Swivel contest
Findings & Analysis Report

2021-11-05

Table of contents

Overview

About C4

Code 432n4 (C4) is an open organization consisting of security researchers, auditors, developers, and individuals with domain expertise in smart contracts.

A C4 code contest is an event in which community participants, referred to as Wardens, review, audit, or analyze smart contract logic in exchange for a bounty provided by sponsoring projects.

During the code contest outlined in this document, C4 conducted an analysis of Swivel contest smart contract system written in Solidity. The code contest took place between September 30—October 6 2021.

Wardens

15 Wardens contributed reports to the Swivel contest code contest:

- 0xsanson
- 0xRajeev
- cmichel
- defsec
- GalloDaSballo
- JMukesh
- leastwood
- loop
- nikitastupin

- pants
- pauliax
- itsmeSTYJ
- gpersoon
- yeOlde
- csanuragjain

This contest was judged by 0xean.

Final report assembled by CloudEllie and itsmetechjay.

Summary

The C4 analysis yielded an aggregated total of 27 unique vulnerabilities and 76 total findings. All of the issues presented here are linked back to their original finding.

Of these vulnerabilities, 4 received a risk rating in the category of HIGH severity, 5 received a risk rating in the category of MEDIUM severity, and 18 received a risk rating in the category of LOW severity.

C4 analysis also identified 20 non-critical recommendations and 28 gas optimizations.

Scope

The code under review can be found within the C4 Swivel contest code repository, is composed of 3 smart contracts written in the Solidity programming language, and includes 996 lines of Solidity code.

Severity Criteria

C4 assesses the severity of disclosed vulnerabilities according to a methodology based on OWASP standards.

Vulnerabilities are divided into three primary risk categories: high, medium, and low.

High-level considerations for vulnerabilities span the following key areas when conducting assessments:

- Malicious Input Handling
- Escalation of privileges
- Arithmetic
- Gas use

Further information regarding the severity criteria referenced throughout the submission review process, please refer to the documentation provided on [the C4 website](#).

High Risk Findings (4)

[H-01] Unsafe handling of underlying tokens

Submitted by 0xsanson, also found by 0xRajeev, cmichel, defsec, GalloDaSballo, JMukesh, leastwood, loop, nikitastupin, pants, and pauliax.

Impact

Not every ERC20 token follows OpenZeppelin's recommendation. It's possible (inside ERC20 standard) that a `transferFrom` doesn't revert upon failure but returns `false`.

The code doesn't check these return values. For example `uToken.transferFrom(msg.sender, o.maker, a);` in `initiateVaultFillingZcTokenInitiate` can be exploited by the msg.sender to initiate a trade without sending any underlying.

Proof of Concept

```
grep 'transfer' Swivel.sol
```

Tools Used

editor

Recommended Mitigation Steps

Consider using [OpenZeppelin's library] with *safe* versions of transfer functions.

[H-02] Swivel: Taker is charged fees twice in exitVaultFillingVaultInitiate

Submitted by itsmeSTYJ, also found by gpersoon.

Impact

Taker is charged fees twice in `exitVaultFillingVaultInitiate()` . Maker is transferring less than premiumFilled to taker and then taker is expected to pay fees i.e. taker's net balance is premiumFilled - 2*fee

Recommended Mitigation Steps

```
function exitVaultFillingVaultInitiate(Hash.Order calldata o, ui
  bytes32 hash = validOrderHash(o, c);

  require(a <= (o.principal - filled[hash]), 'taker amount > ava

  filled[hash] += a;

  uint256 premiumFilled = (((a * 1e18) / o.principal) * o.premiu
  uint256 fee = ((premiumFilled * 1e18) / fenominator[3]) / 1e18

  Erc20 uToken = Erc20(o.underlying);
  // transfer premium from maker to sender
  uToken.transferFrom(o.maker, msg.sender, premiumFilled);

  // transfer fee in underlying to swivel from sender
  uToken.transferFrom(msg.sender, address(this), fee);

  // transfer <a> vault.notional (nTokens) from sender to maker
  require(MarketPlace(marketPlace).p2pVaultExchange(o.underlying

  emit Exit(o.key, hash, o.maker, o.vault, o.exit, msg.sender, a
}
```

[JTraversa (Swivel) confirmed]

Oxean (judge) commented:

> Based on
>
> ```
> 3 — High: Assets can be stolen/lost/compromised directly (or inc
> ```
>
> This is being upgraded to a high risk. The duplicate of it was at that level by the submitting warden and considering that fees are being incorrectly taken from the taker and not the maker, the maker ends up with a higher balance than expected and the taker has no way to recoup these fees (assets are now lost).

JTraversa (Swivel) commented:

> Is that how it is interpreted? I'd assume that high risk would imply a valid attack path that a user could use to drain deposited funds based on that description.
>
> I won't fight this one obviously, just think there's a *clear* differentiation between this and the other high risk issue.

[H-03] `transferNotionalFrom` doesn't check `from != to`

Submitted by gpersoon, also found by cmichel.

Impact

The function `transferNotionalFrom` of `VaultTracker.sol` uses temporary variables to store the balances. If the "from" and "to" address are the same then the balance of "from" is overwritten by the balance of "to". This means the balance of "from" and "to" are increased and no balances are decreased, effectively printing money.

Note: `transferNotionalFrom` can be called via `transferVaultNotional` by everyone.

Proof of Concept

https://github.com/Swivel-Finance/gost/blob/v2/test/vaulttracker/VaultTracker.sol#L144-L196

```
function transferNotionalFrom(address f, address t, uint256 a)
Vault memory from = vaults\[f];
Vault memory to = vaults\[t];
...
vaults\[f] = from;
...
vaults\[t] = to;    // if f==t then this will overwrite vaults
```

https://github.com/Swivel-Finance/gost/blob/v2/test/marketplace/MarketPlace.sol#L234-L238

```
function transferVaultNotional(address u, uint256 m, address t
require(VaultTracker(markets\[u]\[m].vaultAddr).transferNotior
```

Tools Used

Recommended Mitigation Steps

Add something like the following: `require (f != t,"Same");`

JTraversa (Swivel) confirmed

[H-04] return value of 0 from ecrecover not checked

Submitted by gpersoon, also found by 0xRajeev, cmichel, and nikitastupin.

Impact

The solidity function `ecrecover` is used, however the error result of 0 is not checked for. See documentation: https://docs.soliditylang.org/en/v0.8.9/units-and-global-variables.html?highlight=ecrecover#mathematical-and-cryptographic-functions "recover the address associated with the public key from elliptic curve signature or return zero on error. "

Now you can supply invalid input parameters to the `Sig.recover` function, which will then result 0. If you also set `o.maker` to be 0 then this will match and an invalid signature is not detected.

So you can do all kinds of illegal & unexpected transactions.

Proof of Concept

https://github.com/Swivel-Finance/gost/blob/v2/test/swivel/Swivel.sol#L476-L484

```
function validOrderHash(Hash.Order calldata o, Sig.Components
...
require(o.maker == Sig.recover(Hash.message(domain, hash), c),
return hash;
}
```

https://github.com/Swivel-Finance/gost/blob/v2/test/swivel/Sig.sol#L16-L23

```
function recover(bytes32 h, Components calldata c) internal pu
...
return ecrecover(h, c.v, c.r, c.s);
```

Tools Used

Recommended Mitigation Steps

Verify that the result from `ecrecover` isn't 0

JTraversa (Swivel) acknowledged JTraversa (Swivel) commented:

> Id say this is noteable, but because all actions require approvals from o.maker, having 0x00 as o.maker with an "invalid" but valid signature should not be impactful. The suggestion would be to filter 0x00 makers from the orderbook? (which we do)

Medium Risk Findings (5)

[M-01] Admin is a single-point of failure without any mitigations

Submitted by 0xRajeev, also found by 0xsanson and leastwood.

Impact

Admin role has absolute power across Swivel, Marketplace and VaultTracker contracts with several `onlyOwner` functions. There is no ability to change admin to a new address or renounce it which is helpful for lost/compromised admin keys or to delegate control to a different governance/DAO address in future.

The project does not use the widely used OpenZeppelin Ownable library which provides transfer/renounce functions to mitigate such compromised/accidental situations with admin keys. This makes admin role/key a single-point of failure.

Proof of Concept

- https://github.com/Swivel-Finance/gost/blob/5fb7ad62f1f3a962c7bf5348560fe88de0618bae/test/marketplace/MarketPlace.sol#L38
- https://github.com/Swivel-Finance/gost/blob/5fb7ad62f1f3a962c7bf5348560fe88de0618bae/test/swivel/Swivel.sol#L43
- https://github.com/Swivel-Finance/gost/blob/5fb7ad62f1f3a962c7bf5348560fe88de0618bae/test/vaulttracker/VaultTracker.sol#L27

Tools Used

Manual Analysis

Recommended Mitigation Steps

Ensure admins are reasonably redundant/independent (3/7 or 5/9) multisigs and add transfer/renounce functionality for admin. Consider using OpenZeppelin's Ownable library.

JTraversa (Swivel) acknowledged:

> Similar to some other suggestions, i'm personally not sure if this is within the scope of the competition / an "issue". We have immediate plans to add admin

> transfer functionality however left it out of the scope of this audit as not having a transferAdmin function doesn't seem to be any more dangerous than the admin having the functionality it currently has.

> That said, in production the admin will both be a multisig and there is also basic admin timelock/transfer functionality, with on-chain governance as opposed to centralized-ish multisigs coming Q1.

[M-02] Missing event & timelock for critical `onlyAdmin` functions

Submitted by 0xRajeev, also found by defsec.

Impact

`onlyAdmin` functions that change critical contract parameters/addresses/state should emit events and consider adding timelocks so that users and other privileged roles can detect upcoming changes (by offchain monitoring of events) and have the time to react to them.

Privileged functions in all contracts, for e.g. `VaultTracker onlyAdmin`, have direct financial or trust impact on users who should be given an opportunity to react to them by exiting/engaging without being surprised when changes initiated by such functions are made effective opaquely (without events) and/or immediately (without timelocks).

See similar Medium-severity finding in ConsenSys's Audit of 1inch Liquidity Protocol (https://consensys.net/diligence/audits/2020/12/1inch-liquidity-protocol/#unpredictable-behavior-for-users-due-to-admin-front-running-or-general-bad-timing)

Proof of Concept

- https://github.com/Swivel-Finance/gost/blob/5fb7ad62f1f3a962c7bf5348560fe88de0618bae/test/vaulttracker/VaultTracker.sol#L36-L59
- https://github.com/Swivel-Finance/gost/blob/5fb7ad62f1f3a962c7bf5348560fe88de0618bae/test/vaulttracker/VaultTracker.sol#L70-L98

- https://github.com/Swivel-Finance/gost/blob/5fb7ad62f1f3a962c7bf5348560fe88de0618bae/test/vaulttracker/VaultTracker.sol#L102-L129
- https://github.com/Swivel-Finance/gost/blob/5fb7ad62f1f3a962c7bf5348560fe88de0618bae/test/vaulttracker/VaultTracker.sol#L132-L138
- https://github.com/Swivel-Finance/gost/blob/5fb7ad62f1f3a962c7bf5348560fe88de0618bae/test/vaulttracker/VaultTracker.sol#L144-L196
- https://github.com/Swivel-Finance/gost/blob/5fb7ad62f1f3a962c7bf5348560fe88de0618bae/test/vaulttracker/VaultTracker.sol#L201-L239

and others

Tools Used

Manual Analysis

Recommended Mitigation Steps

Add events to all possible flows (some flows emit events in callers) and consider adding timelocks to such onlyAdmin functions.

Oxean (judge) commented:

> removing duplicate mark since the (valid) suggestion to emit events on the changes made by admin calls

[M-03] Previously created markets can be overwritten

Submitted by 0xRajeev, also found by 0xsanson, cmichel, gpersoon, itsmeSTYJ, and pauliax.

Impact

The `createMarket` function allows accidental overwriting of previously created markets for the same combination of underlying and maturity timestamp (u, m)

because there is no zero-address check to see if a previously created market exists for that combination.

Proof of Concept

- https://github.com/Swivel-Finance/gost/blob/5fb7ad62f1f3a962c7bf5348560fe88de0618bae/test/marketplace/MarketPlace.sol#L65
- https://github.com/Swivel-Finance/gost/blob/5fb7ad62f1f3a962c7bf5348560fe88de0618bae/test/marketplace/MarketPlace.sol#L53-L70

Tools Used

Manual Analysis

Recommended Mitigation Steps

Add a zero-address check for `markets\[u]\[m]` in `createMarket` before writing to it

[M-04] fee-on-transfer underlying can cause problems

Submitted by 0xsanson.

Impact

The current implementation doesn't work with fee-on-transfer underlying tokens. Considering that Compound can have these kind of tokens (ex. USDT can activate fees), this issue can affect the protocol.

The problem arises when transferring tokens, basically blocking all functions in `Swivel.sol` for that particular token, since the contract wrongly assumes balances values. This becomes particularly problematic in the following scenario: a market for USDT is running without problems, then they activate the fee: this effectively blocks users from redeeming the underlying.

Proof of Concept

`grep 'transfer' Swivel.sol` for a complete list of affected lines (basically every `tranfer` or `transferFrom` of underlying tokens). Also `grep 'redeemUnderlying' Swivel.sol`.

For example:

```
require(CErc20(mPlace.cTokenAddress(u, m)).redeemUnderlying(re
// transfer underlying back to msg.sender
Erc20(u).transfer(msg.sender, redeemed);
```

This would fail (revert) since the contract would have received less than `redeemed` tokens.

Tools Used

editor

Recommended Mitigation Steps

If the protocol wants to use all possible Compound tokens, a way to handle these tokens must be implemented. A possible way to do it is to check the balance of the contract before and after every time a token is transferred to see the effective quantity. To help keeping the code clear, a function like Compound's `doTransferIn` can be implemented.

JTraversa (Swivel) acknowledged:

> Will review further. I dont believe that any tokens on compound currently have fees. Although it *is* news to me that USDT has toggle-able fees, whoops.

> That said, given we have admin control over added assets, I'd probably also lower this to a low-risk if accepted.

Oxean (judge) commented:

```
2 — Med: Assets not at direct risk, but the function of the prot
```

> based on this “leaking value” I would say it qualifies as a med-severity.

JTraversa (Swivel) commented:

> We can account for the transfers in with a similar balance before transferFrom, and balance after check, in order to prevent additional deposits after a fee has been turned on.

> That said, Im not sure we can account for `redeemUnderlying` easily because should a fee be turned on, all funds would just be stuck in our contract if we added a similar check? (a != balance2-balance1)

> If a fee is turned on for USDT markets, there would be lost fee value, so if adding a check wouldn’t work, the most reasonable response is just to make sure the market is pausable.

[M-05] Swivel: implementation for initiateZcTokenFillingZcTokenExit is incorrect

Submitted by itsmeSTYJ.

Impact

In `initiateZcTokenFillingZcTokenExit()` , this comment `// transfer underlying tokens - the premium paid + fee in underlying to swivel (from sender)` is incorrect because you are actually transferring the underlying tokens - premium paid to the maker (from sender) AND you have to pay fee separately to swivel.

`initiateZCTokenFillingZcTokenExit` means I want to sell my nTokens so that means `a` is the amount of principal I want to fill. Let’s use a hypothetical example where I (taker) wants to fill 10 units of ZcTokenExit for maker.

1. I transfer 10 units of underlying to Swivel. The net balances are: me (-a), swivel (+a)
2. I transfer fee (in underlying) to Swivel. The net balances are: me (-a-fee), swivel (+a+fee)
3. Swivel initiates my position, sends me the ZcToken and sends Maker the nTokens

4. Maker pays me premiumFilled for the nTokens. The net balances are: me (-a-fee+premiumsFilled), swivel (+a+fee), maker (-premiumsFilled)
5. Maker closes position. The net balances are: me (-a-fee+premiumsFilled), swivel (+fee), maker (-premiumsFilled+a)

So effectively, I (taker) should be paying a-premium to maker and fee to swivel.

Recommended Mitigation Steps

```
function initiateZcTokenFillingZcTokenExit(Hash.Order calldata c
  bytes32 hash = validOrderHash(o, c);

  require(a <= o.principal - filled[hash]), 'taker amount > avai

  filled[hash] += a;

  uint256 premiumFilled = (((a * 1e18) / o.principal) * o.premiu
  uint256 fee = ((premiumFilled * 1e18) / fenominator[0]) / 1e18

  // transfer underlying tokens - the premium paid in underlying
  Erc20(o.underlying).transferFrom(msg.sender, o.maker, a - prem
  Erc20(o.underlying).transferFrom(msg.sender, swivel, fee);
  // transfer <a> zcTokens between users in marketplace
  require(MarketPlace(marketPlace).p2pZcTokenExchange(o.underlyi

  emit Initiate(o.key, hash, o.maker, o.vault, o.exit, msg.sende
}
```

JTraversa (Swivel) confirmed:

> Really good eye.

> We had `o.maker` send the fee after receiving it (same # of transfers as suggested mitigation) and actually accidentally lost that in a larger organizational commit 😅
> .

Low Risk Findings (18)

- **[L-01] Different parameter used in while emitting event** *Submitted by JMukesh, also found by pauliax.*

- [L-02] Swivel Markets are not Isolated *Submitted by leastwood.*
- [L-03] Missing zero-address checks *Submitted by 0xRajeev, also found by cmichel, defsec, GalloDaSballo, JMukesh, and pants.*
- [L-04] Missing input validation, threshold check, event and timelock in `setFee` function *Submitted by 0xRajeev, also found by 0xsanson, defsec, JMukesh, and pauliax.*
- [L-05] Compact signatures not being supported could lead to DoS *Submitted by 0xRajeev, also found by pauliax.*
- [L-06] Missing input validation on array length match *Submitted by 0xRajeev, also found by JMukesh, and leastwood.*
- [L-07] Abstract contracts should really be interfaces *Submitted by 0xRajeev.*
- [L-08] Static `chainID` could allow replay attacks on chain splits *Submitted by 0xRajeev, also found by 0xsanson, nikitastupin, and pauliax.*
- [L-09] Swivel: Incorrect dev comments for the 4 initiate functions *Submitted by itsmeSTYJ.*
- [L-10] Open TODOs in Codebase *Submitted by leastwood, also found by pants and yeOlde.*
- [L-11] Complex state variable copied to memory in `redeemZcToken` (MarketPlace.sol) *Submitted by yeOlde.*
- [L-12] swivel and marketPlace contract does not implement the mechanisim to renounce the role of admin *Submitted by JMukesh, also found by GalloDaSballo and pants.*
- [L-13] Missing event and timelock for `setSwivelAddress` *Submitted by 0xRajeev.*
- [L-14] Validations in `setFee` *Submitted by pauliax.*
- [L-15] Missing input validation & event in emergency `blockWithdrawal` could be risky *Submitted by 0xRajeev.*
- [L-16] `createMarket` function missing parameter description *Submitted by loop.*
- [L-17] Wrong parameter name used in function spec *Submitted by loop.*
- [L-18] Swivel: Implement check effect interaction to align with best practices *Submitted by itsmeSTYJ.*

Non-Critical Findings (20)

- [N-01] Potential Reentrancy when Initiating and Exiting Positions *Submitted by leastwood, also found by pants.*
- [N-02] Lack of Pause Mechanism*Submitted by leastwood.*
- [N-03] Use of `ecrecover` is susceptible to signature malleability *Submitted by OxRajeev, also found by nikitastupin.*
- [N-04] Lack of Proper Revert Messages *Submitted by leastwood, also found by pants.*
- [N-05] `balanceOf` should be a `_view_` function *Submitted by Oxsanson.*
- [N-06] `Swivel.sol` - marketplace is an immutable address, yet is always casted to MarketPlace - store as MarketPlace to make code cleaner *Submitted by GalloDaSballo.*
- [N-07] The requires used in `p2pVaultExchange` `transferVaultNotional` in Marketplace.sol are not necessary *Submitted by GalloDaSballo.*
- [N-08] Magic Number `1e26` would best replaced by a constant in `VaultTracker` *Submitted by GalloDaSballo, also found by itsmeSTYJ.*
- [N-09] Missing Dev Comments *Submitted by leastwood.*
- [N-10] Double Spending. No `decreaseAllowance()/ IncreaseAllowance()` *Submitted by pants.*
- [N-11] Return value of `transferNotionalFee` *Submitted by pauliax.*
- [N-12] Can cancel the same order again *Submitted by pauliax.*
- [N-13] Style issues *Submitted by pauliax.*
- [N-14] Missing input validation may cause revert due to underflow *Submitted by OxRajeev.*
- [N-15] Wrong yield computation upon maturity *Submitted by cmichel.*
- [N-16] Missing guarded launch *Submitted by OxRajeev.*
- [N-17] Underlying can be fetched from `cToken` *Submitted by pauliax.*
- [N-18] Missing initial ownership event *Submitted by pants.*
- [N-19] Return value of `transferNotionalFee` ignored *Submitted by loop.*
- [N-20] Prevent underflow in require *Submitted by gpersoon.*

Gas Optimizations (28)

- [G-01] Gas Optimization on the Public Functions *Submitted by defsec, also found by 0xRajeev, loop, and yeOlde.*
- [G-02] Caching state variables in local/memory variables avoids SLOADs to save gas *Submitted by 0xRajeev, also found by pauliax.*
- [G-03] Removing redundant require() can save gas *Submitted by 0xRajeev.*
- [G-04] += can be replaced by = *Submitted by 0xRajeev.*
- [G-05] Converting fenominator to a static array will save storage slots and gas *Submitted by 0xRajeev, also found by defsec, loop, and pauliax.*
- [G-06] Avoiding initialization of loop index can save a little gas *Submitted by 0xRajeev.*
- [G-07] Bounded array lengths or checking `gasleft` will save gas from OOGs*Submitted by 0xRajeev.*
- [G-08] Better Math in `calculateReturn` *Submitted by 0xsanson.*
- [G-09] Math's operations order in Swivel's functions *Submitted by 0xsanson, also found by cmichel, itsmeSTYJ, and loop.*
- [G-10] `require(mPlace.custodialExit)` in Swivel.sol is redundant *Submitted by GalloDaSballo.*
- [G-11] Redundant `require` in Swivel.sol *Submitted by GalloDaSballo.*
- [G-12] Gas: Approve `cToken` address only once for underlying *Submitted by cmichel.*
- [G-13] Use `bytes32` rather than `string/bytes` *Submitted by defsec.*
- [G-14] VaultTracker.sol: Gas optimisation for `addNotional` *Submitted by itsmeSTYJ.*
- [G-15] MarketPlace.sol: Remove maturity from VaultTracker and ZcToken *Submitted by itsmeSTYJ.*
- [G-16] VaultTracker.sol: pass in exchangeRate as a variable to `matureVault()` *Submitted by itsmeSTYJ.*
- [G-17] VaultTracker.sol: `init sVault.exchangeRate` in constructor *Submitted by itsmeSTYJ.*
- [G-18] Gas Savings Upon Market Creation *Submitted by leastwood.*

- [G-19] Array `.length` Used Directly In For Loops *Submitted by yeOlde, also found by pants.*
- [G-20] Title: Double reading from `calldata o` *Submitted by pants.*
- [G-22] Functions returning boolean *Submitted by pauliax.*
- [G-23] 'matured' can be replaced by 'maturityRate' > 0 *Submitted by pauliax.*
- [G-24] matureVault can receive maturityRate cheaper *Submitted by pauliax.*
- [G-25] Input validation on `amount > 0` will save gas *Submitted by 0xRajeev, also found by csanuragjain.*
- [G-26] Bytes constant more efficient than string literal *Submitted by loop.*
- [G-27] 'mature' and 'maturityRate' do not need separate mappings *Submitted by pauliax.*
- [G-28] 'onlyAdmin' and 'onlySwivel' modifiers *Submitted by pauliax.*

Disclosures

C4 is an open organization governed by participants in the community.

C4 Contests incentivize the discovery of exploits, vulnerabilities, and bugs in smart contracts. Security researchers are rewarded at an increasing rate for finding higher-risk issues. Contest submissions are judged by a knowledgeable security researcher and solidity developer and disclosed to sponsoring developers. C4 does not conduct formal verification regarding the provided code but instead provides final verification.

C4 does not provide any guarantee or warranty regarding the security of this project. All smart contract software should be used at the sole risk and responsibility of users.

Top