap);
    amountOut = router.exactInput(
        ISwapRouter.ExactInputParams(
            path,
            address(this),
            block.timestamp + swapLife,
            swap,
            deposit0
        )
    );
}
else{
    //swap token1 for token0
    swap = uint256(swapAmount);
    IHypervisor(pos).token0().transferFrom(msg.sender, address(this), deposit0+swap);

    amountOut = router.exactInput(
        ISwapRouter.ExactInputParams(
            path,
            address(this),
            block.timestamp + swapLife,
            swap,
            deposit1
        )
    );
}
```

Recommendation

Consider approving the exact amount of input tokens before the swap.

3.3 `Uniproxy.depositSwap` - `_router` should not be determined by the caller Major ✓ Fixed

Resolution
Fixed in GammaStrategies/hypervisor@`9a7a3dd` by deleting the `depositSwap` function.

Description

`Uniproxy.depositSwap` uses `_router` that is determined by the caller, which in turn might inject a “fake” contract, and thus may steal funds stuck in the `UniProxy` contract.

The `UniProxy` contract has certain trust assumptions regarding the router. The router is supposed to return not less than `deposit1` (or `deposit0`) amount of tokens but that fact is never checked.

Examples

code/contracts/UniProxy.sol:L168-L177

```
function depositSwap(
  int256 swapAmount, // (-) token1, (+) token0 for token1; amount to swap
  uint256 deposit0,
  uint256 deposit1,
  address to,
  address from,
  bytes memory path,
  address pos,
  address _router
) external returns (uint256 shares) {
```

Recommendation

Consider removing the `_router` parameter from the function, and instead, use a storage variable that will be initialized in the constructor.

3.4 Re-entrancy + flash loan attack can invalidate price check Major ✓ Fixed

Resolution
Fixed in GammaStrategies/hypervisor@`9a7a3dd` by implementing the auditor's recommendation.

Description

The `UniProxy` contract has a price manipulation protection:

code/contracts/UniProxy.sol:L75-L82

```
if (twapCheck || positions[pos].twapOverride) {
  // check twap
  checkPriceChange(
    pos,
    (positions[pos].twapOverride ? positions[pos].twapInterval : twapInterval),
    (positions[pos].twapOverride ? positions[pos].priceThreshold : priceThreshold)
  );
}
```

But after that, the tokens are transferred from the user, if the token transfer allows an attacker to hijack the call-flow of the transaction inside, the attacker can manipulate the Uniswap price there, after the check happened. The Hypervisor's `deposit` function itself is vulnerable to the flash-loan attack.

Recommendation

Make sure the price does not change before the `Hypervisor.deposit` call. For example, the token transfers can be made at the beginning of the `UniProxy.deposit` function.

3.5 The `deposit` function of the Hypervisor contract should only be called from `UniProxy` Major ✓ Fixed

Resolution
Partially fixed in GammaStrategies/hypervisor@`9a7a3dd`, by allowing only `whitelistedAddress` to call `deposit`, or anyone if `whitelisted = false` (currently it is set to `true` by default).

Description

The `deposit` function is designed to be called only from the `UniProxy` contract, but everyone can call it. This function does not have any protection against price manipulation in the Uniswap pair. A deposit can be frontrunned, and the depositor's funds may be "stolen".

Recommendation

Make sure only `UniProxy` can call the `deposit` function.

3.6 `UniProxy.properDepositRatio` - Proper ratio will not prevent liquidity imbalance for all possible scenarios Major ✓ Fixed

Resolution
Fixed in GammaStrategies/hypervisor@`9a7a3dd` by deleting the `properDepositRatio` function.

Description

`UniProxy.properDepositRatio` purpose is to be used as a mechanism to prevent liquidity imbalance. The idea is to compare the deposit ratio with the `hypeRatio`, which is the ratio between the tokens held by the `Hypervisor` contract. In practice, however, this function will not prevent a skewed deposit ratio in many cases. `deposit1 / deposit0` might be a huge number, while `10^16 <= depositRatio <= 10^18`, and `10^16 <= hypeRatio <= 10^18`. Let us consider the case where `hype1 / hype0 >= 10`, that means `hypeRatio = 10^18`, and now if `deposit1 / deposit0 = 10^200` for example, `depositRatio = 10^18`, and the transaction will pass, which is clearly not intended.

Examples

code/contracts/UniProxy.sol:L258-L275

```
function properDepositRatio(
  address pos,
  uint256 deposit0,
  uint256 deposit1
) public view returns (bool) {
  (uint256 hype0, uint256 hype1) = IHypervisor(pos).getTotalAmounts();
  if (IHypervisor(pos).totalSupply() != 0) {
    uint256 depositRatio = deposit0 == 0 ? 10e18 : deposit1.mul(1e18).div(deposit0);
    depositRatio = depositRatio > 10e18 ? 10e18 : depositRatio;
    depositRatio = depositRatio < 10e16 ? 10e16 : depositRatio;
    uint256 hypeRatio = hype0 == 0 ? 10e18 : hype1.mul(1e18).div(hype0);
    hypeRatio = hypeRatio > 10e18 ? 10e18 : hypeRatio;
    hypeRatio = hypeRatio < 10e16 ? 10e16 : hypeRatio;
    return (FullMath.mulDiv(depositRatio, deltaScale, hypeRatio) < depositDelta &&
            FullMath.mulDiv(hypeRatio, deltaScale, depositRatio) < depositDelta);
  }
  return true;
}
```

Recommendation

Consider removing the cap of [0.1,10] both for `depositRatio` and for `hypeRatio`.

3.7 `UniProxy` - `SafeERC20` is declared but safe functions are not used Major ✓ Fixed

Resolution
fixed in GammaStrategies/hypervisor@ `9a7a3dd` by implementing the auditor's recommendation.

Description

The `UniProxy` contract declares the usage of the `SafeERC20` library for functions of the `IERC20` type. However, unsafe functions are used instead of safe ones.

Examples

- Usage of `approve` instead of `safeApprove`
- Usage of `transferFrom` instead of `safeTransferFrom`.

3.8 Missing/wrong implementation Major ✓ Fixed

Resolution
1. Fixed in GammaStrategies/hypervisor@ `9a7a3dd` by introducing two new functions: `toggleDepositOverride`, `setPriceThresholdPos`. 2. Fixed in GammaStrategies/hypervisor@ `9a7a3dd` by keeping only the version of `deposit` function with 4 parameters. 3. Fixed in GammaStrategies/hypervisor@ `9a7a3dd` by removing the unreachable code.

Examples

1. The `UniProxy` contract has different functions used for setting the properties of a position. However, `Position.priceThreshold`, and `Position.depositOverride` are never assigned to, even though they are being used.
2. `UniProxy.deposit` is calling `IHypervisor.deposit` multiple times with different function signatures (3 and 4 parameters), while the `Hypervisor` contract only implements the version with 4 parameters, and does not implement the `IHypervisor` interface.
3. `Hypervisor.uniswapV3MintCallback | uniswapV3SwapCallback` - both these functions contain unreachable code, namely the case where `payer != address(this)`.

Recommendations

1. Consider adding functions to set these properties, or alternatively, a single function to set the properties of a position.
2. Consider supporting a single `deposit` function for `IHypervisor`, and make sure that the actual implementation adheres to this interface.
3. Consider deleting these lines.

3.9 `Hypervisor.withdraw` - Possible reentrancy Major ✓ Fixed

Resolution
Fixed in GammaStrategies/hypervisor@ `9a7a3dd` by implementing the auditor's recommendation.

Description

`Hypervisor.withdraw` can be used by a liquidity provider to withdraw its deposit from the `Hypervisor` contract. A user can get his deposited liquidity back in exchange for the burn of his `shares`. The function is transferring `token0,1` to the user first and then burns his `shares`. In theory, the contracts of `token0,1` may hijack the execution call-flow causing a reentrant call to `deposit`, which will use the stale value for `totalSupply()` to evaluate the number of shares to be minted. Since this value will be greater than what

it should be, the attacker will be able to mint `shares` for free, that could be later redeemed for actual tokens stolen from other depositors.

Recommendation

Consider adding a `ReentrancyGuard` both to `Hypervisor.withdraw` and `Hypervisor.deposit`

3.10 `UniProxy.depositSwap` doesn't deposit all the users' funds Medium ✓ Fixed

Resolution

Fixed in GammaStrategies/hypervisor@`9a7a3dd` by deleting the `depositSwap` function.

Description

When executing the swap, the minimal amount out is passed to the router (`deposit1` in this example), but the actual swap amount will be `amountOut` . But after the trade, instead of depositing `amountOut` , the contract tries to deposit `deposit1` , which is lower. This may result in some users' funds staying in the `UniProxy` contract.

code/contracts/UniProxy.sol:L220-L242

```
else{
    //swap token1 for token0
    swap = uint256(swapAmount);
    IHypervisor(pos).token0().transferFrom(msg.sender, address(this), deposit0+swap);

    amountOut = router.exactInput(
        ISwapRouter.ExactInputParams(
            path,
            address(this),
            block.timestamp + swapLife,
            swap,
            deposit1
        )
    );
}

require(amountOut > 0, "Swap failed");

if (positions[pos].version < 2) {
  // requires lp token transfer from proxy to msg.sender
  shares = IHypervisor(pos).deposit(deposit0, deposit1, address(this));
  IHypervisor(pos).transfer(to, shares);
}
```

Recommendation

Deposit all the user's funds to the Hypervisor.

3.11 `Hypervisor` - Multiple "sandwiching" front running vectors Medium ✓ Fixed

Resolution

Fixed in GammaStrategies/hypervisor@`9a7a3dd` by removing the call to `pool.swap` , and adopting the auditor recommendation for `pool.mint` , `pool.burn` with `slippage = 10%`

Description

The amount of tokens received from `UniswapV3Pool` functions might be manipulated by front-runners due to the decentralized nature of AMMs, where the order of transactions can not be pre-determined. A potential "sandwicher" may insert a buying order before the user's call to `Hypervisor.rebalance` for instance, and a sell order after.

More specifically, calls to `pool.swap` , `pool.mint` , `pool.burn` are susceptible to "sandwiching" vectors.

Examples

`Hypervisor.rebalance`

code/contracts/Hypervisor.sol:L278-L286

```
if (swapQuantity != 0) {
    pool.swap(
        address(this),
        swapQuantity > 0,
        swapQuantity > 0 ? swapQuantity : -swapQuantity,
        swapQuantity > 0 ? TickMath.MIN_SQRT_RATIO + 1 : TickMath.MAX_SQRT_RATIO - 1,
        abi.encode(address(this))
    );
}
```

code/contracts/Hypervisor.sol:L348-L363

```
function _mintLiquidity(
    int24 tickLower,
    int24 tickUpper,
    uint128 liquidity,
    address payer
) internal returns (uint256 amount0, uint256 amount1) {
    if (liquidity > 0) {
        (amount0, amount1) = pool.mint(
            address(this),
            tickLower,
            tickUpper,
            liquidity,
            abi.encode(payer)
        );
    }
}
```

code/contracts/Hypervisor.sol:L365-L383

```
function _burnLiquidity(
    int24 tickLower,
    int24 tickUpper,
    uint128 liquidity,
    address to,
    bool collectAll
) internal returns (uint256 amount0, uint256 amount1) {
    if (liquidity > 0) {
        // Burn liquidity
        (uint256 owed0, uint256 owed1) = pool.burn(tickLower, tickUpper, liquidity);

        // Collect amount owed
        uint128 collect0 = collectAll ? type(uint128).max : _uint128Safe(owed0);
        uint128 collect1 = collectAll ? type(uint128).max : _uint128Safe(owed1);
        if (collect0 > 0 || collect1 > 0) {
            (amount0, amount1) = pool.collect(to, tickLower, tickUpper, collect0, collect1);
        }
    }
}
```

Recommendation

Consider adding an `amountMin` parameter(s) to ensure that at least the `amountMin` of tokens was received.

3.12 Full test suite is necessary Medium

Description

The test suite at this stage is not complete. It is crucial to have a full test coverage that includes the edge cases and failure scenarios, especially for complex system like Gamma.

As we've seen in some smart contract incidents, a complete test suite can prevent issues that might be hard to find with manual reviews.

Some issues such as issue 3.8, issue 3.2 could be caught by a full-coverage test suite.

3.13 Uniswap v3 callbacks access control should be hardened Minor ✓ Fixed

Resolution

Fixed in GammaStrategies/hypervisor@`9a7a3dd` by implementing the auditor's recommendation for `uniswapV3MintCallback`, and deleting `uniswapV3SwapCallback` and the call to `pool.swap`.

Description

Uniswap v3 uses a callback pattern to pull funds from the caller. The caller, (in this case `Hypervisor`) has to implement a callback function which will be called by the Uniswap's pool. Both `uniswapV3MintCallback` and `uniswapV3SwapCallback` restrict the access to the callback functions only for the `pool`. However, this alone will not block a random call from the `pool` contract in case the latter was hacked, which will result in stealing all the funds held in `Hypervisor` or of any user that approved the `Hypervisor` contract to transfer tokens on his behalf.

Examples

code/contracts/Hypervisor.sol:L407-L445

```
function uniswapV3MintCallback(
    uint256 amount0,
    uint256 amount1,
    bytes calldata data
) external override {
    require(msg.sender == address(pool));
    address payer = abi.decode(data, (address));

    if (payer == address(this)) {
        if (amount0 > 0) token0.safeTransfer(msg.sender, amount0);
        if (amount1 > 0) token1.safeTransfer(msg.sender, amount1);
    } else {
        if (amount0 > 0) token0.safeTransferFrom(payer, msg.sender, amount0);
        if (amount1 > 0) token1.safeTransferFrom(payer, msg.sender, amount1);
    }
}

function uniswapV3SwapCallback(
    int256 amount0Delta,
    int256 amount1Delta,
    bytes calldata data
) external override {
    require(msg.sender == address(pool));
    address payer = abi.decode(data, (address));

    if (amount0Delta > 0) {
        if (payer == address(this)) {
            token0.safeTransfer(msg.sender, uint256(amount0Delta));
        } else {
            token0.safeTransferFrom(payer, msg.sender, uint256(amount0Delta));
        }
    } else if (amount1Delta > 0) {
        if (payer == address(this)) {
            token1.safeTransfer(msg.sender, uint256(amount1Delta));
        } else {
            token1.safeTransferFrom(payer, msg.sender, uint256(amount1Delta));
        }
    }
}
```

Recommendation

Consider adding (boolean) storage variables that will help to track whether a call to `uniswapV3MintCallback | uniswapV3SwapCallback` was preceded by a call to `_mintLiquidity | rebalance` respectively. An example for the `rebalance` function would be `bool rebalanceCalled`, this variable will be assigned a `true` value in `rebalance` before the external call of `pool.swap`, then `uniswapV3SwapCallback` will require that `rebalanceCalled == true`, and then right after `rebalanceCalled` will be assigned a `false` value.

3.14 Code quality comments Minor ✓ Fixed

Resolution

1. Fixed in GammaStrategies/hypervisor@`9a7a3dd` by removing the `from` parameter.
2. Fixed in GammaStrategies/hypervisor@`9a7a3dd` by implementing the auditor's recommendation.
3. Fixed in GammaStrategies/hypervisor@`9a7a3dd` by deleting `depositSwap`.

Examples

1. `UniProxy.deposit` - `from` parameter is never used.
2. `UniProxy` - `MAX_INT` should be changed to `MAX_UINT`.
3. Consider using compiler version >= 0.8.0, and make sure that the compiler version is specified explicitly for every .sol file in the repo.
4. `UniProxy` - Minimize code duplication in `deposit` and `depositSwap`.

Appendix 1 - Files in Scope

This audit covered the following files:

File	SHA-1 hash
/contracts/UniProxy.sol	58f485f3b0638da3a953a06e4a5f24c46c313869
/contracts/Hypervisor.sol	91170d74e8b49f874ffb4c8663225a93f81b24ec

Appendix 2 - Disclosure

ConsenSys Diligence ("CD") typically receives compensation from one or more clients (the "Clients") for performing the analysis contained in these reports (the "Reports"). The Reports may be distributed through other means, including via ConsenSys publications and other distributions.

The Reports are not an endorsement or indictment of any particular project or team, and the Reports do not guarantee the security of any particular project. This Report does not consider, and should not be interpreted as considering or having any bearing on, the potential economics of a token, token sale or any other product, service or other asset. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. No Report provides any warranty or representation to any Third-Party in any respect, including regarding the bugfree nature of code, the business model or proprietors of any such business model, and the legal compliance of any such business. No third party should rely on the Reports

in any way, including for the purpose of making any decisions to buy or sell any token, product, service or other asset. Specifically, for the avoidance of doubt, this Report does not constitute investment advice, is not intended to be relied upon as investment advice, is not an endorsement of this project or team, and it is not a guarantee as to the absolute security of the project. CD owes no duty to any Third-Party by virtue of publishing these Reports.

PURPOSE OF REPORTS The Reports and the analysis described therein are created solely for Clients and published with their consent. The scope of our review is limited to a review of code and only the code we note as being within the scope of our review within this report. Any Solidity code itself presents unique and unquantifiable risks as the Solidity language itself remains under development and is subject to unknown risks and flaws. The review does not extend to the compiler layer, or any other areas beyond specified code that could present security risks. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. In some instances, we may perform penetration testing or infrastructure assessments depending on the scope of the particular engagement.

CD makes the Reports available to parties other than the Clients (i.e., “third parties”) – on its website. CD hopes that by making these analyses publicly available, it can help the blockchain ecosystem develop technical best practices in this rapidly evolving area of innovation.

LINKS TO OTHER WEB SITES FROM THIS WEB SITE You may, through hypertext or other computer links, gain access to web sites operated by persons other than ConsenSys and CD. Such hyperlinks are provided for your reference and convenience only, and are the exclusive responsibility of such web sites’ owners. You agree that ConsenSys and CD are not responsible for the content or operation of such Web sites, and that ConsenSys and CD shall have no liability to you or any other person or entity for the use of third party Web sites. Except as described below, a hyperlink from this web Site to another web site does not imply or mean that ConsenSys and CD endorses the content on that Web site or the operator or operations of that site. You are solely responsible for determining the extent to which you may use any content at any other web sites to which you link from the Reports. ConsenSys and CD assumes no responsibility for the use of third party software on the Web Site and shall have no liability whatsoever to any person or entity for the accuracy or completeness of any outcome generated by such software.

TIMELINESS OF CONTENT The content contained in the Reports is current as of the date appearing on the Report and is subject to change without notice. Unless indicated otherwise, by ConsenSys and CD.