Solidly V2 Audit

Scope: LP Safety

Main Contract Deposit/Withdrawal Flow

- **Approve individual tokens on router**
  - **Goals**
    - User cannot be tricked into malicious approvals
  - This is a web concern, not smart contract. Need to be certain that users are always approving on the correct address.
    - Recommend using a solution similar to yearn's allowlist if possible
- **Deposit tokens to LP pair via router**
  - **Goals**
    - Router cannot divert/steal tokens in transit
      - `addLiquidity` is atomic, including the `safeTransfer` of LP funds back to user, so funds should not get stuck in router
  - Router properly creates the correct LP token in correct proportions
  - **Contracts**
    - `BaseV2Router01`, in `BaseV2-periphery.sol`
      - Utilizes either `addLiquidity()` or `addLiquidityETH()`, formerly known as `addLiquidityFTM()`
      - Inherits from `SolidlyImplementation`
      - **Notes**
        - No changes in logic for `addLiquidity()` or `addLiquidityFTM()` from SolidlyV1. `addLiquidityETH()` is the same but with FTM swapped for ETH.
        - Useful for helping understand the slot storage system:
          - How can we update slot values that are supposedly constant with assembly?
            - https://ethereum.stackexchange.com/questions/133546/why-is-constant-variable-value-changing
          - Why do we use `bytes32`?
            - https://ethereum.stackexchange.com/questions/11770/what-is-the-difference-between-bytes-and-bytes32
      - `BaseV2-periphery.sol` diff vs Solidly v1 can be found here
        - For comparison, documents were matched for linting and usage of uint/uint256
    - `BaseV2Pair`, in `BaseV2-core.sol`
      - Inherits from `SolidlyChildImplementation` and `SolidlyImplementation`
    - `BaseV2Factory`, in `BaseV2-core.sol`
      - Inherits from `SolidlyFactory` and `SolidlyImplementation`

- **Issue (low):** `SolidlyChildProxy` assumes we have a `childInterfaceAddress` in the factory (`msg.sender`), but factory does not require a non-zero `childInterfaceAddress` before deploying child proxies.
  - **Recommendation:** Add `initialize()` function to `SolidlyFactory`, and require storage slots to be updated (initialized) before factory can deploy child proxies. Similarly, manually updating the various storage slots on any new pair would be a major headache/time committment, so we should just require that the factory has these updated before it is allowed to create child proxies.
- **Issue (low):** Inconsistency in usage of WETH vs WFTM. For instance, `BaseV1Router01 initialize()` mixes usage of weth and wftm.
  - **Recommendation:** Either migrate completely to WETH in the code (everywhere) or stick with WFTM. The latter would offer fewer chances for regressions and make code diff simpler, but may result in minor confusion for users browsing Etherscan.
- **Minor:** L227 of `BaseV2-Core.sol` is redundant, as `_unlocked` is set equal to 1 when declared as a storage variable.

```
138     // simple re-entrancy check
139     uint256 internal _unlocked = 1;
140
```

  - **Recommendation:** Delete L227 in `initialize()` function of `BaseV1Pair`

```
222     function initialize(
223         address _token0,
224         address _token1,
225         bool _stable
226     ) external notInitialized {
227         _unlocked = 1;
```

  - **Notes:**
    - Reentrancy `lock()` modifier
      - useful for understanding what `_;` does in modifiers
        - https://medium.com/coinmonks/the-curious-case-of-in-solidity-16d9eb4440f1
      - Simple reentrancy example
        - https://solidity-by-example.org/hacks/re-entrancy/

Approve individual tokens on router

- **Goals**
  - User cannot be tricked into malicious approvals
- This is a web concern, not smart contract. Need to be certain that users are always approving on the correct address.
  - Recommend using a solution similar to yearn's allowlist if possible

Deposit LP token to gauge

- **Goals**
  - Simple deposit function
  - Check for accounting issues
    - Any place `balanceOf[account]` or `stake` is called, especially anywhere a write occurs to `balanceOf[account]`.
      - `balanceOf[account]` only updated on deposit and withdrawal.
- **Contracts**
  - `GaugeV2` in `BaseV2-gauges.sol`
- **Notes**
  - LP token referred to as `stake`
    - Set via `initialize()` and cannot be updated again later
      - Proper `notInitialized()` in `SolidlyImplementation` especially important here, as malicious gov could brick user funds by updating `stake` token
- **Minor:** Users are able to `optIn` to the LP token as a rewards pool. If an attacker found a way to `notifyRewardAmount()` for the LP then they could potentially drain some LP tokens or wreck accounting.
  - Recommendation: Although `notifyRewardAmount()` cannot be called on the LP token `stake`, for extra security add `require(token != stake, "Invalid reward token");` to the first line of `_optIn()`.
- **Minor:** `withdrawToken()` does not implement a check to ensure that `amount <= balanceOf[msg.sender]`.
  - Recommendation: Although technically this is fine since Solidity version is > 0.8.x with default safeMath, for clarity of code and added security for any code reusage or forks, recommend adding an explicit `require()` statement that `amount <= balanceOf[msg.sender]`.

Withdraw LP token from gauge

- **Goals**
  - No one except for user can withdraw their LP tokens (`stake`)
    - Cannot be clawed back
    - Only transfer occurs on deposit and withdrawal
  - User LPs can exit even if issues occur with rewards
  - No reentry (user gets more `stake` than they should)
    - `WithdrawToken()` has `lock()` modifier
- **Contracts**
  - `GaugeV2` in `BaseV2-gauges.sol`
- **Issue (high):** Gauge depends on voter for withdrawals, even in emergencies. A malicious voter can lock all user funds, seemingly only for the purpose of an event.
  - **Commentary Solidly Labs: This could not be performed by any random user but only by the Solidly multisig. It assumes the multisig to act maliciously. The issue was fixed anyhow.**

- Description: Before completing a successful withdrawal, the gauge calls `emitWithdraw()` on the `BaseV2Voter`, which ensures that the gauge is, in fact, a gauge, and also emits another `Withdraw` event, with the gauge address added on.

```
651     /**
652      * @notice Withdraws LP tokens, and detaches veNFT from the gauge if specified
653      * @param amount The amount of LP to withdraw
654      * @param tokenId The veNFT to detach, input 0 to skip detachment
655      */
656     function withdrawToken(uint256 amount, uint256 tokenId)
657         public
658         lock
659         updateReward(msg.sender)
660     {
661         totalSupply -= amount;
662         balanceOf[msg.sender] -= amount;
663         _safeTransfer(stake, msg.sender, amount);
664
665         if (tokenId > 0) {
666             require(tokenId == tokenIds[msg.sender], "tokenId auth");
667             tokenIds[msg.sender] = 0;
668             IVoterV2(voter).detachTokenFromGauge(tokenId, msg.sender);
669         } else {
670             tokenId = tokenIds[msg.sender];
671         }
672
673         IVoterV2(voter).emitWithdraw(tokenId, msg.sender, amount);
674         emit Withdraw(msg.sender, tokenId, amount);
675     }
```

```
555
556     function emitWithdraw(
557         uint256 tokenId,
558         address account,
559         uint256 amount
560     ) external {
561         require(isGauge[msg.sender], "Not Gauge");
562         emit Withdraw(account, msg.sender, tokenId, amount);
563     }
564
```

- **Recommendation:** Remove the external call to voter, or add a separate `emergencyWithdraw(amount)` method that ignores the tokenId and also does not make external calls.

Approve LP token on router

- **Goals**
  - User cannot be tricked into malicious approvals
- This is a web concern, not smart contract. Need to be certain that users are always approving on the correct address.
  - Recommend using a solution similar to yearn's allowlist if possible

- **Withdraw from LP to individual tokens via router**
  - **Goals**
    - Router cannot divert/steal tokens in transit
    - User receives split of LP into base tokens
  - **Contracts**
    - `BaseV2Router01`, in `BaseV2-periphery.sol`
    - `BaseV2Pair`, in `BaseV2-core.sol`
      - `BaseV2-core.sol` diff vs Solidly v1 can be found here
        - For comparison, documents were matched for linting and usage of uint/uint256
  - **Notes**
    - `burn()` in `BaseV2Pair` does the heavy lifting here. Code looks good; diff is near zero from V1. Sends liquidity to LP atomically, then sends directly to `to` passed in `removeLiquidity()`.

ProxyPattern Contracts

- All main contracts are deployed via a proxy system, which consists of:
  - `SolidlyProxy.sol`
  - `SolidlyChildProxy.sol`
  - `SolidlyImplementation.sol`
  - `SolidlyChildImplementation.sol`
  - `SolidlyFactory.sol`
  - `SolidlyDeployer.sol`
- **Issue (high):** `SolidlyImplementation` allows infinite re-initialization by governance of any contract that inherits it.
  - **Commentary Solidly Labs: This could not be performed by any random user but only by the Solidly multisig. It assumes the multisig to act maliciously. The issue was fixed anyhow.**
  - Contracts impacted
    - `SolidlyChildImplementation`
      - `FeeDistV2`
      - `BribeV2`
      - `GaugeV2`
    - `SolidlyFactory`
      - `BaseV2FeeDistFactory`
      - `BaseV2BribeFactory`
    - `ve_distV2`

- `veV2`
- `solidly_library`
- `SolidlyLens`
- `BaseV2Router01`
- `BaseV2Voter`
- `BaseV2`
  - aka `BaseV2-token.sol`
- `BaseV2Minter`
- `BaseV2Fees`
- `BaseV2Pair`

- **Recommendation:** Use the version of `notInitialized()` in `SolidlyProxy.sol`, which has an additional `sstore(INITIALIZED_SLOT, 1)` following the first check.

```
modifier notInitialized() {
    bool initialized;
    assembly {
        initialized := sload(INITIALIZED_SLOT)
        if eq(initialized, 1) {
            revert(0, 0)
        }
        sstore(INITIALIZED_SLOT, 1)
    }
    _;
}
```

- **Minor:** i`mplementationAddress()`, `interfaceAddress()`, and `logicAddress()` in `SolidlyProxy` all have the same `return` text.
  - **Recommendation:** Update `@return` natspec to accurately reflect the given function. In general, the comments around these views and their setters are unclear.
- **Minor:** Inconsistent usage of `0x0` vs `0` in `_delegateCallSubimplmentation()` in `SolidlyProxy`.
  - **Recommendation:** Update to use `0` in place of `0x0`.

```
182     /**
183      * @notice Fallback function that delegatecalls the subimplementation instead of what's in the IMPLEMENTATION_SLOT
184      */
185     function _delegateCallSubimplmentation() internal virtual {
186         assembly {
187             let contractLogic := sload(LOGIC_SLOT)
188             calldatacopy(0x0, 0x0, calldatasize())
189             let success := delegatecall(
190                 gas(),
191                 contractLogic,
192                 0x0,
193                 calldatasize(),
194                 0,
195                 0
196             )
197             let returnDataSize := returndatasize()
198             returndatacopy(0, 0, returnDataSize)
199             switch success
200             case 0 {
201                 revert(0, returnDataSize)
202             }
203             default {
204                 return(0, returnDataSize)
205             }
206         }
207     }
```