BABB

BAX Token

SMART CONTRACT AUDIT

10.05.2022

Made in Germany by Chainsulting.de

Table of contents

1. Disclaimer

The audit makes no statements or warrantees about utility of the code, safety of the code, suitability of the business model, investment advice, endorsement of the platform or its products, regulatory regime for the business model, or any other statements about fitness of the contracts to purpose, or their bug free status. The audit documentation is for discussion purposes only.

The information presented in this report is confidential and privileged. If you are reading this report, you agree to keep it confidential, not to copy, disclose or disseminate without the agreement of Babb Platform Ltd. If you are not the intended receptor of this document, remember that any disclosure, copying or dissemination of it is forbidden.

Major Versions / Date	Description
0.1 (04.05.2022)	Layout
0.4 (05.05.2022)	Automated Security Testing Manual Security Testing
0.5 (07.05.2022)	Verify Claims and Test Deployment
0.6 (08.05.2022)	Testing SWC Checks
0.9 (09.05.2022)	Summary and Recommendation
1.0 (09.05.2022)	Final document
1.1 (12.05.2022)	Added deployed contract

2. About the Project and Company

Company address:

Babb Platform Ltd
40 Bank Street 18th Floor
London, E14 5NR, UK

Website: https://getbabb.com

Twitter: https://twitter.com/getbabb

Facebook: https://www.facebook.com/getbabb

Telegram: https://t.me/babb_official

BitcoinTalk: https://bitcointalk.org/index.php?topic=2349340.0

Reddit: https://reddit.com/r/getBABB

LinkedIn: https://www.linkedin.com/company/babb

YouTube: https://www.youtube.com/channel/UCQtkZd7sfzbEugz7VdFhv4Q

Medium: https://babb.medium.com

GitHub: https://github.com/babbplatform

2.1 Project Overview

BABB is building a decentralized bank. Anyone with a smartphone can open a UK bank account, and if you're fully verified yourself, you can actually on board your friends and family. These integrated digital identities allow for quick on boarding for anyone, especially those living in areas where access to banking may be harder to get. This innovative and ambitious project also has its own token behind it, and that allows for much more freedom than what would be allowed with a normal banking system.

Users will even be able to gain access to a payment card that is independent of systems used by Visa or Mastercard. This blockchain project is targeting developing nations which may have a high percentage of citizens without access to a bank account, but with a growing number of mobile phone users, making their solution a perfect addition to these economies.

3. Vulnerability & Risk Level

Risk represents the probability that a certain source-threat will exploit vulnerability, and the impact of that event on the organization or system. Risk Level is computed based on CVSS version 3.0.

Level	Value	Vulnerability	Risk (Required Action)
Critical	9 – 10	A vulnerability that can disrupt the contract functioning in a number of scenarios, or creates a risk that the contract may be broken.	Immediate action to reduce risk level.
High	7 – 8.9	A vulnerability that affects the desired outcome when using a contract, or provides the opportunity to use a contract in an unintended way.	Implementation of corrective actions as soon as possible.
Medium	4 – 6.9	A vulnerability that could affect the desired outcome of executing the contract in a specific scenario.	Implementation of corrective actions in a certain period.
Low	2 – 3.9	A vulnerability that does not have a significant impact on possible scenarios for the use of the contract and is probably subjective.	Implementation of certain corrective actions or accepting the risk.
Informational	0 – 1.9	A vulnerability that have informational character but is not effecting any of the code.	An observation that does not determine a level of risk

4. Auditing Strategy and Techniques Applied

Throughout the review process, care was taken to evaluate the repository for security-related issues, code quality, and adherence to specification and best practices. To do so, reviewed line-by-line by our team of expert pentesters and smart contract developers, documenting any issues as there were discovered.

4.1 Methodology

The auditing process follows a routine series of steps:

1. Code review that includes the following:
   i. Review of the specifications, sources, and instructions provided to Chainsulting to make sure we understand the size, scope, and functionality of the smart contract.
   ii. Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
   iii. Comparison to specification, which is the process of checking whether the code does what the specifications, sources, and instructions provided to Chainsulting describe.
2. Testing and automated analysis that includes the following:
   i. Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run those test cases.
   ii. Symbolic execution, which is analysing a program to determine what inputs causes each part of a program to execute.
3. Best practices review, which is a review of the smart contracts to improve efficiency, effectiveness, clarify, maintainability, security, and control based on the established industry and academic practices, recommendations, and research.
4. Specific, itemized, actionable recommendations to help you take steps to secure your smart contracts.

5. Metrics

The metrics section should give the reader an overview on the size, quality, flows and capabilities of the codebase, without the knowledge to understand the actual code.

5.1 Tested Contract Files

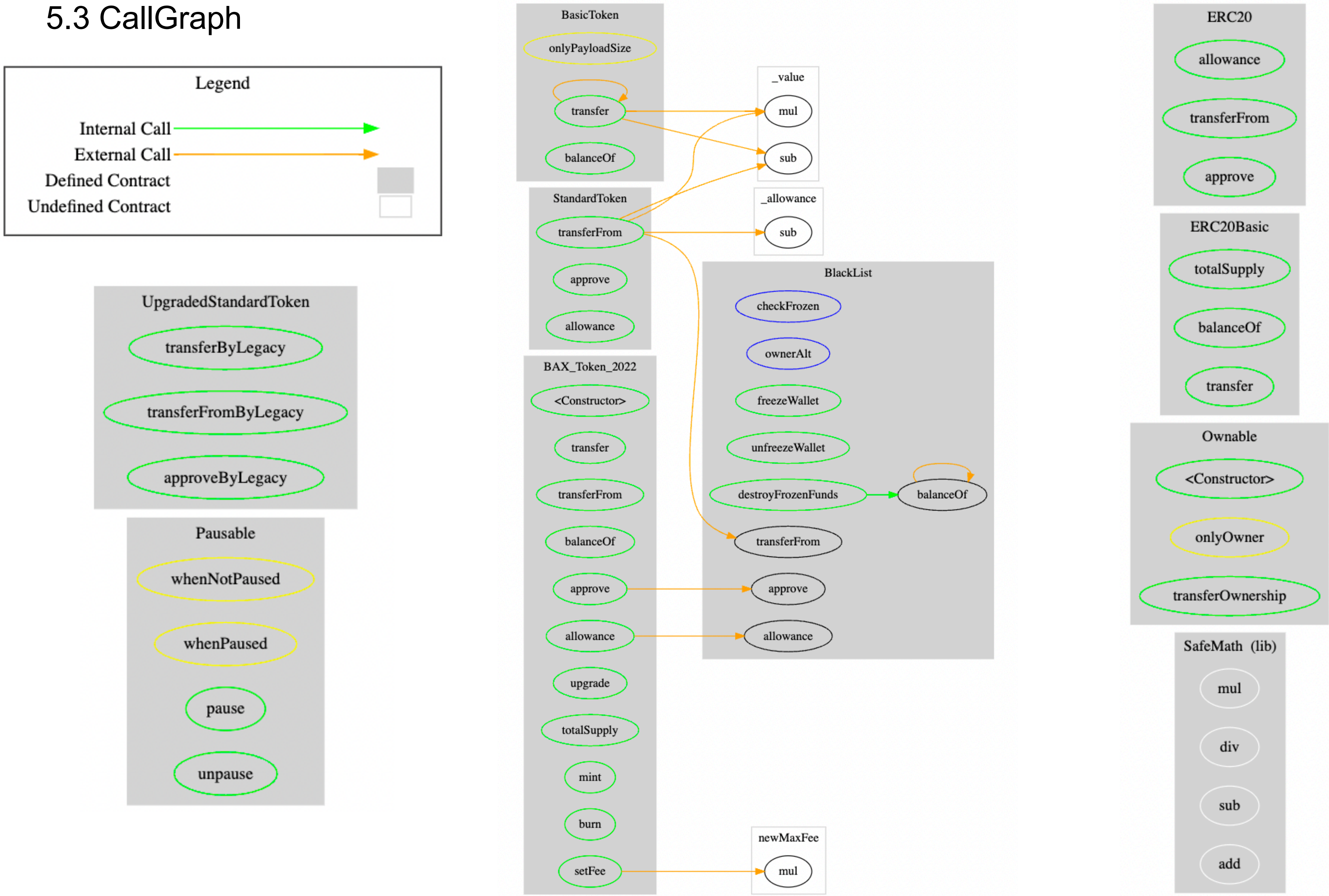
The following are the MD5 hashes of the reviewed files. A file with a different MD5 hash has been modified, intentionally or otherwise, after the security review. You are cautioned that a different MD5 hash could be (but is not necessarily) an indication of a changed condition or potential vulnerability that was not within the scope of the review

File	Fingerprint (MD5)
./BAX_Token.sol	c4c947486c86dfcc569cff2b927a5ee6

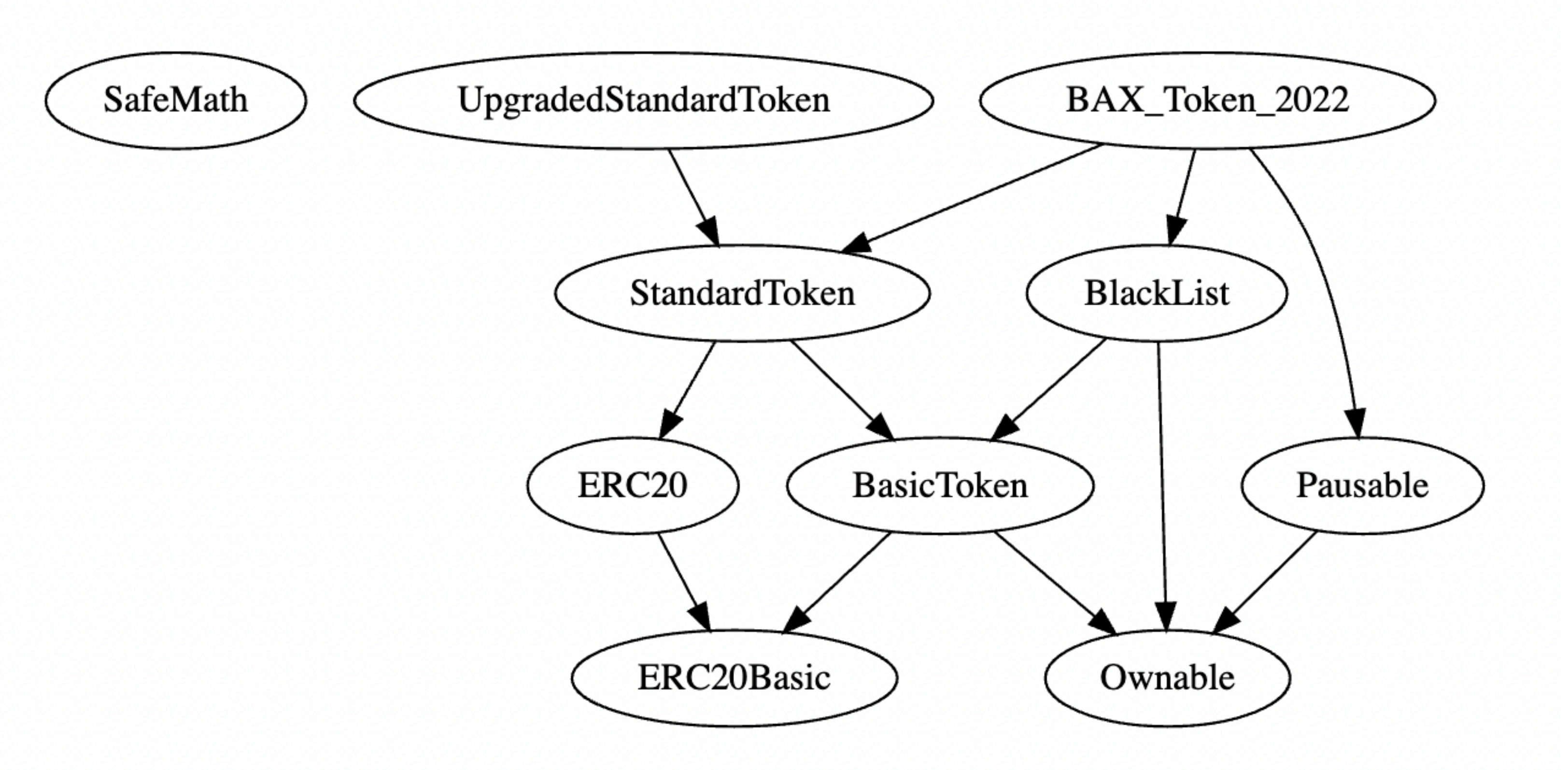
5.2 Used Code from other Frameworks/Smart Contracts (direct imports)

Dependency / Import Path	Source
Ownable.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v1.0.2/contracts/ownership/Ownable.sol
SafeMath.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v1.0.2/contracts/SafeMath.sol
ERC20Basic.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v1.0.2/contracts/token/ERC20Basic.sol
ERC20.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v1.0.2/contracts/token/ERC20.sol
BasicToken.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v1.0.2/contracts/token/BasicToken.sol
Pausable.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v1.0.2/contracts/lifecycle/Pausable.sol

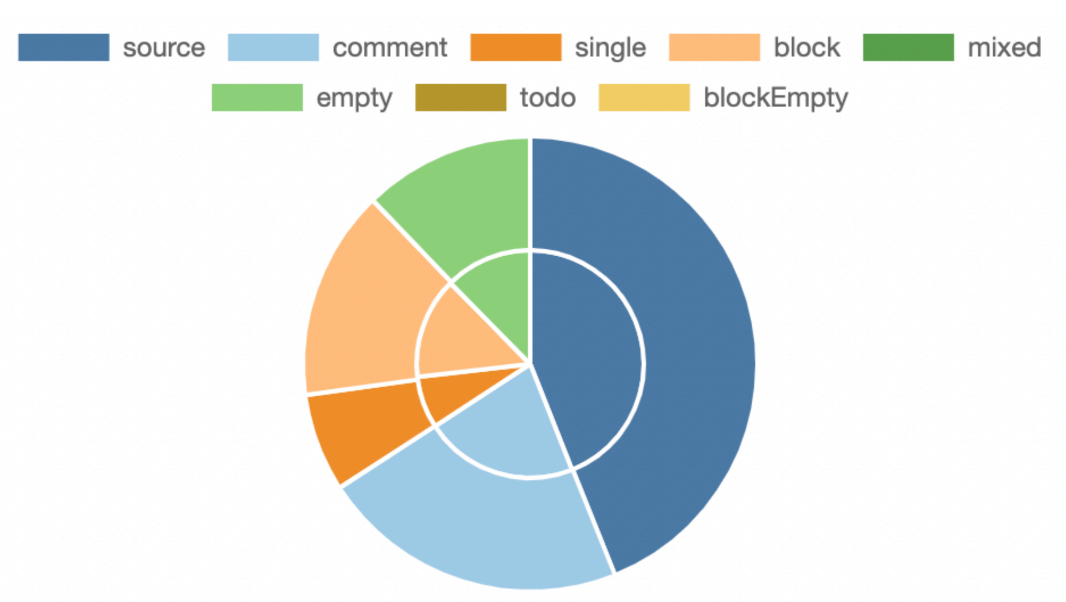
5.3 CallGraph

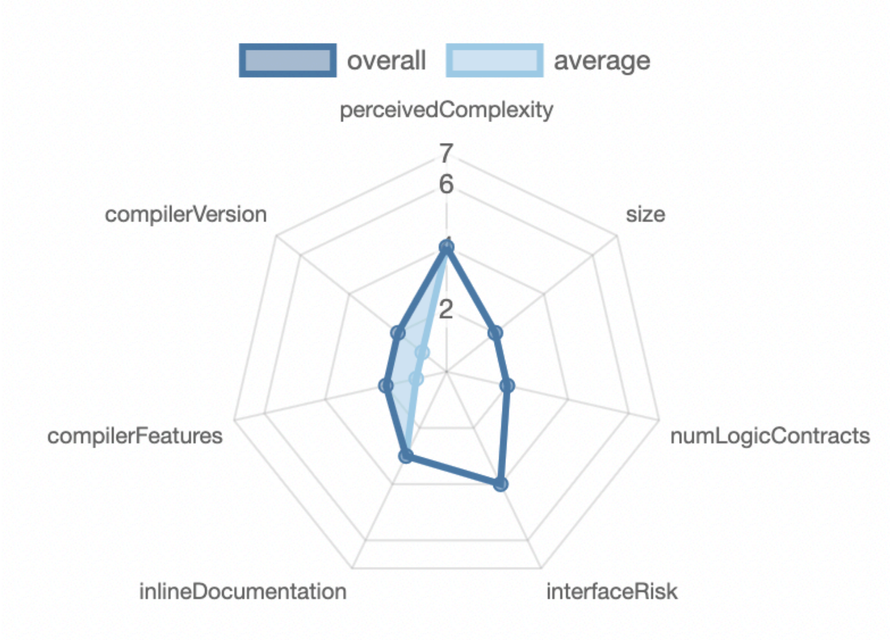


5.4 Inheritance Graph



5.5 Source Lines & Risk





5.6 Capabilities

Solidity Versions observed	🧪 **Experimental Features**	💰 **Can Receive Funds**	🖥 **Uses Assembly**	💣 **Has Destroyable Contracts**
`^0.4.17`				

📤 **Transfers ETH**	⚡ **Low-Level Calls**	👥 **DelegateCall**	🧮 **Uses Hash Functions**	🔖 **ECRecover**	🌀 **New/Create/Create2**
`yes`					

Exposed Functions

This section lists functions that are explicitly declared public or payable. Please note that getter methods for public stateVars are not included.

🌐**Public**	💰**Payable**
34	0

External	**Internal**	**Private**	**Pure**	**View**
2	28	0	4	0

StateVariables

Total	🌐**Public**
15	15

5.7 Source Unites in Scope

Source: https://ropsten.etherscan.io/address/0x059e85b98bea55eeccf5284e4d6633a44eb6be58#code

Type	File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score	Capabilities
📝📚	BAX_Token.sol	10		448	423	238	126	212	📤
📝📚	**Totals**	**10**		**448**	**423**	**238**	**126**	**212**	📤

Legend: [➖]

- **Lines**: total lines of the source unit
- **nLines**: normalized lines of the source unit (e.g. normalizes functions spanning multiple lines)
- **nSLOC**: normalized source lines of code (only source-code lines; no comments, no blank lines)
- **Comment Lines**: lines containing single or block comments
- **Complexity Score**: a custom complexity score derived from code statements that are known to introduce code complexity (branches, loops, calls, external interfaces, ...)

6. Scope of Work

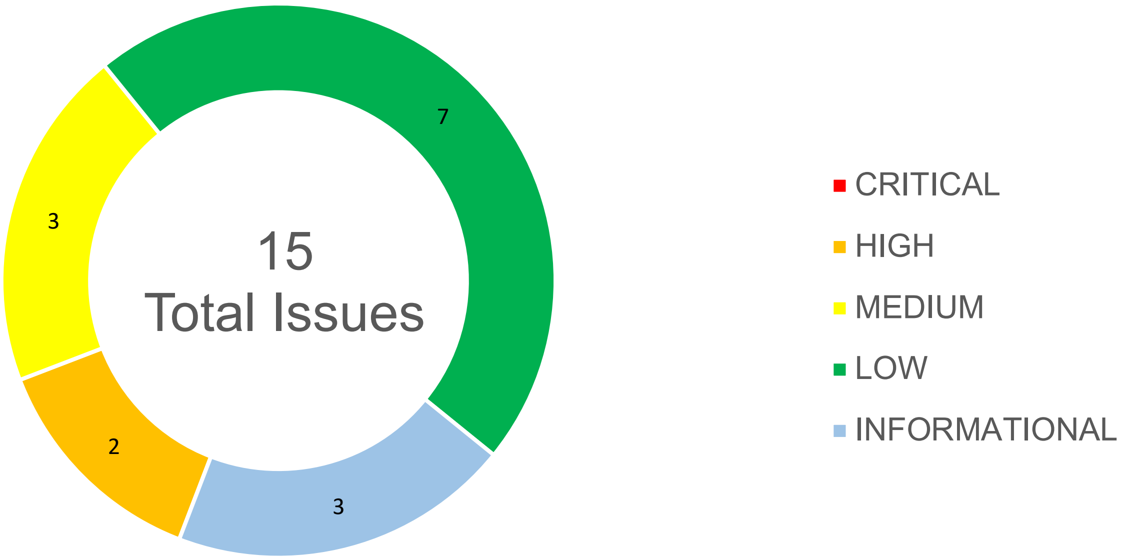
The BABB Team provided us with the files that needs to be tested. The scope of the audit is the new BAX token contract.

The team put forward the following assumptions regarding the security, usage of the contracts:

- The ERC-20 Token standard is correctly implemented
- Deployer/Owner cannot mint any new Token
- Deployer/Owner cannot burn or lock user funds
- Deployer/Owner cannot pause the contract
- The smart contract is coded according to the newest standards and in a secure way.

The main goal of this audit was to verify these claims. The auditors can provide additional feedback on the code upon the client's request.

6.1 Findings Overview



No	Title	Severity	Status
6.2.1	Owner Can Burn Any User Funds	HIGH	ACKNOWLEDGED
6.2.2	Owner Can Freeze User Funds	HIGH	ACKNOWLEDGED
6.2.3	Outdated Compiler Version	MEDIUM	ACKNOWLEDGED
6.2.4	Unexplicit Types	MEDIUM	ACKNOWLEDGED
6.2.5	Replacement Of Contract Logic	MEDIUM	ACKNOWLEDGED
6.2.6	Wrong Import Of OpenZeppelin Library	LOW	ACKNOWLEDGED
6.2.7	Deprecated Use Of Constant	LOW	ACKNOWLEDGED
6.2.8	Missing OwnershipTransferred Event	LOW	ACKNOWLEDGED
6.2.9	Use Require Messages	LOW	ACKNOWLEDGED

6.2.10	Deprecated Var Keyword	LOW	ACKNOWLEDGED
6.2.11	Redundant Function	LOW	ACKNOWLEDGED
6.2.12	Unlimited Minting To Max Cap	LOW	ACKNOWLEDGED
6.2.13	Floating Pragma Version Identified	INFORMATIONAL	ACKNOWLEDGED
6.2.14	Improper Use Of Interfaces	INFORMATIONAL	ACKNOWLEDGED
6.2.15	Variable Usage Before Definition	INFORMATIONAL	ACKNOWLEDGED

6.2 Manual and Automated Vulnerability Test

CRITICAL ISSUES

During the audit, Chainsulting‘s experts found **0 Critical issues** in the code of the smart contract.

HIGH ISSUES

During the audit, Chainsulting’s experts found **2 High issues** in the code of the smart contract.

6.2.1 Owner Can Burn Any User Funds

Severity: HIGH
Status: ACKNOWLEDGED
Code: NA
File(s) affected: BAX_Token.sol
Update: Due to the nature of regulated and centralized stablecoins, this functions are required.

<table>
<tr><td><b>Attack / Description</b></td><td>After blacklisting a user, the owner is able to burn funds from the user.</td></tr>
<tr><td><b>Code</b></td><td>Line 288 - 294 (BAX_Token.sol)
<pre><code>function destroyFrozenFunds (address _blackListedUser) public onlyOwner {
        require(checkFrozenAlt[_blackListedUser]);
        uint dirtyFunds = balanceOf(_blackListedUser);
        balances[_blackListedUser] = 0;
        totalSupplyAlt -= dirtyFunds;
        DestroyedFrozenFunds(_blackListedUser, dirtyFunds);
    }
</code></pre></td></tr>
</table>

Result/Recommendation	It is highly recommended to remove overpowered owner rights like burning funds. In case this function is required by regulations of stablecoins, the owner rights should be secured by a multi-signature wallet.

6.2.2 Owner Can Freeze User Funds

Severity: HIGH
Status: ACKNOWLEDGED
Code: NA
File(s) affected: BAX_Token.sol
Update: Due to the nature of regulated and centralized stablecoins, this functions are required.

Attack / Description	The contract owner can blacklist a user at any time. Afterwards the selected user funds are frozen, they are able to be transferred.

Code	Line 278 - 281 (BAX_Token.sol)
	`function freezeWallet (address _evilUser) public onlyOwner {`
	`checkFrozenAlt[_evilUser] = true;`
	`WalletFrozen(_evilUser);`
	`}`

Result/Recommendation	It’s highly recommended to remove overpowered owner rights like freezing funds. In case this function is required by regulations of stablecoins, the owner rights should be secured by a multi-signature wallet.

MEDIUM ISSUES

During the audit, Chainsulting's experts found **3 Medium issues** in the code of the smart contract.

6.2.3 Outdated Compiler Version

Severity: MEDIUM
Status: ACKNOWLEDGED
Code: SWC-102
File(s) affected: BAX_Token.sol

Attack / Description	The used compiler version ^0.4.17 was released in September 2017 and is deprecated. The current solidity compiler version is 0.8.13.
Code	Line 1 (BAX_Token.sol) `pragma solidity ^0.4.17;`
Result/Recommendation	It is highly recommended to use an up-to-date version of solidity, which is not too recent. Currently we recommend using solidity 0.8.0, which requires to upgrade the whole contract, which would also fix all issues at once.

6.2.4 Unexplicit Types

Severity: MEDIUM
Status: ACKNOWLEDGED
Code: NA
File(s) affected: BAX_Token.sol

Attack / Description	In the current implementation unexplicit types of unsigned integer variables are used.
Code	NA
Result/Recommendation	It is recommended to use explicit types of uint (i.e. uint256, uint128, …) to clarify storage usage and expected size of numbers. It is also helpful to reduce storage of structs with multiple integer variables due to packing.

6.2.5 Replacement Of Contract Logic

Severity: MEDIUM
Status: ACKNOWLEDGED
Code: NA
File(s) affected: BAX_Token.sol

Attack / Description	The logic of the underlying token contract can be replaced by the owner at any time. This replacement can be unaudited malicious code and implement various attack vectors, if the owner wallet is breached / hacked.

Code	Line 384 - 388 (BAX_Token.sol)
	
	`function upgrade(address _upgradeContract) public onlyOwner {`
	`      upgradeApplied = true;`
	`      upgradeContract = _upgradeContract;`
	`      Upgrade(_upgradeContract);`
	`  }`

Result/Recommendation	It is recommended to remove the implementation of replacing underlying contract logic. If the contract should be changeable use upgradeable contract from OpenZeppelin and implement a multi-signature on the business side, to ensure maximal security for the owner wallet.

LOW ISSUES

During the audit, Chainsulting's experts found **7 Low issues** in the code of the smart contract.

6.2.6 Wrong Import Of OpenZeppelin Library

Severity: LOW
Status: ACKNOWLEDGED
Code: NA
File(s) affected: BAX_Token.sol

Attack / Description	In the current implementation, OpenZeppelinfiles are added directly into the code. This violates OpenZeppelin's MIT license, which requires the license and copyright notice to be included if its code is used. Moreover, updating code manually is error-prone.
Code	Ownable, SafeMath, ERC20Basic, ERC20, BasicToken, Pausable
Result/Recommendation	We highly recommend using npm (import "@openzeppelin/contracts/..) in order to guarantee that original OpenZeppelin contracts are used with no modifications. This also allows for any bug-fixes to be easily integrated into the codebase.

6.2.7 Deprecated Use Of Constant

Severity: LOW
Status: ACKNOWLEDGED
Code: NA

File(s) affected: BAX_Token.sol

Attack / Description	The “constant” keyword on functions is deprecated and was removed on 0.5.0. It is an alias for the newer view keyword.
Code	Lines: 79, 80, 95, 164, 244, 295, 299, 417, 445, 463
Result/Recommendation	We recommend removing constant function modifiers and replace them with view.

6.2.8 Missing OwnershipTransferred Event

Severity: LOW
Status: ACKNOWLEDGED
Code: NA
File(s) affected: BAX_Token.sol

Attack / Description	The current implementation of ownable library is missing the OwnershipTransferred event, which is normally emitted on transferring ownership.

Code	Line 64 - 68 (BAX_Token.sol)
	
	`function transferOwnership(address newOwner) public onlyOwner {`
	`        if (newOwner != address(0)) {`
	`            owner = newOwner;`
	`        }`
	`    }`

Result/Recommendation	We recommend adding the OwnershipTransferred event. The best practice is the usage of OpenZeppelins Ownable library.

6.2.9 Use Require Messages

Severity: LOW
Status: ACKNOWLEDGED
Code: NA
File(s) affected: BAX_Token.sol

Attack / Description	The current implementation is not providing any feedback on failing actions.
Code	NA
Result/Recommendation	It is recommended to update the compiler version and use require messages to provide feedback for failing transactions.

6.2.10 Deprecated Var Keyword

Severity: LOW
Status: ACKNOWLEDGED
Code: NA
File(s) affected: BAX_Token.sol

Attack / Description	The unexplicit var keyword has been deprecated for security reasons in 0.4.20 and should not be used anymore.
Code	Line 169 (BAX_Token.sol) `var _allowance = allowed[_from][msg.sender];`

Result/Recommendation	We highly recommend using uint256 instead of the deprecated var keyword. https://docs.soliditylang.org/en/v0.4.21/control-structures.html?highlight=var#destructuring-assignments-and-returning-multiple-values

6.2.11 Redundant Function

Severity: LOW
Status: ACKNOWLEDGED
Code: NA
File(s) affected: BAX_Token.sol

Attack / Description	In the current implementation are redundant functions to get the owner of the BlackList.

Code	Line 272 - 274 (BAX_Token.sol)
	
	`function ownerAlt() external constant returns (address) {`
	`        return owner;`
	`    }`

Result/Recommendation	We recommend removing the redundant ownerAlt function.

6.2.12 Unlimited Minting To Max Cap

Severity: LOW
Status: ACKNOWLEDGED
Code: NA
File(s) affected: BAX_Token.sol

Attack / Description	In the current implementation the owner is able to mint new tokens at any time until the max cap is reached.

Code	Line 403 - 410 (BAX_Token.sol)
	`function mint(uint amount) public onlyOwner {`
	`        require(totalSupplyAlt + amount > totalSupplyAlt);`
	`        require(balances[owner] + amount > balances[owner]);`
	`        require(totalSupplyAlt + amount  <= MAX_CAP);`
	`        balances[owner] += amount;`
	`        totalSupplyAlt += amount;`
	`        Mint(amount);`
	`    }`

Result/Recommendation	It is highly recommended to remove overpowered owner rights like uncontrolled minting of funds. If the function is needed cause of the usage as stablecoin, we recommend that the owner rights should be secured by a multi-signature wallet.

INFORMATIONAL ISSUES

During the audit, Chainsulting's experts found **3 Informational issues** in the code of the smart contract.

6.2.13 Floating Pragma Version Identified

Severity: INFORMATIONAL
Status: ACKNOWLEDGED
Code: SWC-103
File(s) affected: BAX_Token.sol

Attack / Description	It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.
Code	Line 1 `^0.4.17`
Result/Recommendation	It is recommended to follow the latter example, as future compiler versions may handle certain language constructions in a way the developer did not foresee. It is advised that floating pragma should not be used in production. Both truffle-config.js and hardhat.config.js support locking the pragma version. i.e. Pragma solidity 0.4.17

6.2.14 Improper Use Of Interfaces

Severity: INFORMATIONAL
Status: ACKNOWLEDGED
Code: NA
File(s) affected: BAX_Token.sol

Attack / Description	In the current implementation contract keyword is used for interfaces.
Code	NA
Result/Recommendation	We recommend using interface keyword to define interfaces to clarify its occurrence and usage.

6.2.15 Variable Usage Before Definition

Severity: INFORMATIONAL
Status: ACKNOWLEDGED
Code: NA
File(s) affected: BAX_Token.sol

Attack / Description	In the current implementation a variable is used before it was defined.

Code	Line 268 - 270 (BAX_Token.sol)
	`function checkFrozen(address _maker) external constant returns (bool) {`
	`        return checkFrozenAlt[_maker];`
	`    }`
	Line 276 (BAX_Token.sol)

	`mapping (address => bool) public checkFrozenAlt;`
Result/Recommendation	It is recommended to define variables before using them to enhance code readability.

6.3 SWC Attacks

ID	Title	Relationships	Test Result
SWC-131	Presence of unused variables	CWE-1164: Irrelevant Code	✅
SWC-130	Right-To-Left-Override control character (U+202E)	CWE-451: User Interface (UI) Misrepresentation of Critical Information	✅
SWC-129	Typographical Error	CWE-480: Use of Incorrect Operator	✅
SWC-128	DoS With Block Gas Limit	CWE-400: Uncontrolled Resource Consumption	✅
SWC-127	Arbitrary Jump with Function Type Variable	CWE-695: Use of Low-Level Functionality	✅
SWC-125	Incorrect Inheritance Order	CWE-696: Incorrect Behavior Order	✅
SWC-124	Write to Arbitrary Storage Location	CWE-123: Write-what-where Condition	✅
SWC-123	Requirement Violation	CWE-573: Improper Following of Specification by Caller	✅

ID	Title	Relationships	Test Result
SWC-122	Lack of Proper Signature Verification	CWE-345: Insufficient Verification of Data Authenticity	✅
SWC-121	Missing Protection against Signature Replay Attacks	CWE-347: Improper Verification of Cryptographic Signature	✅
SWC-120	Weak Sources of Randomness from Chain Attributes	CWE-330: Use of Insufficiently Random Values	✅
SWC-119	Shadowing State Variables	CWE-710: Improper Adherence to Coding Standards	✅
SWC-118	Incorrect Constructor Name	CWE-665: Improper Initialization	✅
SWC-117	Signature Malleability	CWE-347: Improper Verification of Cryptographic Signature	✅
SWC-116	Timestamp Dependence	CWE-829: Inclusion of Functionality from Untrusted Control Sphere	✅
SWC-115	Authorization through tx.origin	CWE-477: Use of Obsolete Function	✅
SWC-114	Transaction Order Dependence	CWE-362: Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')	✅

ID	Title	Relationships	Test Result
SWC-113	DoS with Failed Call	CWE-703: Improper Check or Handling of Exceptional Conditions	✅
SWC-112	Delegatecall to Untrusted Callee	CWE-829: Inclusion of Functionality from Untrusted Control Sphere	✅
SWC-111	Use of Deprecated Solidity Functions	CWE-477: Use of Obsolete Function	✅
SWC-110	Assert Violation	CWE-670: Always-Incorrect Control Flow Implementation	✅
SWC-109	Uninitialized Storage Pointer	CWE-824: Access of Uninitialized Pointer	✅
SWC-108	State Variable Default Visibility	CWE-710: Improper Adherence to Coding Standards	✅
SWC-107	Reentrancy	CWE-841: Improper Enforcement of Behavioral Workflow	✅
SWC-106	Unprotected SELFDESTRUCT Instruction	CWE-284: Improper Access Control	✅
SWC-105	Unprotected Ether Withdrawal	CWE-284: Improper Access Control	✅
SWC-104	Unchecked Call Return Value	CWE-252: Unchecked Return Value	✅

ID	Title	Relationships	Test Result
SWC-103	Floating Pragma	CWE-664: Improper Control of a Resource Through its Lifetime	X
SWC-102	Outdated Compiler Version	CWE-937: Using Components with Known Vulnerabilities	X
SWC-101	Integer Overflow and Underflow	CWE-682: Incorrect Calculation	✅
SWC-100	Function Default Visibility	CWE-710: Improper Adherence to Coding Standards	✅

6.4 Verify Claims

6.4.1 The ERC-20 Token standard is correctly implemented
Status: tested and verified (Note: outdated libraries are used) ✅

6.4.2 Deployer/Owner cannot mint any new Token
Status: Owner is able to mint new Token until MAX_CAP

6.4.3 Deployer/Owner cannot burn or lock user funds
Status: Owner is able to burn and lock user funds

6.4.4 Deployer/Owner cannot pause the contract
Status: Owner is able to pause the contract

6.4.5 The smart contract is coded according to the newest standards and in a secure way.
Status: Outdated compiler version and libraries used

6.5 Token Upgrade Notice

OLD BABB BAX Token: https://etherscan.io/token/0x9a0242b7a33dacbe40edb927834f96eb39f8fbcb
CoinMarketCap: https://coinmarketcap.com/currencies/babb/
Coingecko: https://www.coingecko.com/en/coins/babb
Audit: https://blog.openzeppelin.com/bax-token-audit-85fe7b186c89/amp/

NEW BABB BAX Token: https://etherscan.io/address/0xf920e4f3fbef5b3ad0a25017514b769bdc4ac135#code

7. Executive Summary

Two (2) independent Chainsulting experts performed an unbiased and isolated audit of the smart contract codebase.

The main goal of the audit was to verify the claims regarding the security and functions of the smart contract. During the audit, no critical, 2 high, 3 medium, 7 low and 3 informational issues have been found, after the manual and automated security testing. We highly recommended to use the latest compiler version and OpenZeppelin libraries.

8. Deployed Smart Contract

VERIFIED

https://etherscan.io/address/0xf920e4f3fbef5b3ad0a25017514b769bdc4ac135#code

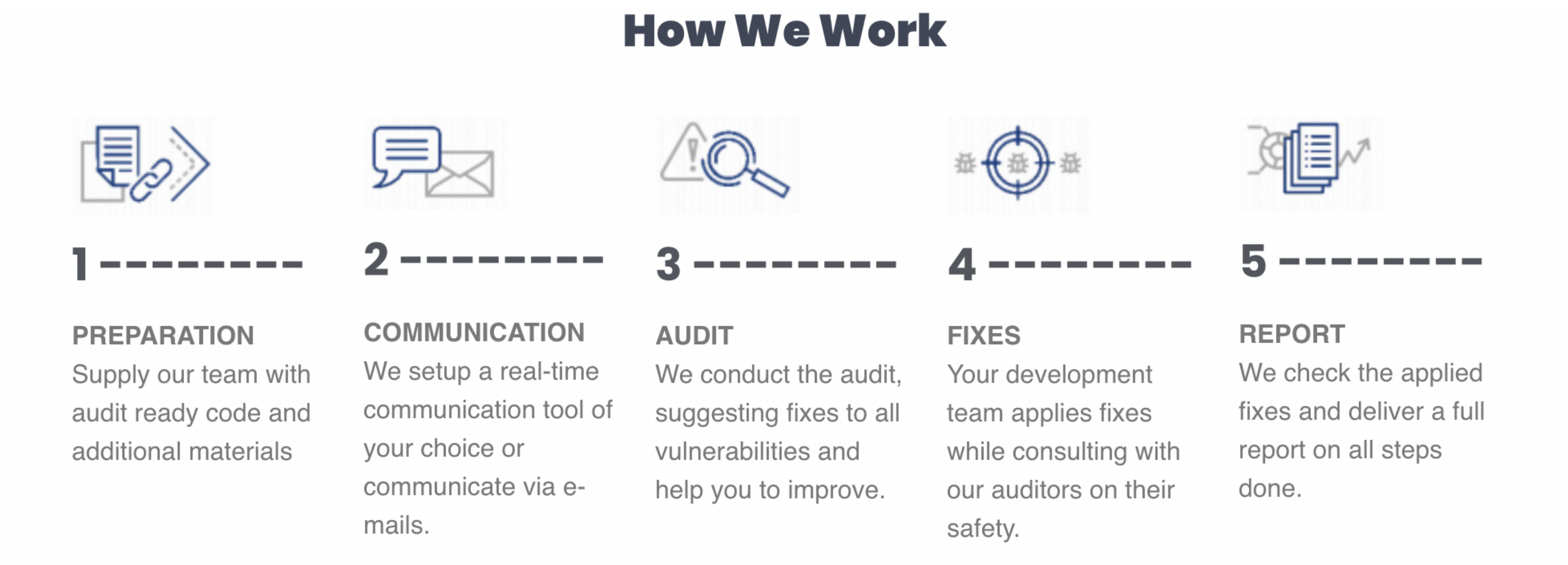
9. About the Auditor

Chainsulting is a professional software development firm, founded in 20217 and based in Germany. They show ways, opportunities, risks and offer comprehensive blockchain solutions. Some of their services include blockchain development, smart contract audits and consulting.

Chainsulting conducts code audits on market-leading blockchains such as Hyperledger, Tezos, Ethereum, Binance Smart Chain, and Solana to mitigate risk and instil trust and transparency into the vibrant crypto community. They have also reviewed and secure the smart contracts of 1Inch, POA Network, Unicrypt, Amun, Furucombo among numerous other top DeFi projects.

Chainsulting currently secures $100 billion in user funds locked in multiple DeFi protocols. The team behind the leading audit firm relies on their robust technical know-how in the blockchain sector to deliver top-notch smart contract audit solutions, tailored to the clients' evolving business needs.

Check our website for further information: https://chainsulting.de



How We Work

1 --------

PREPARATION
Supply our team with audit ready code and additional materials

2 --------

COMMUNICATION
We setup a real-time communication tool of your choice or communicate via e-mails.

3 --------

AUDIT
We conduct the audit, suggesting fixes to all vulnerabilities and help you to improve.

4 --------

FIXES
Your development team applies fixes while consulting with our auditors on their safety.

5 --------

REPORT
We check the applied fixes and deliver a full report on all steps done.