



Code Security Audit Report

For

EDBank

31th Oct 2024

Table of Contents

Summary

Overview

Audit Summary

Result Summary

Audit Result

EDK-01(Low): Leaving residual contract code after removal may pose unknown risks

EDK-02(Info): Excessive authorization can lead to redundant code logic.

EDK-03(Info): Insecure authentication method.

About

Summary

This report has been prepared for **EDBank** to discover issues and vulnerabilities in the source code of the **EDBank** project as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Static Analysis and Manual Review techniques.

The auditing process pays special attention to the following consideration:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

Overview

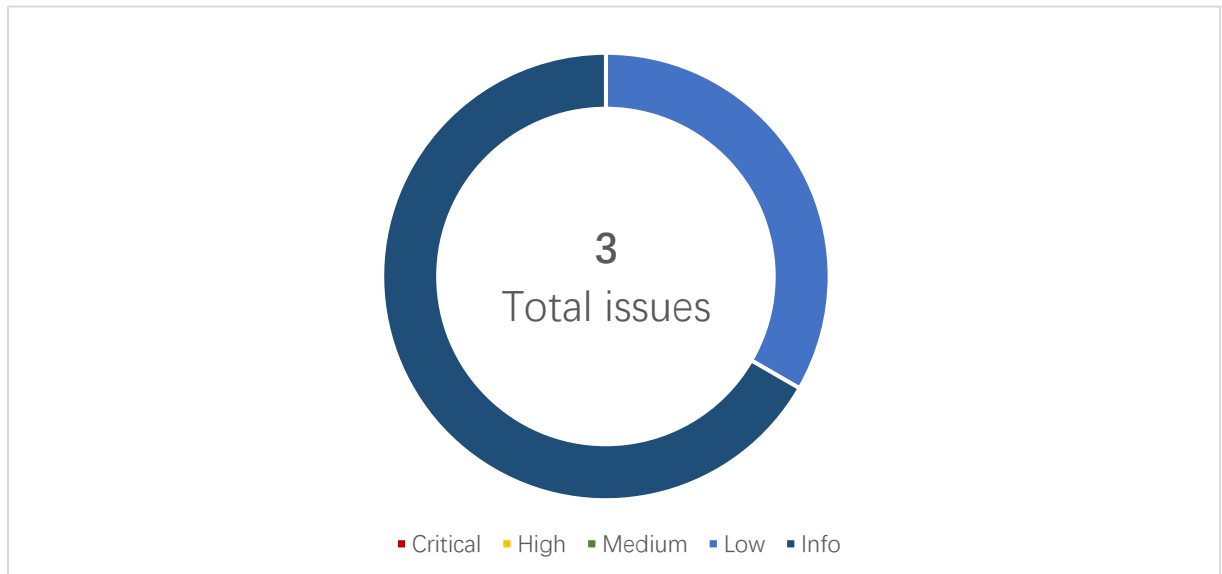
Audit Scope

Contract Name	EDBankDao/src
Platform	Etherum
Language	Solidity
Code Base	https://github.com/edbankDao/SmartContract/src
Commit	a9a2a4a75f1bffc47448cc31627d121222acd4bf

Result Summary

Vulnerability Level	Total	Pending	Solved	Acknowledged
Critical	0	0	0	0
High	0	0	0	0
Medium	0	0	0	0
Low	1	0	1	0
Info	2	0	0	2

Audit Result



EDK-01(Low): Leaving residual contract code after removal may pose unknown risks

Category	Severity	Location	Status
Logic Issue	Low	end.sol:373 end.sol:395 end.sol:443 end.sol:442	Solved

Description

The project code was entirely forked from the MakerDao project, which includes the Cat.sol contract. While EDBank did not directly import Cat.sol, it indirectly referenced interfaces from Cat in its code logic. This issue could potentially lead to some unpredictable problems.

Vulnerability Analysis

1. In end.sol, the Catlike and Fliplike interface is used and an attempt is made to assign a value to it.

```
// --- Administration ---
function file(bytes32 what, address data) external auth {
    require(live == 1, "End/not-live");
    if (what == "vat") vat = VatLike(data);
    else if (what == "cat") cat = CatLike(data);
    else if (what == "dog") dog = DogLike(data);
    else if (what == "vow") vow = VowLike(data);
    else if (what == "pot") pot = PotLike(data);
    else if (what == "spot") spot = SpotLike(data);
    else if (what == "cure") cure = CureLike(data);
    else revert("End/file-unrecognized-param");
    emit File(what, data);
}
```

```
function skip(bytes32 ilk, uint256 id) external {
    require(tag[ilk] != 0, "End/tag-ilk-not-defined");

    (address _flip, , ) = cat.ilks(ilk);
    FlipLike flip = FlipLike(_flip);
    (, uint256 rate, , ) = vat.ilks(ilk);
    (uint256 bid, uint256 lot, , , address usr, , uint256 tab) = flip
        .bids(id);

    vat.suck(address(vow), address(vow), tab);
    vat.suck(address(vow), address(this), bid);
    vat.hope(address(flip));
    flip.yank(id);

    uint256 art = tab / rate;
    Art[ilk] = add(Art[ilk], art);
    require(int256(lot) >= 0 && int256(art) >= 0, "End/overflow");
    vat.grab(
        ilk,
        usr,
        address(this),
        address(vow),
        int256(lot),
        int256(art)
    );
}
```

Recommendation

1. Clean up the interfaces generated from files that have not been imported.As for the dependency solution.

EDK-02(Info): Excessive authorization can lead to redundant code logic.

Category	Severity	Location	Status
Code Issue	Info	esd.sol:94 esd.sol:113	Acknowledged

Description

Excessive token authorization in the permit function leads to invalid initial allowance checks. The code is redundant and does not allow for reducing authorization for authorized accounts.

Vulnerability Analysis

1. The logic check "`allowance[src][msg.sender] != type(uint).max`" is present in both functions **Transferfrom** and **Burn**.

```
}  
function transferFrom(  
    address src,  
    address dst,  
    uint wad  
) public returns (bool) {  
    require(balanceOf[src] >= wad, "Esd/insufficient-balance");  
    if (src != msg.sender && allowance[src][msg.sender] != type(uint).max) {  
        require(  
            allowance[src][msg.sender] >= wad,  
            "Esd/insufficient-allowance"  
        );  
        allowance[src][msg.sender] = sub(allowance[src][msg.sender], wad);  
    }  
    balanceOf[src] = sub(balanceOf[src], wad);  
    balanceOf[dst] = add(balanceOf[dst], wad);  
    emit Transfer(src, dst, wad);  
    return true;  
}  
  
function mint(address usr, uint wad) external auth {  
    balanceOf[usr] = add(balanceOf[usr], wad);  
    totalSupply = add(totalSupply, wad);  
    emit Transfer(address(0), usr, wad);  
}  
  
function burn(address usr, uint wad) external {  
    require(balanceOf[usr] >= wad, "Esd/insufficient-balance");  
    if (usr != msg.sender && allowance[usr][msg.sender] != type(uint).max) {  
        require(  
            allowance[usr][msg.sender] >= wad,  
            "Esd/insufficient-allowance"  
        );  
        allowance[usr][msg.sender] = sub(allowance[usr][msg.sender], wad);  
    }  
    balanceOf[usr] = sub(balanceOf[usr], wad);  
    totalSupply = sub(totalSupply, wad);  
    emit Transfer(usr, address(0), wad);  
}
```

2. However, in the permit function, if authorization is allowed, the allowance will be set to `type(uint).max`. This will result in the condition `allowance[usr][msg.sender] != type(uint).max` always being invalid.

```
require(holder != address(0), "Esd/invalid-address-0");  
require(holder == ecrecover(digest, v, r, s), "Esd/invalid-permit");  
require(expiry == 0 || block.timestamp <= expiry, "Esd/permit-expired");  
require(nonce == nonces[holder]++, "Esd/invalid-nonce");  
uint wad = allowed ? type(uint).max : 0;  
allowance[holder][spender] = wad;  
emit Approval(holder, spender, wad);
```


Recommendation

Modify the condition for evaluation, or change '!=' to '<='.

EDK-03(Info): Insecure authentication method.

Category	Severity	Location	Status
Code Issue	Info	esd. sol:30	Acknowledged

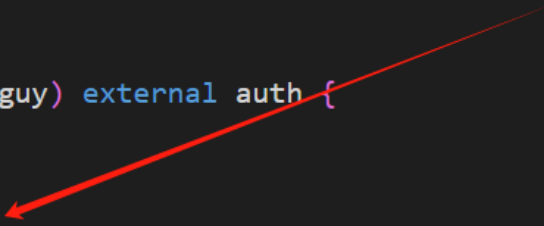
Description

Granting all high-level users the same operational permissions using a self-written security authentication method may lead to the issue of permission sprawl.

Vulnerability Analysis

1. Storing high-level wallet addresses in a mapping and granting all authorized users the same permissions may lead to the issue of permission abuse. In the future, if a private key is lost, an attacker holding the private key may remove the original high-level addresses and grant high-level permissions to their other accounts.

```
mapping(address => uint) public wards;
function rely(address guy) external auth {
    wards[guy] = 1;
}
function deny(address guy) external auth {
    wards[guy] = 0;
}
modifier auth() {
    require(wards[msg.sender] == 1, "Esd/not-authorized");
    _;
}
```



Recommendation

suggest upgrading to use OpenZeppelin's access control library and assigning different permission levels for different functions.

Link: [OpenZeppelin](https://openzeppelin.org/)

About

Damocles is a 2023 web3 security company specializing in online security services, including smart contract audit, Product audit, penetration testing, GameFi security audit and cheat detection.

Main Web: <https://damocleslabs.com/>

Twitter: <https://twitter.com/DamoclesLabs>

Email: support@damocleslabs.com