Security Audit of `bridge-contracts`

Conclusion

Audit was made by “BlockSec” team by Vladimir Smelov vladimirfol@gmail.com (mailto:vladimirfol@gmail.com).

In the final contract were not found:

- Backdoors for investor funds withdrawal by anyone.
- Bugs allowing to steal money from the contract.

The client was acknowledged about all secutiry notes below

Scope

https://github.com/TehnobitSystems/bridge-contracts (https://github.com/TehnobitSystems/bridge-contracts)
commit: `ad0988a231974f8cde084453e1999c387e6aeccd`

also 2 python files from `pysigner` archive

```
$ md5sum *.py
7241d27d4b78abac601af319b84d3c06  signer.py
c8ae14e6c0586b3772f48b92f7787b76  signerserver.py
```

Methodology

1. Blind audit. Try to understand the structure of the code.
2. Find info in internet.

3. Ask quiestions to developers.
4. Draw the scheme of cross-contracts interactions.
5. Write user-stories, usage cases.
6. Run static analyzers

Find problems with:

- backdoors
- bugs
- math
- potential leaking of funds
- potential locking of the contract
- validate arguments and events
- others

Result

Critical

1. [pysigner] Avoid double-spending.

Hash for BscClaim and EthClaim is calculated by the same formula.
It makes it possible to approve cross on the other token.

Recommendation.

- Add salt to hash depending on the blockChain like `bsc` / `eth` .
- Add `assert adminBsc != adminEth` .
- Use blockChain specific data like chainId (see https://ethereum.stackexchange.com/questions/56749/retrieve-chain-id-of-the-executing-chain-from-a-solidity-contract (https://ethereum.stackexchange.com/questions/56749/retrieve-chain-id-of-the-executing-chain-from-a-solidity-contract)).

Status.

NEW

Major

1. Return value is ignored of ERC20 methods.

```
ZEFUVaultETH.swapToBSC(uint256) (BridgeVault.sol#75)
ZEFUVaultETH.swapFromBSC(uint256,bytes) (BridgeVault.sol#90)
```

The return value of the method call is ignored.

Recommendation.

Use SafeERC20 methods.

Status.

NEW

2. Impossible to process the previous user swap.

At

- BridgeToken.sol:187
- BridgeVault.sol:83
- NEWBSCContract.sol:451
  the last `getNonceIn` value is used, which makes it impossible to sign the previous swap. So in the case when someone makes 2 swap one by one and if the signature was not applied yet for the 1th swap, but 2nd is already in progress, it will be not possible to process the 1th swap.

Recommendation.

Pass nonce as an argument.

Status.

NEW

3. [pysigner] Impossible to sign the previous swap.

At

- signer.py:28 (http://signer.py:28)
- signer.py:33 (http://signer.py:33)
  the sign will only process the latest swap.

See also MAJOR-2.

Recommendation.

Add nonce as an argument.

Status.

NEW

4. [pysigner] Do not use Flask built-in WebServer in prod.

At signerserver.py:32 (http://signerserver.py:32)

https://flask.palletsprojects.com/en/2.0.x/tutorial/deploy/
(https://flask.palletsprojects.com/en/2.0.x/tutorial/deploy/)

> When running publicly rather than in development, you should not use the built-in development server (flask run). The development server is provided by Werkzeug for convenience, but is not designed to be particularly efficient, stable, or secure.

https://stackoverflow.com/questions/12269537/is-the-server-bundled-with-flask-safe-to-use-in-production (https://stackoverflow.com/questions/12269537/is-the-server-bundled-with-flask-safe-to-use-in-production)

e.g. if you leave `FLASK_ENV=development` in `.env`

> - If you leave debug mode on and an error pops up, it opens up a shell that allows for arbitrary code to be executed on your server (think os.system('rm -rf /')).

Recommendation.

It is not secure. Do not use it.
Follow recommendations from official docs.

Status.

NEW

Warning

1. Address zero checks

- BridgeVault.sol#28
- BridgeVault.sol#64
- BridgeToken.sol#39
- BridgeToken.sol#92
- NEWBSCContract.sol#413
- NEWBSCContract.sol#440
- NEWBSCContract.sol#509

Recommendation.

Add `require(newAddre != address(0), "ZERO_ADDRESS")`
because it is easy to make a mistake on front-end.

Status.

NEW

2. Potential Reentry may lead to misordering.

in ZEFUVaultETH.swapToBSC
BridgeVault.sol#73-80

you do storage update

```
uint256 nonce = getNonceOut(user);
_transferOut[user].amount[nonce] = amount;
_transferOut[user].nonce++;
```

after external call

ZEFU.transferFrom.

As a developer of `ZEFUVaultETH` you don't know implementation of `ZEFU`, anything can be inside. Even reentry to `swapToBSC` it will lead that order of actual transfer will be different from `nonces`.

Recommendation.

Update storage before external calls.

Status.

NEW

3. Duplication of `BridgedZEFU` (and interface mismatch).

You have BridgedZEFU contract implemented twice.
It's not clear which one to use and what is the difference.
They also have different interfaces (check transfer/approve etc).

Recommendation.

Make it clear.

Status.

NEW

4. [pysigner] Dangerous CORS usage.

At

- signerserver.py:8 (http://signerserver.py:8)
  you use CORS to allow any domain request.
  This is not really safe production.

Recommendation.

Disallow CORS for any domain, be strict.

Status.

NEW

Comment.

1. Define methods as external.

Various of methods you don't use inside the contract, but they have `public` access modifier.

Recommendation.

Use `external` to save gas.

Status.

NEW

2. Incorrect ERC20 interface.

```
NEWBSCContract.sol#155-163
NEWBSCContract.sol#170-180
NEWBSCContract.sol#92
NEWBSCContract.sol#93
NEWBSCContract.sol#82
NEWBSCContract.sol#119-123
NEWBSCContract.sol#462-468
NEWBSCContract.sol#471-477
NEWBSCContract.sol#489-495
```

does not match ERC20 interface because they don't have return-value.

Also, not that you return something in BridgedZEFU contract but methods defined as return nothing.

Recommendation.

Add

```
returns(uint256)
return true;
```

Status.

NEW

3. Lack of event.

NEWBSCContract.sol#67

Recommendation.

Add

```
emit NewOwner(newOwner);
```

Status.

NEW

4. Better naming

```
Cross -> UserSwaps
_transferIn -> _userTransferIn
_transferOut -> _userTransferOut
```

5. You don’t need SafeMath.

Since solidity 0.8.0 you don’t need SafeMath this is already included to the compiler. Using of SafeMath is wasting of gas.

Recommendation.

Remove SafeMath.

Status.

NEW

6. Optimizae boolean operations.

NEWBSCContract.sol#110
(1)

```
require(!(msg.data.length < size + 4));
```

(2)

```
require(!((_value != 0) && (allowed[msg.sender][_spender] != 0)));
```

Recommendation.

(1)

```
require(msg.data.length >= size + 4, "bad length");
```

(2)

```
require((_value == 0) || (allowed[msg.sender][_spender] == 0), "new value or ol
```

Also, don’t forget to add revert-message.

Status.

NEW

7. Save gas.

```
uint256 nonce = getNonceOut(user);
_transferOut[user].amount[nonce] = amount;
_transferOut[user].nonce++;
```

Recommendation.

```
_transferOut[user].amount[
    _transferOut[user].nonce++
] = amount;
```

or even

```
Cross storage cross = _transferOut[user];
cross.amount[cross.nonce++] = amount;
```

Status.

NEW

8. Redundant `public` for variable.

NEWBSCContract.sol#145

```
mapping (address => mapping (address => uint)) public allowed;
```

but you already have `allowance` getter

Recommendation.

```
mapping (address => mapping (address => uint)) internal allowed;
```

Status.

NEW

9. Disable internal solidity SafeMath to save gas.

BridgeToken.sol#110
BridgeToken.sol#123
BridgeToken.sol#136
BridgeToken.sol#159
the same in other contracts, in fact compiler will check the same condition twice.

Recommendation.

See
https://github.com/OpenZeppelin/openzeppelin-contracts/issues/2465#issuecomment-758314712 (https://github.com/OpenZeppelin/openzeppelin-contracts/issues/2465#issuecomment-758314712)

do something like

```
require(amount >= balance, "Insufficient balance");
unchecked {
  balance -= amount;
}
```

Status.

NEW

10. Argument validation in external methods.

in
_transfer
_mint
_burn

etc you have validation of address arguments.
But you can move these checks on external call level and save some gas because some checks will turn out to be redundant or duplicated.

Recommendation.

Do argument validation in external methods.

Status.

NEW

11. Proper way to get uint256 maxValue.

```
uint public constant MAX_UINT = 2**256 - 1;
```

this looks strange because 2**256 == 0.

Recommendation.

Use `type(uint256).max`

Status.

NEW

12. Safe approve for BridgeToken.

In NEWBSCContract you use

```
require(!((_value != 0) && (allowed[msg.sender][_spender] != 0)));
```

but not in BridgeToken.

Recommendation.

Be consistent. Use such check in BridgeToken also.

Status.

NEW

13. Code duplication.

`library ECDSA` implemented twice in project.

Recommendation.

Avoid code duplication.

Status.

NEW

14. Use imports from OpenZeppelin.

A lot of universal code is used.

Recommendation.

It’s better to keep codebase as small as possible and use imports from external projects.

Status.

NEW

15. Redundant set.

NEWBSCContract.sol#415

```
deprecated = false;
_totalSupply = 0;
```

but this is zero by default.

Recommendation.

Remove redundant code.

Status.

NEW

16. Prohibit `deprecate` calls multiple times.

At NEWBSCContract.sol#507
you can set new implementation as many times as you want.
I’m not sure if it is supposed to work like this.

Recommendation.

```
require(!deprecated, "already deprecated");
```

Status.

NEW

17. logic duplication in bulkTransfer.

At NEWBSCContract.sol#525

```
balances[msg.sender] = balances[msg.sender].sub(amounts[i]);
balances[addrs[i]] = balances[addrs[i]].add(amounts[i]);
emit Transfer(msg.sender, addrs[i], amounts[i]);
```

but you can use `transfer`, also not that implementation may change due to upgrade, but this code will be frozen.

Recommendation.

Use `transfer` (don't forget to check success).

Status.

NEW

18. Use noReentrant for external methods.

It is always good to add `noReentrant` modifier from openzeppelin reentryGuard to completely avoid reentry attacks.

Recommendation.

Use reentryGuard and sleep well.

Status.

NEW

19. Avoid keeping every Cross swap in the storage.

Storage is expensive but you store every Cross swap in storage forever, but it is not really need for the contract logic.

Recommendation.

Optimize the usage of storage.

But be careful about marking the swap as finished (or even better freeing up the storage to have gas refund, you can use mapping for that) to avoid double spending.

Status.

NEW

20. Unclear external methods purpose.

It is not clear what is the purpose of the methods.

- getAmountIn
- getAmountOut

Recommendation.

Remove or add docstring.
Think about retreiving this information from events history.

Status.

NEW

21. Unclear purpose of the storage variables.

It is not clear what is the purpose of the variables.

- _transferIn
- _transferOut

Recommendation.

Remove or add docstring.

Status.

NEW

22. Unclear revert message.

“Invalid transaction.” does not really say what is invalid.

Recommendation.

Use something like.
“Not validator signature”

Status.

NEW

23. [pysigner] Follow language code-style.

- Optimize imports.
- Follow PEP-8.

Recommendation.

Remove or add docstring.

Status.

NEW

24. [pysigner] No need to use `getHash` method.

You can calculate the hash by direct web3 method and also avoid blockChain RPC Node call.

Recommendation.

Implement on python web3.

Or you can include

```
nonce=self.vault.functions.getNonceIn(user).call()
amount=self.peg.functions.getAmountOut(user, nonce).call()
self.vault.functions.getHash(user, nonce, amount).call()
```

as a contract method to get hash to sign in one RPC call.

Status.

NEW

25. [pysigner] Close the file descriptor after usage.

At

- signer.py:14 (http://signer.py:14)
- signer.py:16 (http://signer.py:16)

You do not close the file descriptor.
You use redundant default argument ‘r’.
You do not need to use loads.

Recommendation.

```
with open('vault.abi') as f:
    VAULT_ABI=json.load(f)
```

Status.

NEW