divergence SWAP2 Security Review

Auditors

Noah Marconi, Lead Security Researcher

Deadrosesxyz, Lead Security Researcher

Kaden, Security Researcher

Report prepared by: Lucas Goiriz

October 7, 2024

Contents

1 About Spearbit

Spearbit is a decentralized network of expert security engineers offering reviews and other security related services to Web3 projects with the goal of creating a stronger ecosystem. Our network has experience on every part of the blockchain technology stack, including but not limited to protocol design, smart contracts and the Solidity compiler. Spearbit brings in untapped security talent by enabling expert freelance auditors seeking flexibility to work on interesting projects together.

Learn more about us at spearbit.com

2 Introduction

divergence is a boutique software studio providing engineering and product-management services to web3 clients. Specialising in Ethereum-compatible smart contracts and tooling, they developed the `SWAP2` NFT-trading protocol for brokerage firm, Fountain Digital. `SWAP2` provides a least-privileges alternative to other trading protocols, circumventing the risks associated with approval-for-all trading by using ephemeral but gas-efficient contracts.

Disclaimer: This security review does not guarantee against a hack. It is a snapshot in time of kairos according to the specific commit. Any modifications to the code will require a new security review.

3 Risk classification

Severity level	Impact: High	Impact: Medium	Impact: Low
Likelihood: high	Critical	High	Medium
Likelihood: medium	High	Medium	Low
Likelihood: low	Medium	Low	Low

3.1 Impact

- High - leads to a loss of a significant portion (>10%) of assets in the protocol, or significant harm to a majority of users.
- Medium - global losses <10% or losses to only a subset of users, but still unacceptable.
- Low - losses will be annoying but bearable--applies to things like griefing attacks that can be easily repaired or even gas inefficiencies.

3.2 Likelihood

- High - almost certain to happen, easy to perform, or not easy but highly incentivized
- Medium - only conditionally possible or incentivized, but still relatively likely
- Low - requires stars to align, or little-to-no incentive

3.3 Action required for severity levels

- Critical - Must fix as soon as possible (if already deployed)
- High - Must fix (before deployment if not already deployed)
- Medium - Should fix
- Low - Could fix

4 Executive Summary

Over the course of 7 days in total, divergence engaged with Spearbit to review the SWAP2 protocol. In this period of time a total of **16** issues were found.

Summary

Project Name	divergence
Repository	SWAP2
Commit	dd8bb632
Type of Project	NFT Trading
Audit Timeline	Jul 15th to Jul 22nd

The Spearbit team reviewed divergence's SWAP2 changes holistically on commit hash 78e2c4187ab6cc0ca2a471bae5cef63436fca9b1 and determined that all issues were resolved and no new issues were identified.

The review team states that overall the codebase is well written and well tested. Prior to the review, the developer briefed the review team on the overarching architecture and threat model they developed. The client took time to accompany modifications to the codebase with supplementary tests.

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	0	0	0
High Risk	0	0	0
Medium Risk	0	0	0
Low Risk	5	5	0
Gas Optimizations	2	1	1
Informational	9	5	4
Total	**16**	**11**	**5**

5 Findings

5.1 Low Risk

5.1.1 Consider using `safeTransferFrom` instead of `transferFrom` with ERC721s

Severity: Low Risk

Context: ERC721TransferLib.sol#L57

Description: Currently, `transferFrom` is used to transfer ERC721s. Given that, in the current context, no impactful reentrancy can be made, it would be better to use `safeTransferFrom`, to make sure that the recipient address can in fact hold/operate with ERC721s.

```
function _reusableTransferCallData(Parties memory parties) private pure returns (bytes memory) {
    return abi.encodeCall(IERC721.transferFrom, (parties.seller, parties.buyer, 0));
}
```

Recommendation: Consider using `safeTransferFrom` instead of `transferFrom`.

divergence: Fixed in PR 46. The fix implements `safeTransfer()` variants for all `ERC721TransferLib` struct types, but does *not* toggle the feature in the `<T>Swapper` implementations. We will defer the decision whether or not to use this functionality as it requires further consideration of the cost-benefit trade off.

Cantina Managed: Fix looks good.

5.1.2 Seller can steal buyer's funds if an offer is created for a to-be-deployed NFT collection

Severity: Low Risk

Context: ERC721TransferLib.sol#L79

Description: Currently, the NFT transfer happens by doing a low-level call to the NFT address and checking the return value

```
for (uint256 end = idSrc + ids.length * 0x20; idSrc < end; idSrc += 0x20) {
    assembly ("memory-safe") {
        mcopy(idDst, idSrc, 0x20)
    }
    (bool success,) = addr.call(reusableCallData);

    if (!success) {
        assembly ("memory-safe") {
            let free := mload(0x40)
            returndatacopy(free, 0, returndatasize())
            revert(free, returndatasize())
        }
    }
```

The problem is that the contract never verifies that `addr.code.length > 0`. Any call made to an address with no code deployed, will always return `true`.

If the swap offer has been created for a to-be-deployed NFT collection with a pre-determined address, the seller could execute the trade without actually transferring the NFT.

Recommendation: Check that `addr.code.length > 0`.

divergence: Fixed in PR 39.

Cantina Managed: Fix looks good.

5.1.3 The `buyer` can grief cancellation

Severity: Low Risk

Context: ConsiderationLib.sol#L101

Description: During a `CANCEL` with ETH as the consideration item, the ETH balance of the contract is sent to the `buyer`, and if the `call` reverts for some reason, it deposits the amount in `escrow`:

```
function _cancel(Consideration memory, PayableParties memory parties, IEscrow escrow) internal {
    // MUST remain as the last step for the same reason as _disburseFunds().
    _sendOrEscrow(parties.buyer, address(this).balance, escrow);
}

function _sendOrEscrow(address payable to, uint256 amount, IEscrow escrow) internal {
    // 20k for a fresh SSTORE and an arbitrary 10k overhead
    (bool success,) = to.call{value: amount, gas: 30_000}("");
    if (success) {
        return;
    }
    escrow.deposit{value: amount}(to);
}
```

We send the funds to escrow in the case of a failed call to prevent the `buyer` from griefing cancellation by intentionally reverting, however, there are a couple other circumstances in which the `buyer` can grief cancellation anyway.

1. Return bomb:

   The `buyer` can create a return bomb by having a `fallback` that returns a very large amount of data. Even though the return data is not explicitly referenced, it still gets copied into memory, and since memory expansion cost grows quadratically, this can cause the `seller` to pay a significant amount of gas, likely causing execution to revert due to an insufficient gas limit.

2. Sending ETH back to the contract: Prior to completing execution, we `assert` that `address(this).balance == 0`. The `buyer` can break this assertion with a `fallback` which sends ETH back to the contract to invalidate this assertion and cause a revert.

Impact: While it's undesirable for any party to be able to grief cancellation, the least concerning outcome is for the `buyer` to have this ability. This is because the `buyer` is the only party that actually sends funds to the contract. It's possible for the `buyer` to delay the swap until it's preferable for them to execute it, but the `seller` always has the ability to revoke their approval(s), effectively cancelling the swap regardless.

Likelihood: This attack vector is considered to be unlikely as the `buyer` must lock their ETH in the contract to be able to execute the attack. Additionally, since the `seller` can effectively cancel by revoking approval(s), it's unlikely that an attempted attack would succeed.

Recommendation: To maintain the post execution invariant of `address(this).balance == 0` and prevent a return bomb, the `buyer` must not receive a callback. The only way to ensure that the `buyer` receives their funds without having a callback is to wrap the ETH into WETH before sending to the `buyer`. In this case, they can't possibly grief execution as they won't have a callback or a way to revert.

Alternatively, we can instead use a pull-based payment wherein the ETH is sent directly to `escrow` instead of the `buyer` whereby the `buyer` can then withdraw the funds at any time without the ability to grief cancellation.

divergence: Fixed in PR 41. Thank you! The first commit of the fix implements attack (2) in a test and a failing CI run against the PR demonstrates as much. The original attempt to send funds to the buyer is now skipped entirely and funds are always sent to escrow, in line with pull-payment best practice. Tests of the return-bomb attack vector revealed that, while present, the attacker's limit of 30k gas significantly capped its own memory expansion such that, even with quadratic costs, aggregate gas was still measured in $\mathcal{O}(10k)$. That said, such a scenario was still undesirable and is also fixed with pull-based payments.

Cantina Managed: Fixed by sending ETH directly to escrow, preventing the `buyer` from receiving a callback, thus preventing the `buyer` from griefing cancellation.

5.1.4 No fork or cross network protection

Severity: Low Risk

Context: *(No context files were provided by the reviewer)*

Description: Addresses are generated using a combination of:

- `SWAP2` address.
- The swap details.
- A salt.

The Swapper contract's address when deployed on one network will be identical to the one deployed on other equivalent EVM networks.

There are rare edge cases where this can be problematic. Namely, when:

- Asset address is the same on two networks (which is the convention used for some cross chain tokens deployed through message passing bridges).
- Seller has same asset ID on two networks (or is confused about which network the high value asset is on).
- Seller approves the asset to trade on the network with a lower value native token.

Above describes a rare set of circumstances that require a specific scenario *and* require user error in order for loss of funds to occur.

Recommendation: Combining chain id with the `CREATE2` salt would mitigate the issue above.

To defend against forks, a check against the chain id at the time of calling `fill` would add additional protection.

divergence: Fixed in PR 40. The current chain ID is now required as an additional constructor argument in the `<T>Swapper` contracts, coupling the ID to the deployment bytecode and hence to the address. This pattern was chosen as it:

- Automatically surfaces, via compiler errors, all other parts of the code that need to change; and
- Avoids overloading parameter intent, which could lead to accidental bugs in future refactors.

Cantina Managed: Confirmed, `chainId` is now used in constructor arguments affecting the `CREATE2` derived address. `chainId` is referenced at the time of deploy for the `fill` or `cancel` contracts meaning the protection extends to chain forks as well.

5.1.5 No input validation on `escrow_`

Severity: Low Risk

Context: SWAP2.sol#L41

Description: User error on deploy could set a `address(0)`, an EOA, or other contract as `escrow_`. In theses cases, attempts to `_cancel` may be DOS'd.

Recommendation: Consider adding:

- An `address(0)` check.
- A `contractId()` function to `Escrow` and calling in the `SWAP2` constructor to enforce correct configuration.

divergence: Fixed in PR 37.

Cantina Managed: Confirmed.

5.2 Gas Optimization

5.2.1 Minimal overflow risk allows for simplified `muldiv` operation

Severity: Gas Optimization

Context: TMPLSwapperBase.tmpl.sol#L44

Description: In `TMPLSwapperBase.constructor`, we compute the fee by applying the `swap.consideration.total` to the fee `basisPoints` using `Math.mulDiv`:

```
uint256 fee = Math.mulDiv(swap.consideration.total, basisPoints, 10_000);
```

`Math.mulDiv` provides the ability to compute any `muldiv` operation which doesn't result in the scaled product overflowing the `uint256` type, even if the initial multiplication would result in an overflow. This is particularly value when computing the result using a high precision denominator, e.g. 1e18.

Since we're computing the fee only using basis points, as long as the `swap.consideration.total` is less than `type(uint256).max / 10000`, we don't risk overflowing. Since having such a high amount of ERC20 tokens or ETH to transfer is highly unlikely, we can generally assume that such a large amount would only be provided as a result of nefarious behavior. For context, Uniswap v2 only supports `uint112` reserves. Additionally, in the event that we do have such a large quantity, we only risk the swap not executing, still allowing for the cancellation to proceed.

Recommendation: Since an overflow is highly unlikely, we can simply compute the fees using plain multiplication and division operations:

```
- uint256 fee = Math.mulDiv(swap.consideration.total, basisPoints, 10_000);
+ uint256 fee = swap.consideration.total * basisPoints / 10_000;
```

divergence: Acknowledged. The `mulDiv()` implementation has short-circuit functionality to return early if the product fits in a single word. The native-Solidity overflow protection thus means that the gas savings in my experiments were only 20 to 30 units, depending on compiler settings. I don't think that such small savings warrant introducing a path that can revert unexpectedly, no matter how improbable.

Cantina Managed: Acknowledged.

5.2.2 Assembly optimizations

Severity: Gas Optimization

Context: ERC721TransferLib.sol#L79-L87, Create2.sol#L25-L35

Description: In the `MultiERC721Token` variant of `ERC721TransferLib._transfer` and `Create2.deploy`, we copy return data from failing calls to free memory prior to referencing this spot in memory to `revert`:

```
// Create2.deploy
assembly ("memory-safe") {
    let free := mload(0x40)

    if iszero(returndatasize()) {
        mstore(free, 0x33d2bae4) // Create2EmptyRevert()
        revert(add(free, 28), 4)
    }

    returndatacopy(free, 0, returndatasize())
    revert(free, returndatasize())
}
```

```
// ERC721TransferLib._transfer
(bool success,) = addr.call(reusableCallData);

if (!success) {
    assembly ("memory-safe") {
        let free := mload(0x40)
        returndatacopy(free, 0, returndatasize())
        revert(free, returndatasize())
    }
}
```

However, we can instead use scratch space (0x00-0x3f) in memory to save some gas while preserving memory safety. Even if the `returndatasize` exceeds 0x3f bytes, since we're immediately reverting, we don't use memory prior to exiting the call context, thus maintaining memory safety.

Recommendation: Use scratch space instead of free memory to allocate return data in the above noted assembly blocks:

```
// Create2.deploy
assembly ("memory-safe") {
    if iszero(returndatasize()) {
        mstore(0, 0x33d2bae4) // Create2EmptyRevert()
        revert(28, 4)
    }

    returndatacopy(0, 0, returndatasize())
    revert(0, returndatasize())
}
```

Note that we can further optimize `ERC721TransferLib._transfer` by writing the entire `for` loop in assembly:

```
// ERC721TransferLib._transfer
assembly ("memory-safe") {
    idSrc := add(ids, 0x20)
    idDst := add(reusableCallData, 0x64)

    for { let end := add(idSrc, mul(mload(ids), 0x20)) } lt(idSrc, end) { idSrc := add(idSrc, 0x20) } {
        mcopy(idDst, idSrc, 0x20)

        if iszero(call(gas(), addr, 0, add(reusableCallData, 0x20), reusableCallData, 0, 0)) {
            returndatacopy(0, 0, returndatasize())
            revert(0, returndatasize())
        }
    }
}
```

divergence: Fixed in PR 43.

Cantina Managed: Fixed as recommended.

5.3 Informational

5.3.1 Neither of the parties should be able to freely choose the `salt`

Severity: Informational

Context: ET.sol#L75

Description: The way currently Swaps work is by giving approval to a pre-determined address. After both parties have given approval, the contract can be deployed at said address and the trade is executed.

It all works well as long as the `salt` is generated outside of either party's control.

If either of them can choose them, they can generate `2**82` addresses with different salts and then generate `2**82` EOAs. With this number of addresses, they have ~99.96% chance of collision.

Then, they can give the `salt` which results in the address for which they've found the collision. As soon as the other party gives approval, the malicious party can drain it.

Note: at the time of writing, such attack is practically impossible. On paper, UniswapV3's router is also 'vulnerable' to such attack.

Recommendation: Always use a randomly generated `salt`.

divergence: Thank you for the quantification of the probability, and the context around UniswapV3. Please note that there is a discussion and explicit warning about this in the design document of the audit commit, which is the reason for the `<T>SwapperProposer` being developed as one possible approach to ensuring a random salt. That said, I appreciate why this should be called out in the audit.

Cantina Managed: Acknowledged.

5.3.2 Missing deadline

Severity: Informational

Context: *(No context files were provided by the reviewer)*

Description: While swap logic in the system allows for cancellation, without explicitly cancelling the swap or revoking approvals, it's possible to execute the swap at any time in the future. Over time, if the swap is not cancelled, conditions may become more favorable for one party, i.e. the market price of the NFT increases or decreases, making buying or selling more profitable.

Recommendation: Include a `deadline` in all variations of the `TMPLSwap` struct to be enforced in the `action == FILL` case.

divergence: Fixed in PR 47.

Cantina Managed: Fixed by including a `notValidAfter` field in the `TMPLSwap` struct variations and reverting during fill if the current `block.timestamp` is greater than `notValidAfter`.

5.3.3 Opcode incompatibilities between networks

Severity: Informational

Context: *(No context files were provided by the reviewer)*

Description: The codebase uses recent versions of the Solidity compiler as well as the EVM. Along with this usage of recent versions are opcodes which are not universally supported by all EVM networks. Those opcodes include: `mcopy`, `tstore`, `tload`, and `push0`.

Deploying the contracts to networks which don't support these opcodes will cause deployment to revert. Modifying the bytecode for support or accidental usage of account nonce during reverted deployment can break expectations of deterministic contract addresses. This can be particularly damaging for, e.g. the `propose` mechanism, whereby it's imperative that the pre-computed contract address matches the actual address, even across networks.

Recommendation: Be aware of opcode incompatibilities across EVM networks and carefully plan deployment and versioning accordingly, with a focus on consistent contract bytecode and deterministic addresses where necessary. The linked table provides an overview of opcode support across EVM networks.

divergence: Acknowledged. Please see commit `026e2b44fe6b2d481c37280447cc973c32ef7588`, demonstrating failure to deploy `SWAP2Proposer` on an earlier EVM version (Geth v1.10.26 `SimulatedBackend`), with the specific error being the lack of `PUSH0` support. Removing the opcode would necessitate use of the paris EVM version, also breaking `T{STORE,LOAD}` support and the core `ET` contracts—any developer refactoring the protocol to achieve this would be working well beyond the scope of this audit.

Cantina Managed: Acknowledged. It is correct that deployment will fail without both changing the EVM version and modifying the code accordingly to be supported, which is a very low likelihood operational risk.

5.3.4 Sellers can approve and revoke approvals prior to `fill` being called

Severity: Informational

Context: TMPLSwapperBase.tmpl.sol#L41

Description: A minor annoyance, and gas costs, may be incurred if a seller approves and then revokes approval prior to fills executing.

There is no loss of funds to the purchaser and, at most, the gas costs associated with a single failed `fill` attempt would occur.

Recommendation: Handle in the UI by checking that approval state has not changed before `fill` is called by the purchaser.

divergence: Acknowledged. This surface is common to all protocols that perform atomic swaps and a check will be included in the UI.

Cantina Managed: Acknowledged.

5.3.5 Emit event for platform fee changes

Severity: Informational

Context: Global scope

Description: Events are useful for users to monitor important modifications to the protocol params. When platform fees are updated, no event is emitted.

Recommendation: Emit an event whenever the platform fee changes.

divergence: Fixed in PR 45.

Cantina Managed: Confirmed.

5.3.6 *Magic number* discouraged

Severity: Informational

Context: *(No context files were provided by the reviewer)*

Description/Recommendation: In lieu of inlining the constant, declare it instead with `uint256 constant BASIS_POINTS_DENOMINATOR = 10_000;`. See Magic number (programming) for more context.

divergence: Acknowledged. Although I agree that magic numbers should be avoided as a rule of thumb, the adjacent variable name was deliberately chosen as `basisPoints` to provide the necessary context. By analogy, if fees were calculated as a percentage, I would use a magic number of 100 without explanation. I believe that a "*basis point*" is a sufficiently well-known term that no change is necessary here.

Cantina Managed: Acknowledged.

5.3.7 Consider Enabling `via-ir`

Severity: Informational

Context: Global scope

Description: There are optimizations the compiler will apply when `via-ir` is enabled. Micro optimizations such as replacing `i++` with `++i` can be omitted when building with `via-ir`.

Recommendation: Enable `via-ir` in the `foundry.toml` file.

For time efficiency when developing, a local or dev profile can be used to selectively disable `via-ir`. *(See the Foundry book for more information).*

```
[profile.local]
via_ir = false
```

divergence: Fixed in PR 42. We have explicitly disabled `via_ir` in the *default* profile and introduced a dedicated *deploy* profile that the deployment scripts confirm in their `setUp()`. This avoids accidentally deploying on a development-specific profile.

Canitna Managed: Confirmed.

5.3.8 Consider Verbose Function Names

Severity: Informational

Context: TMPLSwapperDeployer.tmpl.sol#L39, TMPLSwapperDeployer.tmpl.sol#L52, TMPLSwapperDeployer.tmpl.sol#L75

Description: Since the `TMPL` pattern is already in use, consider adding more verbose function overrides for each of `fill`, `cancel`, `swapper`, `propose`.

Current function signatures rely on the different tuples to differentiate themselves:

```
"fill(((address,address),(address,uint256),((address,uint256)[],uint256,uint256)),bytes32)": "94fcf766",
"fill(((address,address),(address,uint256),((address,uint256)[],uint256,uint256,address)),bytes32)":
↪  "91485487",
"fill(((address,address),(address,uint256[])[],((address,uint256)[],uint256,uint256)),bytes32)":
↪  "d5a09918",
"fill(((address,address),(address,uint256[])[],((address,uint256)[],uint256,uint256,address)),bytes32)"
↪  :
↪  "ed3274b7"
```

Integrating with the system, in a UI for example, could benefit from verbose names rather than the types alone. No issues identified related to above.

Recommendation: A more verbose approach would be to rename the functions:

- `fillTMPL`.
- `cancelTMPL`.
- `swapperTMPL`.
- `proposeTMPL`.

divergence: Fixed in PR 44.

Cantina Managed: Confirmed.

5.3.9 Function argument shadows state variable

Severity: Informational

Context: SWAP2.sol#L114

Description: The `SWAP2` constructor argument `Escrow escrow` shadows an inherited state variable from `SWAP2Deployer`.

Recommendation: Avoid variable shadowing by considering the pattern used in the `SWAP2Deployer` constructor of adding a suffix `Escrow escrow_`.

For more information see Shadowing State Variables.

divergence: Fixed in PR 37.

Canitna Managed: Confirmed.