Oxumarkhatab X TechFund

SECURITY REVIEW REPORT FOR ZKEVM - ETH BRIDGE

CONTENTS

ABOUT 0xUMARKHATAB

0xumarkhatab is a security researcher aiming to secure DeFi world

By preventing hacks from the bad guys. His 3 years of block-chain development, understanding low-level block-chain architecture and mastery on Ethereum ecosystem has given him an additional advantage to become a better security researcher.

Currently, Umar has audited 5 protocols finding multiple highs and mediums. He is currently perusing immunifi bug bounties to secure newly launching web3 protocols.

This is his time-boxed engagement with the zkevm-ethereum's set of contracts to uncover potential vulnerabilities to demonstrate his skill to join hands with Techfund to make web3 a safer place than we found

AUDIT LED BY

UMAR KHATAB

Security Researcher

Audit Starting Date	Audit Completion Date
04.03.2024	06.03.2024

DECIDING SEVERITY

SEVERITY STRUCTURE

The vulnerability severity is calculated based on two components

IMPACT \ LIKELIHOOD	Rare	Unlikely	Likely	Very Likely
Low / Info	Low / Info	Low / Info	Medium	Medium
Medium	Low / Info	Medium	Medium	High
High	Medium	Medium	High	Critical
Critical	Medium	High	Critical	Critical

SEVERITY CHARACTERISTICS

Vulnerabilities can range in severity and impact, and it's important to understand their level of severity in order to prioritize their resolution. Here are the different types of severity levels of vulnerabilities:

CRITICAL

Vulnerabilities with this level of severity can result in significant financial losses or reputational damage. They often allow an attacker to gain complete control of a contract, directly steal or freeze funds from the contract or users, or permanently block the functionality of a protocol. Examples include infinite mints and governance manipulation.

HIGH

Vulnerabilities with this level of severity can result in some financial losses or reputational damage. They often allow an attacker to directly steal yield from the contract or users, or temporarily freeze funds. Examples include inadequate access control integer overflow/underflow, or logic bugs.

MEDIUM

Vulnerabilities with this level of severity can result in some damage to the protocol or users, without profit for the attacker. They often allow an attacker to exploit a contract to cause harm, but the impact may be limited, such as temporarily blocking the functionality of the protocol. Examples include uninitialized storage pointers and failure to check external calls.

LOW

Vulnerabilities with this level of severity may not result in financial losses or significant harm. They may, however, impact the usability or reliability of a contract. Examples include slippage and front-running, or minor logic bugs.

It's important to consider all types of vulnerabilities, including informational ones, when assessing the security of the project. A comprehensive security audit should consider all types of vulnerabilities to ensure the highest level of security and reliability.

EXECUTIVE SUMMARY

OVERVIEW

Polygon zkEVM is a novel technology and is the first EVM-equivalent zero-knowledge scaling solution, where existing smart contracts, developer tooling and wallets can work seamlessly. Polygon zkEVM harnesses zero knowledge proofs in order to reduce transaction costs and increase throughput, while inheriting the security of Ethereum.

The transactions in zkEVM network (L2) are being compiled into batches, these batches are then sequenced in an Ethereum smart contract and after that their state transitions are being proved and verified on the Ethereum, achieving a trusted state.

During the engagement , 0xumarkhatab has uncovered multiple critical , high and medium severity vulnerabilities that proves his skill set in auditing Web3 protocols.

SCOPE

The analyzed resources are located on:

https://etherscan.io/address/0xbf7e1FA05e9c183aDD217fE56521bb7Eb2489e41
https://etherscan.io/address/0x5ac4182A1dd41AeEf465E40B82fd326BF66AB82C
https://etherscan.io/address/0xbc1ea504fc54d078514efcca1f6860b5219b6bc3
https://etherscan.io/address/0x4F9A0e7FD2Bf6067db6994CF12E4495Df938E6e9
https://etherscan.io/address/0xCB19eDdE626906eB1EE52357a27F62dd519608C2
https://zkevm.polygonscan.com/address/0xCB19eDdE626906eB1EE52357a27F62dd519608C2
https://zkevm.polygonscan.com/address/0x5ac4182A1dd41AeEf465E40B82fd326BF66AB82C
https://zkevm.polygonscan.com/address/0x0200143Fa295EE4dffEF22eE2616c2E008D81688

FINDINGS SUMMARY

SEVERITY	INSTANCES
CRITICAL	3
HIGH	**1**
MEDIUM	5

TOTAL : 9

1. ERC777 RE-ENTRANCY ATTACK

SEVERITY: CRITICAL

Impact : Infinite assets bridge by paying only a fraction to the protocol

Remediation

implement re-entrancy guard for the bridge/claim functions in the Polygon zkEVM Bridge contract. My suggestion is to use openZeppelin Re-Entrancy Guard

Description

In the function bridgeAsset() any user can bridge his ERC20 tokens specifying the destination network and the destination address, in case it is not a wrapped token it will call transferFrom (SafeERC20.safeTransferFrom) and update the balance of the contract and eventually create a deposit leaf in the Merkle tree. The bridge gives natural opportunity to bridge any kind of ERC20 tokens, also including extended version of ERC20, such as ERC777.

The ERC777 tokens (some examples are Verasity, imBTC, AMP, p.Network tokens and etc, for example Polygon Bridge has these type of tokens even in their controlled list) are type of ERC20 tokens that extend its functionality with a couple of features such as call hooks.

Given this and the fact of how balance calculation is being done in the contract there is a possibility for anyone to drain all of these tokens out of bridge via a subtle reentrancy issue.

Attack breakdown:

The vulnerability arises since the balance update is done the following way:

```
             // In order to support fee tokens check the amount received, not the transferred
                uint256 balanceBefore = IERC20Upgradeable(token).balanceOf(
                    address(this)
                );
                IERC20Upgradeable(token).safeTransferFrom(
                    msg.sender,
                    address(this),
                    amount
                );
                uint256 balanceAfter = IERC20Upgradeable(token).balanceOf(
                    address(this)
                );

                // Override leafAmount with the received amount
                leafAmount = balanceAfter - balanceBefore;
```

So the leafAmount is calculating the balance difference before and after the call, to be compliant with fee-on-transfer tokens. Although at the first sight it may seem impossible to re-enter in the transferFrom call, since the receiver is the bridge contract and is not controlled by an attacker, there is a lesser known call hook called "ERC777TokensSender" which will be called for the "from" address before the balance has been actually transferred to the destination. In order to register the hook, attacker needs to call setInterfaceImplementer on _ERC1820_REGISTRY (which is a known address in the Ethereum) and register itself as its own

"ERC777TokensSender" interface implementer.

At this moment attacking contract is able to receive tokensToSend() callback call, whenever the transferFrom (safeTransferFrom) is being called with "from" set to attackers address.

PoC

In order to utilize the reentrancy and generate unfair deposits the attacker needs to re-enter the bridgeAsset() function in the following way: all of the re-entered calls except the last one are called with amount = 0 (the attack is also fully possibly with amount=1 as well, so the zero check would not mitigate it), and only in the last re-entered call it should send some amount of tokens that he wants to amplify; for the sake of simplicity we will take 10**18 (one token).

Such designed reentrancy will work as the

```
uint256 balanceBefore =IERC20Upgradeable(token).balanceOf(address(this));
```

will already get set and will not change even if we re-enter in the bridgeAsset call. And the deposit will be amplified by the level of reentrancy.

Re-entrancy breakdown:

- bridgeAsset(amount = 0):
- LEVEL = 1, balanceBefore=0:
    - token.safeTransferFrom() -> bridgeAsset(amount = 0):
    - LEVEL = 2, balanceBefore=0
        - token.safeTransferFrom()-> bridgeAsset(amount = 0):
        - LEVEL = 3, balanceBefore=0
            - token.safeTransferFrom() -> bridgeAsset(amount = 10e18):

- LEVEL = 4, balanceBefore = 0, balanceAfter = leafAmount = 1018
- LEVEL = 3, balanceAfter = leafAmount = 10e18
- LEVEL = 2, balanceAfter = leafAmount = 10e18
- LEVEL = 1, balanceAfter = leafAmount = 10e18

It can be seen that all of the reentrancy levels will eventually calculate leafAmount = 10*18 and make a correct deposit and emit the corresponding event (with corresponding depositCount). Thus the amount deposited in this case will be 3 * (10**18), this can be done for any number of token amount and any number of reentrancy level, in general having deposits for LEVEL * amount Also the attack only needs one transactions to be executed, and it can be done prior to active bridge usage, as the attacker can create the deposit leaves and wait for the right moment to claim them.

2. PolygonZKEVM Bridge allows bridging to Chain ID 0 causing users loss of funds

SEVERITY: CRITICAL

Impact : Loss of funds for users .if huge amount is mistakenly sent to non-existing chain like 1 Million ETH , this loss will escalate.

Remediation

Implement non-zero check for destinationNetwork in revert case

```
destinationNetwork == 0 &&

destinationNetwork >= _CURRENT_SUPPORTED_NETWORKS
```

Description

The function bridgeAsset() and bridgeMessage() do check that the destination network is different than the current network. However, the validity check of destination Network chain id is incorrect. Current implementation checks if networkId is >=2 , which means it intends to allow chain Id 1 (Ethereum mainnet), which is the correct check but unfortunately , it also allows chain Id 0 to be passed as a valid chain id (this is related to the other "DoS in Bridging native MATIC from Polygon zkEVM mainnet to other chains" issue in PolygonZKEVM bridge deployed on ethereum mainnet).

If accidentally the wrong networkId is given as a parameter (), then the function is sent to a nonexisting network. If the network would be deployed in the future the funds would be recovered. However, in the meantime they are inaccessible and thus lost for the sender and recipient. Note: other bridges usually have validity checks on the destination.

```
  function bridgeAsset(
    uint32 destinationNetwork,
    address destinationAddress,
    uint256 amount,
    address token,
    bool forceUpdateGlobalExitRoot,
    bytes calldata permitData
) public payable virtual ifNotEmergencyState nonReentrant {
    if (
        destinationNetwork == networkID ||
        destinationNetwork >= _CURRENT_SUPPORTED_NETWORKS
    ) {
        revert DestinationNetworkInvalid();
    }
// remaining code
..
..

}
function bridgeMessage(
    uint32 destinationNetwork,
    address destinationAddress,
    bool forceUpdateGlobalExitRoot,
    bytes calldata metadata
) external payable ifNotEmergencyState {
  if (
        destinationNetwork == networkID ||
        destinationNetwork >= _CURRENT_SUPPORTED_NETWORKS
    ) {
        revert DestinationNetworkInvalid();
    }
    //remaining code
}
```

3. DoS in Bridging native MATIC from Polygon zkEVM mainnet to other chains

SEVERITY: CRITICAL

Impact : Loss of Funds for users

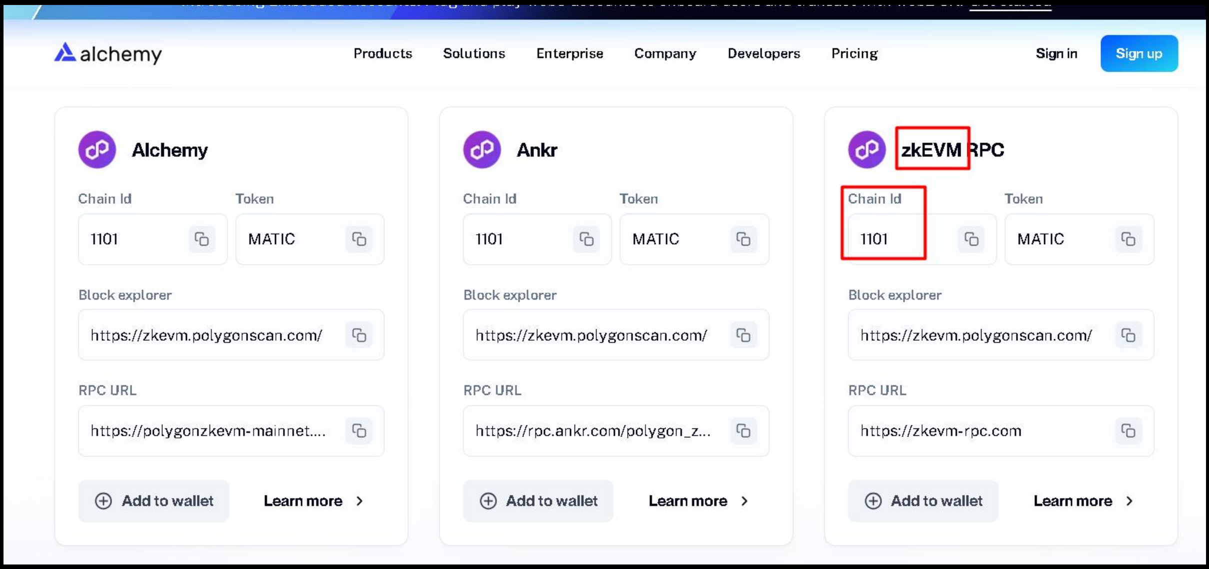
Remediation

Insert Correct Mainnet Network ID as 1101

```
contract PolygonZkEVMBridge is
 DepositContract,
 EmergencyManager,
 IPolygonZkEVMBridge
{
 // other declarations

 // Mainnet identifier
 uint32 private constant _MAINNET_NETWORK_ID = 1101;
 // remaining code
}
```

Description

The function bridgeAsset() function bridges assets from one chain to other. Those assets can either be tokens or native ether from mainnet (chain id of zkEVM Polygon mainnet is 1101 remember). Both cases of token and ether are intended and tried to be safely implemented. However theirs a fatal mistake in the code which will cause a huge loss of funds to users if they want to bridge native assets to some destination chain.



Alchemy	Ankr	zkEVM RPC
Chain Id: 1101	Chain Id: 1101	Chain Id: 1101
Token: MATIC	Token: MATIC	Token: MATIC
Block explorer: https://zkevm.polygonscan.com/	Block explorer: https://zkevm.polygonscan.com/	Block explorer: https://zkevm.polygonscan.com/
RPC URL: https://polygonzkevm-mainnet....	RPC URL: https://rpc.ankr.com/polygon_z...	RPC URL: https://zkevm-rpc.com
Add to wallet / Learn more	Add to wallet / Learn more	Add to wallet / Learn more

```
contract PolygonZkEVMBridge is

    DepositContract,

    EmergencyManager,

    IPolygonZkEVMBridge

{

    // other declarations


    // Mainnet identifier

    uint32 private constant _MAINNET_NETWORK_ID = 0;

    // remaining code

}
```

The problem starts from the constant _MAINNET_NETWORK_ID which is incorrectly configured to be 0. Interestingly the this variable is kept constant so no part of the code can change it after it's deployed (becomes part of the byte code).

This _MAINNET_NETWORK_ID is used in bridgeAsset function as a chainId of mainnet when users are trying to send MATIC from ZkEVM Polygon Mainnet to some other chain.

```
function bridgeAsset(
     uint32 destinationNetwork,
     address destinationAddress,
     uint256 amount,
     address token,
     bool forceUpdateGlobalExitRoot,
     bytes calldata permitData
 ) public payable virtual ifNotEmergencyState nonReentrant {
     if (
         destinationNetwork == networkID ||
         destinationNetwork >= _CURRENT_SUPPORTED_NETWORKS
     ) {
         revert DestinationNetworkInvalid();
     }
     address originTokenAddress;
     uint32 originNetwork;
     bytes memory metadata;
     uint256 leafAmount = amount;
     if (token == address(0)) {
         // Ether transfer
         if (msg.value != amount) {
             revert AmountDoesNotMatchMsgValue();
         }
// Ether is treated as ether from mainnet
->             originNetwork = _MAINNET_NETWORK_ID;
        } else {
        // remaining code ( originNetwork is changed here correctly )
        }
// remaining code ( originNetwork is not changed here )
   }
```

When sending assets from Mainnet , the user does not provide token address and it will essentially be zero

```
      if (token == address(0)) {
           // Ether transfer
           if (msg.value != amount) {
               revert AmountDoesNotMatchMsgValue();
           }

           // Ether is treated as ether from mainnet
->             originNetwork = _MAINNET_NETWORK_ID;
        }
```

But unfortunately , the `_MAINNET_NETWORK_ID` that is passed as originNetwork is invaid. It's unclear from just the contracts if _MAINNET_NETWORK_ID=0 is processed as mainnet network but if it is not , the sequencers or other tools of the protocol using correct mainnet chainId will potentially discard this transaction as invalid.

This will cause users to lose their funds if they are bridging ETH worth Billions of dollars

4. DoS Cross chain messaging : Attacker can spam users for bridgeMessage

SEVERITY: HIGH

Impact : Denial of service for users when there are billions of spam messages to filter one or two genuine messages.

Remediation

Denial of service for users when there are billions of spam messages to filter one or two genuine messages.

```
// declare this before constructor

 mapping(uint=>mapping(address=>uint)) public perBlockMessagesSent;

uint public maxMessagesPerBlock=1000;


   // insert this in starting of bridgeMessage

   require(

   perBlockMessagesSent[block.number][msg.sender]<=maxMessagesPerBlock,

   "Your Message Limit for this block is exceeded"

   )
```

Description

The bridgeMessage function allows anyone from source chain to send cross chain message to anyone on other chain.

```
function bridgeMessage(

     uint32 destinationNetwork,

     address destinationAddress,

     bool forceUpdateGlobalExitRoot,

     bytes calldata metadata

 ) external payable ifNotEmergencyState {
    if (
        destinationNetwork == networkID ||
        destinationNetwork >= _CURRENT_SUPPORTED_NETWORKS
     ) {
        revert DestinationNetworkInvalid();
     }
     //remaining code
 }
```

All it checks for is if chain ids are correct.

A Malicious Attacker can spam other users by constantly watching over mempool for potential interactions with the bridge. Then take their addresses and send them millions of spam messages because currently there is no restriction or time-lock for how many messages a user can send to others.

As a result , the legit users will see millions of spam messages in their other chain's inbox losing sight of genuine meaningful messages sent by legit users and they will rather abandon the use of bridging messages than ti find one or two from a list of billions of spam messages.

Specifically , this scenario can be leveraged to attack a group of people making them suffer denial of service.

5. Nonce Behavior Discrepancy Between zkSync Era (ZKEVM bridge)and EIP-161

SEVERITY: MEDIUM

Impact : Non-interoperability with existing tools that adhere to EIP-161 and will cause DoS for users.

Remediation

It is advisable to use pre-increment instead.

```
    bytes32 hashStruct = keccak256(
            abi.encode(
                PERMIT_TYPEHASH,
                owner,
                spender,
                value,
->>>            ++nonces[owner],
                deadline
            )
        );
```

Description

The discrepancy in deployment nonce behavior between zkSync Era and EVM can cause problems for contract factories and developers. zkSync Era starts the deployment nonce at zero, unlike the EVM, where it starts at one. This difference may lead to incompatibility with the tools that most users will use because they are ethereum EIP-161 compliant and as zkEVM promises to be compatible with EVM , they would not know what about the real issue causing confusion and frustration in people leaving them give up usage of system.

Proof of Concept

As per EIP-161, it's specified that account creation transactions and the CREATE operation should increase the nonce beyond its initial value by one.

https://github.com/ethereum/EIPs/blob/master/EIPS/eip-161.md#specification

" *Account creation transactions and the CREATE operation SHALL, prior to the execution of the initialisation code, increment the nonce over and above its normal starting value by one (for normal networks, this will be simply 1, however test-nets with non-zero default starting nonces will be different).* "

In other words, when an EOA signs their first permit using tools which are EIP161 compliant , the starting nonce will be 1 adhering to EIP-161 , however , when using zkEVM bridge , nonce that will be checked will be zero .

nonce(EOA): 1 -> 2

nonce(zkEVMBridge): 0 -> 1

See the post increment in building the digest for checking validity of signature.

```
    function permit(

      address owner,

      address spender,

      uint256 value,

      uint256 deadline,

      uint8 v,

      bytes32 r,

      bytes32 s

  ) external {
     require(
          block.timestamp <= deadline,
          "TokenWrapped::permit: Expired permit"
       );
       bytes32 hashStruct = keccak256(
          abi.encode(
              PERMIT_TYPEHASH,
              owner,
              spender,
              value,
->>>          nonces[owner]++,
              deadline
          )
       );
   }
```

when it's user's first transaction , nonces[owner] will be 0 . Due to post increment , the value 0 will be used in the expression , and later incremented unlike pre-increment.

This will cause the digest to be built using nonce=0 , and due to Avalanche Effect in Hash Function keccak256, a small change in the value to be hashed will result in a huge change in generation of Hash.

So , a different hash value containing the v ,r and s values will be generated. ecrecover return address on the generated digest will not match the owner address and permit will fail

even if the signature values (v,r,s) were correct.

6. Tfi Example Rug Pull vectors, Incompatibility with EIP1967 standard and lack of access control

SEVERITY: MEDIUM

Impact : Malicious or Compromised owner can rug users or brick protocol.

Remediation

Use Proxy-Admin from openzeppelin and make it the owner of Tfi proxy so that chances of rug pull are nearly zero.

Additionally use multisig wallet for owner operations of high order like 5-of-7

Description

Implementing a valid access control policy is an essential step in maintaining the security of a smart-contract. All the features of the smart contract , such as add/remove roles and upgrade contracts are given by Access Control. For instance, Ownership is the most common form of Access Control. In other words, the owner of a contract (the account that deployed it by default) can do some administrative tasks on it. Additional authorization levels are needed to implement the least privilege principle, also known as least-authority, which ensures only authorized processes, users, or programs can access the necessary resources or information. The ownership role is useful in a simple system, but more complex projects require the use of more roles by using Role-based access control.

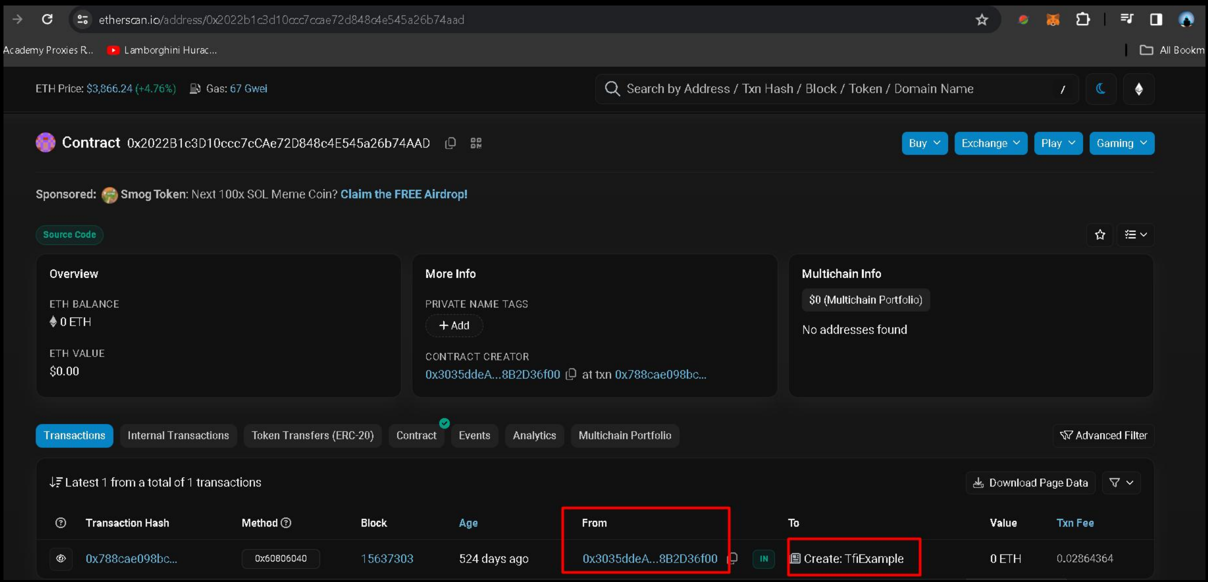
There could be multiple roles such as manager, admin in contracts which use a proxy contract.In TfiExample.sol proxy contract, owner is the only one privileged role. Owner can transfer the contract ownership, call the following functions. In conclusion, owner role can do too many actions in TfiExample smart contract.

If the private key of the owner account is compromised and multi-signature is not implemented, the attacker can perform many actions such as transferring ownership or changing oracle , fee , and other critical parameters without following the principle of least privilege.

```
function changeOracle(address _oracle) public onlyOwner {

        oracleId = _oracle;

    }

    function changeJobId(string memory _jobId) public onlyOwner {

        jobId = _jobId;

    }

    function changeFee(uint256 _fee) public onlyOwner {

        fee = _fee;

    }

    function changeToken(address _address) public onlyOwner {

        setChainlinkToken(_address);

    }
```

That’s why it’s recommended to use proxy-admin contract given by openzeppelin to minimze the attack surface by making proxy-admin contract as the sole owner of the proxy and logic contract.

Currently , we can see from the deployment transaction of the



ETH Price: $3,866.24 (+4.76%) Gas: 67 Gwei

Contract 0x2022B1c3D10ccc7cCAe72D848c4E545a26b74AAD

Sponsored: Smog Token: Next 100x SOL Meme Coin? Claim the FREE Airdrop!

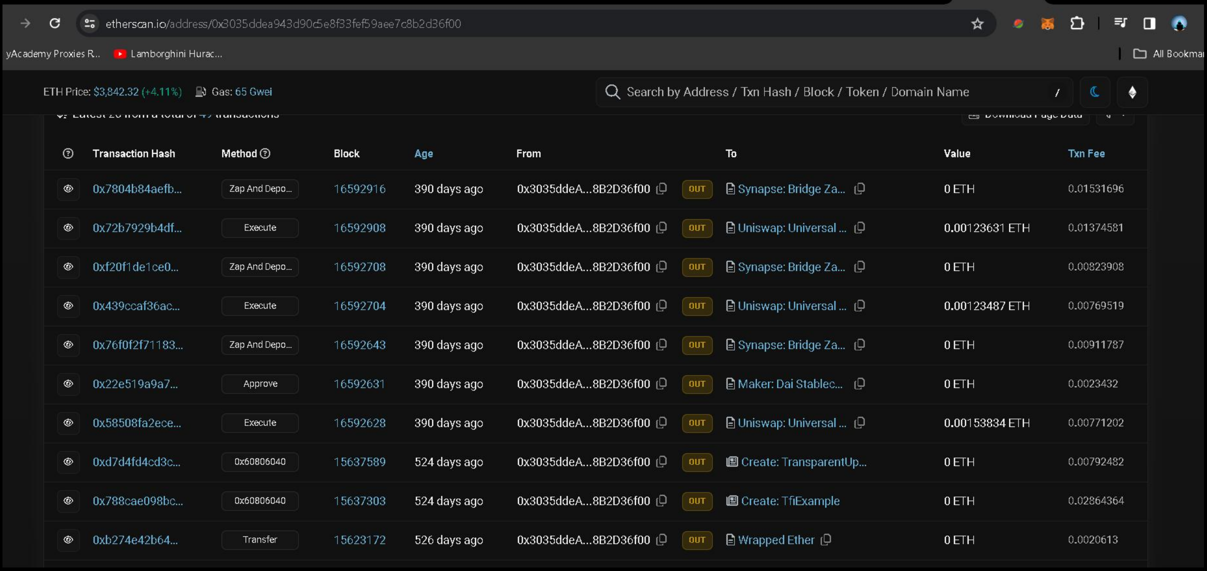
Overview
ETH BALANCE
0 ETH
ETH VALUE
$0.00

More Info
PRIVATE NAME TAGS
+ Add
CONTRACT CREATOR
0x3035ddeA...8B2D36f00 at txn 0x788cae098bc...

Multichain Info
$0 (Multichain Portfolio)
No addresses found

Latest 1 from a total of 1 transactions

Transaction Hash	Method	Block	Age	From		To	Value	Txn Fee
0x788cae098bc...	0x60806040	15637303	524 days ago	0x3035ddeA...8B2D36f00	IN	Create: TfiExample	0 ETH	0.02864364


etherscan.io/address/0x3035ddea943d90c5e8f33fef59aee7c8b2d36f00

ETH Price: $3,842.32 (+4.11%) Gas: 65 Gwei

Transaction Hash	Method	Block	Age	From		To	Value	Txn Fee
0x7804b84aefb...	Zap And Depo...	16592916	390 days ago	0x3035ddeA...8B2D36f00	OUT	Synapse: Bridge Za...	0 ETH	0.01531696
0x72b7929b4df...	Execute	16592908	390 days ago	0x3035ddeA...8B2D36f00	OUT	Uniswap: Universal ...	0.00123631 ETH	0.01374581
0xf20f1de1ce0...	Zap And Depo...	16592708	390 days ago	0x3035ddeA...8B2D36f00	OUT	Synapse: Bridge Za...	0 ETH	0.00823908
0x439ccaf36ac...	Execute	16592704	390 days ago	0x3035ddeA...8B2D36f00	OUT	Uniswap: Universal ...	0.00123487 ETH	0.00769519
0x76f0f2f71183...	Zap And Depo...	16592643	390 days ago	0x3035ddeA...8B2D36f00	OUT	Synapse: Bridge Za...	0 ETH	0.00911787
0x22e519a9a7...	Approve	16592631	390 days ago	0x3035ddeA...8B2D36f00	OUT	Maker: Dai Stablec...	0 ETH	0.0023432
0x58508fa2ece...	Execute	16592628	390 days ago	0x3035ddeA...8B2D36f00	OUT	Uniswap: Universal ...	0.00153834 ETH	0.00771202
0xd7d4fd4cd3c...	0x60806040	15637589	524 days ago	0x3035ddeA...8B2D36f00	OUT	Create: TransparentUp...	0 ETH	0.00792482
0x788cae098bc...	0x60806040	15637303	524 days ago	0x3035ddeA...8B2D36f00	OUT	Create: TfiExample	0 ETH	0.02864364
0xb274e42b64...	Transfer	15623172	526 days ago	0x3035ddeA...8B2D36f00	OUT	Wrapped Ether	0 ETH	0.0020613

TfiExample proxy ,

https://etherscan.io/address/0x3035ddeA943D90c5e8F33fef59aee7c8B2D36f00

Someone has deployed this contract almost 1.5 years ago.

When we look at the address , we see following :

The Explorer shows it is not a smart contract (proxy-admin),

Rather just an EOA that is creating a bunch of transactions.

Being not a proxy-admin contract as an admin of proxy contract is already in-compatible with EIP1967 standard suggested by openzeppelin.

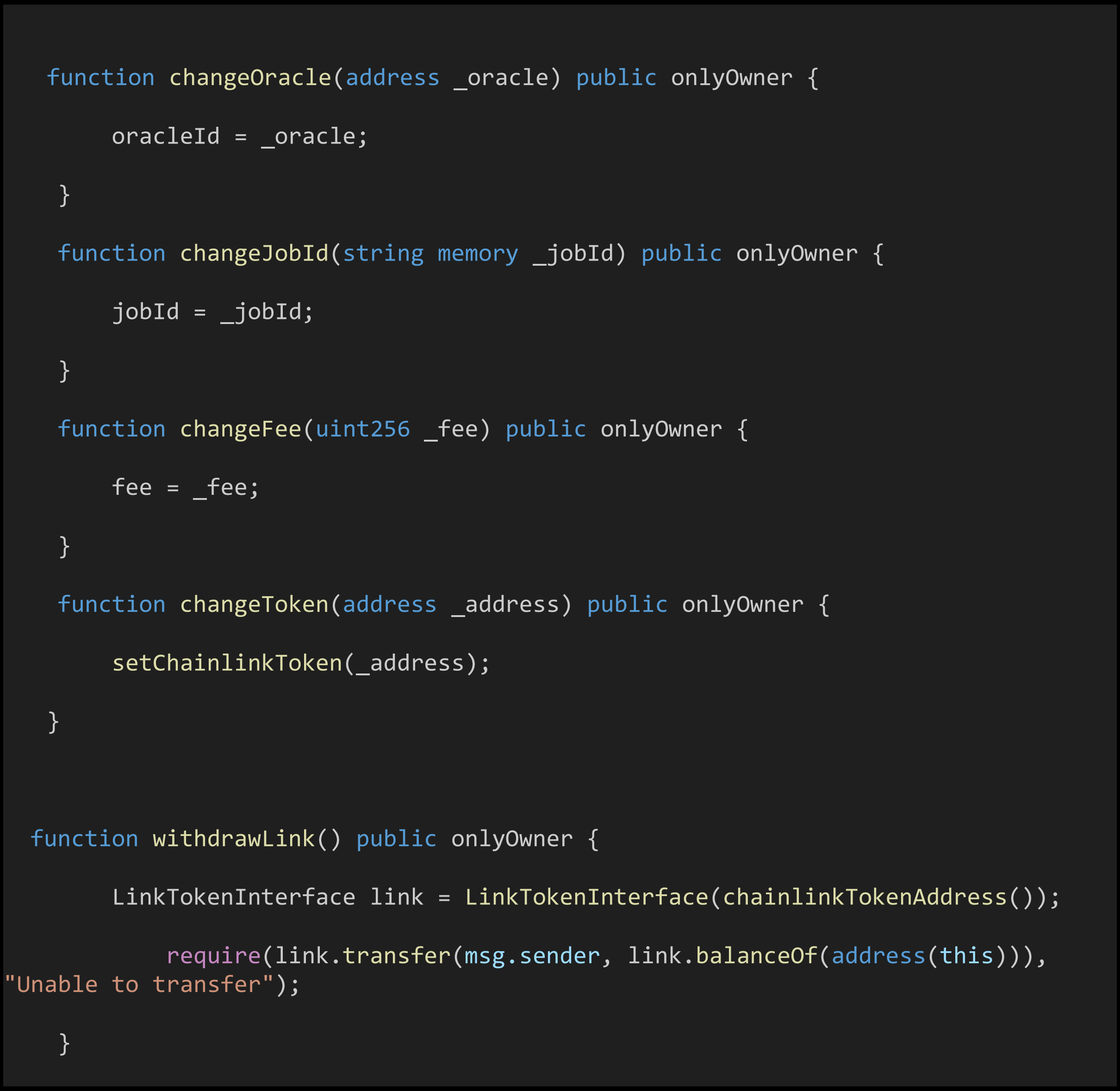
Furthermore , it creates following attack vectors

- Private key of Owner is hacked and hacker takes control of the ownership

- Owner becomes malicious

In Either case , the owner (being an EOA) can non-deterministically interact with following functions of TfiExample and potentially rug users or brick the system deeming it useless.

As only owner , it can also withdraw any accumulated funds.



```
   function changeOracle(address _oracle) public onlyOwner {

        oracleId = _oracle;

    }

    function changeJobId(string memory _jobId) public onlyOwner {

        jobId = _jobId;

    }

    function changeFee(uint256 _fee) public onlyOwner {

        fee = _fee;

    }

    function changeToken(address _address) public onlyOwner {

        setChainlinkToken(_address);

   }


  function withdrawLink() public onlyOwner {

        LinkTokenInterface link = LinkTokenInterface(chainlinkTokenAddress());

            require(link.transfer(msg.sender, link.balanceOf(address(this))),
"Unable to transfer");

    }
```

Technical users , seeing this scenario will be reluctant to use the platform because of rug pull potential of owner.

7. Add missing input validation on constructor/initializer/setters in TfiExample

SEVERITY: MEDIUM

Impact : System can be left non-functional

Remediation

Consider implementing all the checks suggested below

Description

TfiExample.sol#changeOracle :

- ensure new Oracle address is not zero address address(0)
- ensure new oracle is not blocked/abandoned by chainlink due to misbehavior

TfiExample.sol#changeChainlinkToken :

- ensure new chainlinkToken address is not zero address address(0)
- ensure the name returned by this token is `Link`

TfiExample.sol#changeFee :

- ensure new fee is greater than some MIN_FEE & less than some Max_FEE
- declare MIN_FEE and Max_FEE in the contract.

TfiExample.sol#changeJobId :

- ensure new jobId is non-zero

8. OwnableUpgradeable uses single-step ownership transfer

SEVERITY : **MEDIUM**

Impact : Protocol functionality can be bricked

Remediation

It is a best practice to use two-step ownership transfer pattern, meaning ownership transfer gets to a "pending" state and the new owner should claim his new rights, otherwise the old owner still has control of the contract. Consider using OpenZeppelin's Ownable2Step contract

Description

Single-step ownership transfer means that if a wrong address was passed when transferring ownership or admin rights it can mean that role is lost forever. The ownership pattern implementation for the protocol is in OwnableUpgradeable.sol where a single-step transfer is implemented.This can be a problem for all methods marked in onlyOwner throughout the protocol, some of which are core protocol functionality.

9. Protocol fee change in TfiExample should be time-locked

SEVERITY : MEDIUM

Impact : Users will not be able to make calculated decisions might lose funds

Remediation

Add some timelock mechanism to change fee or a delayed two step fee change. For example , make a function proposeFeeChange , which will accept new fee as proposedFee and emit and event on-chain to let others know that fee of transacting on TfiExample is about to be changed and then after some time delay , owner will call confirmFeeChange which will change the fee and community will be notified through new event that fee has been changed.

Description

Inside TfiExample contract's implementation , we have a changeFee , which can change the fee in single step . However , using the smart contract , users will be transacting with their Link token based upon the fee they can afford .

If initially the fee was 5% and a lot of users are transacting because 5% is acceptable for them but owner suddenly changes the fee to 40% due to which the users might still believe that the fee was 5% but they will end up losing millions of dollars if billions of dollars were being transacted just after the fee change.

Big protocols imply a suitable method to first inform the community that the critical parameters like fee are being changed by a two step process (read Remediation).

This way users can make informed decisions about their investment helping them protect their funds.

Closing Remarks

This was a good opportunity given by Techfund team

And Umar has tried his best to uncover potential vulnerabilities in the course of few days.

Props to Polygon Labs for designing such noble solution , techfund team for giving me the challenge and my coffee.

Indeed a great experience and I hope I'll join hands with TechFund to secure the New , More Challenging and More Innovative Web3 protocols along the way.

Thank you for your time

If you want to reach out , please send me an email at

umarkhatabfrl@gmail.com

See you there !

THANK YOU