Multichain Contracts

Smart Contract Security Assessment

Mar. 28, 2022

ABSTRACT

Dedaub was commissioned to perform an audit of three of the anyswap-v1-core contracts as listed below:

- AnyswapV6Router.sol at commit hash `36d85e128401f362c4d194714006914cf0cf6690`
- AnyswapV6ERC20.sol at commit hash `8c8c988763737b7e2452f162706774ce901b2385`
- AnyswapV4CallProxy.sol at commit hash `c529548fe811fa90b0fe5a2aa623a0c377048f64`

Two auditors worked on the codebase over one week.

Setting & Caveats

The audited codebase is of medium size, ~1.3KLoC, still there is some added complexity due to the very nature of the project which involves cross-chain functionality and reasoning.

The audit's main target is security threats, i.e., what the community understanding would likely call "hacking", rather than regular use of the protocol. Functional correctness (i.e. issues in "regular use") is a secondary consideration. Functional correctness relative to low-level calculations (including units, scaling, quantities returned from external protocols) is generally most effectively done through thorough testing rather than human auditing.

Architecture and High-Level recommendations

The overall correctness and viability of the protocol is highly dependent on the properties of the external Multichain SMPC network. More specifically, regarding the protocol's correctness, external code is supposed to properly handle a number of event logs by correctly interpreting log information (e.g., token id, chain id) and then triggering a suitable transaction that completes the desired operation (e.g., actually mint tokens in the destination chain with a swap-in tx). A particular threat that the off-chain code is

expected to counter successfully is untrusted tokens, possibly impersonating others (either Anyswap ERC tokens, or pairs for swaps). It is expected that tokens are correctly identified and vetted in the Multichain SMPC network.

Regarding the protocol's viability, the fees for cross-chain token bridging are calculated in the Multichain SMPC network as an intermediary step between the two phases of a cross-chain operation (i.e., burn token amount in local chain, mint token amount in destination chain). It is important, especially for chains with high gas price volatility, that fees be adaptable to gas prices, to ensure the system's viability but also eliminate potential DoS attacks.

As is usual in cross-chain related projects, the audited codebase indicates some centralization issues. All of the cross-chain operations rely on an external authority (called `Vault/mpc`) which is the only privileged entity allowed to access any minting-related functions but also, in the case of AnyCall contract, to trigger the execution of a requested transaction. This authority is also privileged to handle crucial parameters of the protocol (e.g. enable/disable swap trade, set/revoke minter contracts, change Vault's address itself). Per the Multichain developers, this concern is completely addressed off-chain, the Vault EOA being secured by the Multichain SMPC network via multiparty threshold ecdsa signatures (tss signatures) during the key generation and signing.
We also refer to some risks related to the Vault's address update mechanisms in issue L1.

VULNERABILITIES & FUNCTIONAL ISSUES

This section details issues that affect the functionality of the contract. Dedaub generally categorizes issues according to the following severities, but may also take other considerations into account such as impact or difficulty in exploitation:

Category	Description
CRITICAL	Can be profitably exploited by any knowledgeable third party attacker to drain a portion of the system's or users' funds OR the contract does not function as intended and severe loss of funds may result.
HIGH	Third party attackers or faulty functionality may block the system or cause the system or users to lose funds. Important system invariants can be violated.
MEDIUM	Examples: -User or system funds can be lost when third party systems misbehave. -DoS, under specific conditions. -Part of the functionality becomes unusable due to programming errors.
LOW	Examples: -Breaking important system invariants, but without apparent consequences. -Buggy functionality for trusted users where a workaround exists. -Security issues which may manifest when the system evolves.

Issue resolution includes "dismissed", by the client, or "resolved", per the auditors.

CRITICAL SEVERITY:

[No critical severity issues]

HIGH SEVERITY:

[No high severity issues]

MEDIUM SEVERITY:

[No medium severity issues]

LOW SEVERITY:

ID	Description	STATUS
L1	The pending updates mechanism is error-prone	**LARGELY RESOLVED**

There is a mechanism to change vaults in AnyswapV6ERC20. This mechanism maintains a pending new vault. The mechanism is quite error-prone, although changing the vault will happen rarely, so this issue is only low-severity. (Currently most AnyswapERC20 instances seem to have an AnyswapV?Router contract as `mpc/vault`, and this does not even support calling any of the "pending" functions.)

Specifically, one problem is that the effective vault is the pending vault, even before the vault change has been accepted by calling `applyVault`. However, if the pending vault does not become the value of the actual `vault` variable, a second change can overwrite it, restoring the old vault. In this way, the owner can lock themselves out.

```
modifier onlyVault() {
      require(msg.sender == mpc(), "AnyswapV3ERC20: FORBIDDEN");
      _;
}

function owner() public view returns (address) {
      return mpc();
}
```

```
function mpc() public view returns (address) {
      if (block.timestamp >= delayVault) {
            return pendingVault;
      }
      return vault;
}

function setVault(address _vault) external onlyVault {
      require(_vault != address(0), "AnyswapV3ERC20: address(0x0)");
      pendingVault = _vault;
      delayVault = block.timestamp + delay;
}

function applyVault() external onlyVault {
      require(block.timestamp >= delayVault);
      vault = pendingVault;
}
```

In the above, although `mpc` or `owner` return the pending vault, a second call to `setVault` before a call to `applyVault` will overwrite the `pendingVault` variable, restoring the old vault.

(Incidentally, the different terminology: `vault`, `mpc`, `owner`, for the same entity, is confusing.)

A second source of error-proneness is the `changeVault` function, which completely bypasses the delay in installing a pending vault:

```
function changeVault(address newVault) external onlyVault returns (bool) {
      require(newVault != address(0), "AnyswapV3ERC20: address(0x0)");
      vault = newVault;
      pendingVault = newVault;
      emit LogChangeVault(vault, pendingVault, block.timestamp);
      return true;
```

```
}
```

Finally, a similar issue arises with respect to `pendingMinter`, which can be overwritten before the minter is actually applied:

```
function setMinter(address _auth) external onlyVault {
      require(_auth != address(0), "AnyswapV3ERC20: address(0x0)");
      pendingMinter = _auth;  // Dedaub: could be overwriting another
      delayMinter = block.timestamp + delay;
}

function applyMinter() external onlyVault {
      require(block.timestamp >= delayMinter);
      isMinter[pendingMinter] = true;
      minters.push(pendingMinter);
}
```

Although none of these issues will necessarily result in problems, the behavior is counter-intuitive.

L2	Caution: Router can never hold wrapped native token	**ACKNOWLEDGED**

(We list this as a “low” issue instead of “advisory” only to bring attention to it.)

Although currently not an issue, a warning for both future maintenance and external (Go) code behavior is that the AnyswapV6Router contract should never directly hold wrapped native tokens. If it does, any attacker can steal these funds via the `withdrawNative` function. This can be done by creating a fake AnyswapERC20 token that merely supports methods `underlying` (returning `wNATIVE`) and `withdrawVault` (doing nothing), without the attacker holding any actual funds with Multichain. The result will be that the router contract will withdraw native tokens and transfer them to the attacker.

```
function withdrawNative(address token, uint amount, address to) external
 returns (uint) {
```

```
        require(AnyswapV1ERC20(token).underlying() == wNATIVE,
                "AnyswapV3Router: underlying is not wNATIVE");
        AnyswapV1ERC20(token).withdrawVault(msg.sender, amount,
                                            address(this));
          // Dedaub: attacker can make the above be a no-op
        IwNATIVE(wNATIVE).withdraw(amount);
        TransferHelper.safeTransferNative(to, amount);
        return amount;
}
```

Generally, it would be a good practice in the code, whenever there is an action that would result in a withdrawal “here” (i.e., would change the balance of the current contract), to check the balance before and after and only act on the difference between the two numbers, not on the original requested amount.

OTHER/ADVISORY ISSUES:

This section details issues that are not thought to directly affect the functionality of the project, but we recommend considering them.

ID	Description	STATUS
A1	Dead code	**RESOLVED**

1. In AnyCallProxy contract the address of authority entity “mpc” can be immediately updated by calling the `changeMPC` function:

```
/// @param _newMPC The address of the new MPC
function changeMPC(address _newMPC) external onlyMPC {
    mpc = _newMPC;
}
```

There also seem to be some remnants of former update functionality although there is currently no way that they can be of any use:

```
struct TransferData {
    uint96 effectiveTime;
    address pendingMPC;
}

TransferData private _transferData;

/// @notice Get the effective time at which pendingMPC may become MPC
function effectiveTime() external view returns(uint256) {
    return _transferData.effectiveTime;
}

/// @notice Get the address of the pending MPC
function pendingMPC() external view returns(address) {
    return _transferData.pendingMPC;
}
```

Similarly, event

```
event TransferMPC(address oldMPC, address newMPC, uint256 effectiveTime);
```

is only thrown during construction.

We suggest removing the dead code and creating another, more suitable event for authority updates.

2. In AnySwapV6ERC20 contract burn functionality checks twice the account address against address(0):

```
function burn(address from, uint256 amount) external onlyAuth returns
(bool) {
```

```
    // Dedaub: redundant requirement, also checked in _burn()
    require(from != address(0), "AnyswapV3ERC20: address(0x0)");
    _burn(from, amount);
    return true;
}

function _burn(address account, uint256 amount) internal {
    require(account != address(0), "ERC20: burn from the zero address");
    // ...
}
```

We suggest removing the redundant requirement from `burn()`.

A2	Gas optimizations through code simplification	**RESOLVED**

In AnyswapV6Router.sol functions `anySwapOutExactTokensForTokensUnderlying()` and `anySwapOutExactTokensForNativeUnderlying()` perform a mint()ing and, right afterwards, a burn()ing of a specified amount:

```
function anySwapOutExactTokensForTokensUnderlying(
    uint amountIn,
    uint amountOutMin,
    address[] calldata path,
    address to,
    uint deadline,
    uint toChainID
) external virtual swapTradeEnabled ensure(deadline) {
    IERC20(AnyswapV1ERC20(path[0]).underlying()).
      safeTransferFrom(msg.sender, path[0], amountIn);

    // Dedaub: the next two lines can be omitted just like in
    //  anyswapOutUnderlying(), since it's just a burn after minting
    AnyswapV1ERC20(path[0]).depositVault(amountIn, msg.sender);
    AnyswapV1ERC20(path[0]).burn(msg.sender, amountIn);
```

```
    emit LogAnySwapTradeTokensForTokens(path, msg.sender, to, amountIn,
amountOutMin, cID(), toChainID);
}
```

Since these two operations are essentially a no-op we suggest removing them for gas savings.

A3	Typographic error	RESOLVED

In AnyCallProxy contract there is a typo in the name of the event thrown in case of a withdrawal:

```
event Withdrawl(address indexed account, uint256 amount);
```

We recommend fixing it for the sake of readability.

A4	Misleading event information	RESOLVED

In AnySwapV6ERC20 contract the function changeVault() emits an event with other than the expected information as described in the following snippet and comments:

```
event LogChangeVault(address indexed oldVault, address indexed newVault,
uint indexed effectiveTime);

function changeVault(address newVault) external onlyVault returns (bool) {
    require(newVault != address(0), "AnyswapV3ERC20: address(0x0)");
    vault = newVault;
    pendingVault = newVault;
    // Dedaub: (newVault, newVault, ts) instead of (oldVault, newVault, ts)
    emit LogChangeVault(vault, pendingVault, block.timestamp);
    return true;
}
```

The potential damage of this error depends on the specifics of the usage of this kind of event logs in external code. In any case, we suggest fixing this issue for accuracy.

A5	Revert messages often refer to outdated versions	RESOLVED

The contract version numbers in revert messages are usually of old versions. It is not clear whether this is done on purpose because of some external functionality that processes the revert messages, but it seems unlikely. E.g., in AnyswapV6ERC20:

```
modifier onlyAuth() {
      require(isMinter[msg.sender], "AnyswapV4ERC20: FORBIDDEN");
      // Dedaub: why V4?
      _;
}

modifier onlyVault() {
      require(msg.sender == mpc(), "AnyswapV3ERC20: FORBIDDEN");
      // Dedaub: why V3?
      _;
}
```

The issue arises in tens of revert messages–virtually every `require` in the contract.

A6	Library SafeERC20 is redundant, performs the same function as TransferHelper	RESOLVED

The two libraries TransferHelper and SafeERC20 in AnyswapV6Router are effectively encoding the same functionality (with the former also supporting native tokens):

```
library TransferHelper {
   function safeApprove(address token, address to, uint value) internal {
       // bytes4(keccak256(bytes('approve(address,uint256)')));
       (bool success, bytes memory data) =
          token.call(abi.encodeWithSelector(0x095ea7b3, to, value));
       require(success && (data.length == 0 || abi.decode(data, (bool))),
               'TransferHelper: APPROVE_FAILED');
   }

   function safeTransfer(address token, address to, uint value) internal {
       // bytes4(keccak256(bytes('transfer(address,uint256)')));
```

```
        (bool success, bytes memory data) =
            token.call(abi.encodeWithSelector(0xa9059cbb, to, value));
        require(success && (data.length == 0 || abi.decode(data, (bool))),
                'TransferHelper: TRANSFER_FAILED');
    }

    function safeTransferFrom(address token, address from, address to,
                              uint value) internal {
        // bytes4(keccak256(bytes('transferFrom(address,address,uint256)')));
        (bool success, bytes memory data) =
            token.call(abi.encodeWithSelector(0x23b872dd, from, to, value));
        require(success && (data.length == 0 || abi.decode(data, (bool))),
                'TransferHelper: TRANSFER_FROM_FAILED');
    }

    function safeTransferNative(address to, uint value) internal {
        (bool success,) = to.call{value:value}(new bytes(0));
        require(success, 'TransferHelper: NATIVE_TRANSFER_FAILED');
    }
}
...
library SafeERC20 {
    using Address for address;

    function safeTransfer(IERC20 token, address to, uint value) internal {
        callOptionalReturn(token,
            abi.encodeWithSelector(token.transfer.selector, to, value));
    }

    function safeTransferFrom(IERC20 token, address from, address to, uint value)
    internal {
        callOptionalReturn(token,
            abi.encodeWithSelector(token.transferFrom.selector, from, to, value));
    }

    function safeApprove(IERC20 token, address spender, uint value) internal {
        require((value == 0) || (token.allowance(address(this), spender) == 0),
               "SafeERC20: approve from non-zero to non-zero allowance"
        );
        callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector,
```

```
                        spender, value));
    }

    function callOptionalReturn(IERC20 token, bytes memory data) private {
        require(address(token).isContract(), "SafeERC20: call to non-contract");

        // solhint-disable-next-line avoid-low-level-calls
        (bool success, bytes memory returndata) = address(token).call(data);
        require(success, "SafeERC20: low-level call failed");

        if (returndata.length > 0) { // Return data is optional
               // solhint-disable-next-line max-line-length
               require(abi.decode(returndata, (bool)),
                       "SafeERC20: ERC20 operation did not succeed");
        }
    }
}
```

A7	Functions could be external instead of public	**RESOLVED**

Functions `changeMPC` and `changeVault` in AnyswapV6Router could be external instead of public, since they are never called internally.

A8	Comments document wrong behavior	**RESOLVED**

Comments could be outdated in minor ways, but especially the comments on `transfer` and `transferFrom` of AnyswapV6ERC20 give the wrong description of behavior and, hence, are actively misleading:

```
/// A transfer to `address(0)` triggers an ETH withdraw matching the
/// sent AnyswapV3ERC20 token in favor of caller.
// Dedaub: wrong comment
...
function transfer(address to, uint256 value) external override
returns (bool) {
```

```
      require(to != address(0) && to != address(this));
...

/// A transfer to `address(0)` triggers an ETH withdraw matching the sent
//  AnyswapV3ERC20 token in favor of caller.
// Dedaub: wrong comment
...
function transferFrom(address from, address to, uint256 value) external
override returns (bool) {
      require(to != address(0) && to != address(this));
...
```

A9	Constants can be declared as such, and magic constants should best be avoided	**RESOLVED**

Storage variable `delay` in AnyswapV6ERC20 is currently a constant, with no means of updating it (nor any need to initialize it to a dynamic value):

```
// configurable delay for timelock functions
uint public delay = 2*24*3600;
```

Furthermore, we recommend the use of symbolic names (e.g., “TWO_DAYS”) instead of magic constants, for clarity and less error-proneness. The above constant (2*24*3600) also appears in AnyswapV6Router and can be given a name there as well.

```
function changeMPC(address newMPC) public onlyMPC returns (bool) {
      require(newMPC != address(0), "AnyswapV3Router: address(0x0)");
      _oldMPC = mpc();
      _newMPC = newMPC;
      _newMPCEffectiveTime = block.timestamp + 2*24*3600;
       // Dedaub: magic constant
      emit LogChangeMPC(_oldMPC, _newMPC, _newMPCEffectiveTime, cID());
      return true;
}
```

A10	Floating pragma	DISMISSED
The floating pragma ^0.8.2 and ^0.8.6 are used in the contracts, allowing them to be compiled, respectively, with versions 0.8.2 - 0.8.12 and 0.8.6 - 0.8.12 of the Solidity compiler. Although the differences between these versions are small, floating pragmas should be avoided and the pragma should be fixed to the version that will be used for the contracts' deployment.		
A11	Compiler known issues	INFO
Solidity compiler versions v0.8.2, v.0.8.6 have, at the time of writing, some known bugs. We inspected the code and found that it is not affected by these bugs.		

DISCLAIMER

The audited contracts have been analyzed using automated techniques and extensive human inspection in accordance with state-of-the-art practices as of the date of this report. The audit makes no statements or warranties on the security of the code. On its own, it cannot be considered a sufficient assessment of the correctness of the contract. While we have conducted an analysis to the best of our ability, it is our recommendation for high-value contracts to commission several independent audits, as well as a public bug bounty program.

ABOUT DEDAUB

Dedaub offers technology and auditing services for smart contract security. The founders, Neville Grech and Yannis Smaragdakis, are top researchers in program analysis. Dedaub's smart contract technology is demonstrated in the contract-library.com service, which decompiles and performs security analyses on the full Ethereum blockchain.