CERTIK

Audit Report

Produced by CertiK

for

Sep 11th, 2019

Contents

Disclaimer

This Report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Verification Services Agreement between CertiK and WaltonChain(the "Company"), or the scope of services/verification, and terms and conditions provided to the Company in connection with the verification (collectively, the "Agreement"). This Report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This Report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes without CertiK's prior written consent.

About CertiK

CertiK is a technology-led blockchain security company founded by Computer Science professors from Yale University and Columbia University built to prove the security and correctness of smart contracts and blockchain protocols.

CertiK, in partnership with grants from IBM and the Ethereum Foundation, has developed a proprietary Formal Verification technology to apply rigorous and complete mathematical reasoning against code. This process ensures algorithms, protocols, and business functionalities are secured and working as intended across all platforms.

CertiK differs from traditional testing approaches by employing Formal Verification to mathematically prove blockchain ecosystem and smart contracts are hacker-resistant and bug-free. CertiK uses this industry-leading technology together with standardized test suites, static analysis, and expert manual review to create a full-stack solution for our partners across the blockchain world to secure 6.2B in assets.

For more information: https://certik.org/

Executive Summary

This report has been prepared as the product of the Smart Contract Audit request by WaltonChain. This audit was conducted to discover issues and vulnerabilities in the source code of WaltonChain's Smart Contracts. Utilizing CertiK's Formal Verification Platform, Static Analysis, and Manual Review, a comprehensive examination has been performed. The auditing process pays special attention to the following considerations.

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessment of the codebase for best practice 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.

Vulnerability Classification

For every issue found, CertiK categorizes them into 3 buckets based on its risk level:

Critical

The code implementation does not match the specification, or it could result in loss of funds for contract owner or users.

Medium

The code implementation does not match the specification at certain conditions, or it could affect the security standard by lost of access control.

Low

The code implementation is not a best practice, or use a suboptimal design pattern, which may lead to security vulnerabilies, but no concern found yet.

Testing Summary

Type of Blockchain Issues

CertiK review the blockchain source code according to the various blockchain layers as Application, Contract, Insensitive, Consensus, Network and Data to evaluate the follow type of risk and common attacks.

Blockchain Layer	Description	Risk
Application	Providing the development environment for implementing the blockchain business related scenario and logic	Centralized resource, architecture design, and cyber security management
Contract	Containing a set of rules on the blockchain using various hash algorithm and computer code for triggering the blockchain process automatically	Smart contract and virtual machine vulnerability
Insensitive	Distributing rewards to participants who contribute to maintain the blockchain ecosystem	Liveness of nodes
Consensus	A protocol that describes the format of a ledger that is publicly visible and a consensus function that anyone can use to determine which of multiple candidate ledgers is the consensus ledger. The protocol must also allow new blocks to be added to the ledger. This is the core layer in the blockchain ecosystem and mainly corresponding to consensus, mining, semantic and propagation layers	Centralized, data correctness, liveness of miners, stability of nodes
Network	Propagating or broadcasting transactions among nodes.	RPC interface exposure, network resource consumption
Data	Storing the raw blockchain block data and structure, hash functions, digital signature and cryptographic protocol	Data Integrity

Vulnerability	Description	Categories
Eclipse attack	A node will depend on “x” number of nodes selected using a Peer selection strategy to have its view of the distributed ledger. But if an attacker can manage to make the node to choose all the “x” number of nodes from his malicious nodes alone, then he can eclipse the original ledger’s view and present his own manipulated ledger to the node	Peer-to-Peer Network-based Attacks

Sybil attack	While the Eclipse attack is about eclipsing a user's view of the true ledger, the Sybil attack targets the whole network. In a Sybil attack, an attacker will flood the network with large number of nodes with pseudonymous identity and try to influence the network. These nodes, though appearing like unrelated individuals, are operated by a single operator at the back. In this case the objective is not to target one user, but a number of nodes or network as whole, and generate a fork in the ledger if possible, allowing the attacker to make double spending and other attacks	Peer-to-Peer Network-based Attacks
Selfish mining attack	Many blockchains consider the longest chain to be the true latest version of the ledger. So a selfish miner can try to keep building blocks in stealth mode on top of the existing chain, and when he can build a lead of greater than two or more blocks than the current chain in the network, he can publish his private fork, which will be accepted as a new truth as it is the longest chain. He can do transactions in the public network just before publishing his longer stealth chain to reverse the transaction he just did	Consensus Mechanism and Mining-based Attacks
Mining malware	Malware uses the computing power of unsuspecting victims' computer to mine cryptocurrencies for hackers	Consensus Mechanism and Mining-based Attacks
51% attack	This attack is possible when a miner or a group of miners controls 51% or more of the mining power of the blockchain network. Though it is very difficult to happen for large networks, the possibility of a 51% attack is higher in small networks. Once a group has majority control over transactions on a blockchain network, it can prevent specific transaction or even reverse older transactions	Consensus Mechanism and Mining-based Attacks
Timejack attack	Nodes in certain blockchain networks like Bitcoin depend on internal timing derived from median time reported by its peer nodes. For example, you depend on your friends to know the time. Let us say an attacker manages to put a lot of malicious people in your friends' list, then he can manipulate your time. The first step to this attack can be an Eclipse attack on the target node. Once this attack is complete on a target node, then the target node will not accept blocks from the actual network as the timestamp of the blocks will not be in line with its timestamp. This provides an opportunity for the attacker to be double spending or do transactions with the targeted node as these transactions can't be submitted to the actual blockchain network	Consensus Mechanism and Mining-based Attacks

Finney attack	If you can mine a block with one of your transactions in it and keep it in stealth, there is an opportunity for you to double spend the money. If a merchant accepts the unconfirmed transaction, you can transfer him this earlier transacted currency. Next you publish the earlier mined block, which was kept in stealth, before your new transaction is confirmed on network	Consensus Mechanism and Mining-based Attacks
Race attack	This attack is minor variation of the Finney attack. The difference is that the attacker need not pre-mine the block with his transaction, which he intends to double spend. During the attack, the attacker submits an unconfirmed transaction to a merchant (victim) and simultaneously does another transaction which he broadcasts to the network. It is easier for the attacker to launch the attack if he is directly connected to the merchant's node. This would give the merchant an illusion that his transaction is the first, but that is never submitted to the blockchain network by the attacker	Consensus Mechanism and Mining-based Attacks
Distributed denial of service	Distributed denial of service (DDoS) attacks are hard to execute on a blockchain network. Still, blockchain technology is susceptible to DDoS attacks and these attacks are actually the most common type on blockchain networks. When attacking a blockchain network, hackers intend to bring down a server by consuming all its processing resources with numerous requests. DDoS attackers aim to disconnect mining pools, e-wallets, crypto exchanges, and other financial services of the network. A blockchain can also be hacked with DDoS at its application layer when hackers use DDoS botnets	Network Attack

Type of Smart Contract Issues

CertiK smart label engine applied 100% coveraged formal verification labels on the source code, and scanned the code using our proprietary static analysis and formal verification engine to detect the follow type of issues.

Title	Description	Issues	SWC ID
Integer Overflow and Underflow	An overflow/underflow happens when an arithmetic operation reaches the maximum or minimum size of a type.	0	SWC-101
Function incorrectness	Function implementation does not meet the specification, leading to intentional or unintentional vulnerabilities.	0	
Buffer Overflow	An attacker is able to write to arbitrary storage locations of a contract if array of out bound happens	0	SWC-124

Reentrancy	A malicious contract can call back into the calling contract before the first invocation of the function is finished.	0	SWC-107
Transaction Order Dependence	A race condition vulnerability occurs when code depends on the order of the transactions submitted to it.	0	SWC-114
Timestamp Dependence	Timestamp can be influenced by minors to some degree.	0	SWC-116
Insecure Compiler Version	Using an fixed outdated compiler version or floating pragma can be problematic, if there are publicly disclosed bugs and issues that affect the current compiler version used.	0	SWC-102 SWC-103
Insecure Randomness	Block attributes are insecure to generate random numbers, as they can be influenced by minors to some degree.	0	SWC-120
"tx.origin" for authorization	tx.origin should not be used for authorization. Use msg.sender instead.	0	SWC-115
Delegatecall to Untrusted Callee	Calling into untrusted contracts is very dangerous, the target and arguments provided must be sanitized.	0	SWC-112
State Variable Default Visibility	Labeling the visibility explicitly makes it easier to catch incorrect assumptions about who can access the variable.	0	SWC-108
Function Default Visibility	Functions are public by default. A malicious user is able to make unauthorized or unintended state changes if a developer forgot to set the visibility.	0	SWC-100
Uninitialized variables	Uninitialized local storage variables can point to other unexpected storage variables in the contract.	0	SWC-109
Assertion Failure	The assert() function is meant to assert invariants. Properly functioning code should never reach a failing assert statement.	0	SWC-110
Deprecated Solidity Features	Several functions and operators in Solidity are deprecated and should not be used as best practice.	0	SWC-111
Unused variables	Unused variables reduce code quality	0	

Vulnerability Details

Critical

No issue found.

Medium

No issue found.

Low

Categories Breakdown	Issues
Application Layer	No issue found
Contract Layer	No issue found
Insensitive Layer	No issue found
Consensus Layer	No critical issues found that impacting the current stage, highly recommend for taking consideration improvement for the long-term
Network Layer	No issue found
Data Layer	No issue found
Smart Contract	No critical issues found that impacting the current stage, highly recommend for taking consideration improvement for the long-term

Manual Review Summary

Project Description

Launched in 2016, the Waltonchain (WTC) project is building an ecosystem that melds Blockchain, RFID technology, and IoT (Internet of Things) with complete data exchange and absolute information transparency based on the blockchain. The Walton team develops and produces Transaction ID-reading RFID chips, which can generate their own random ID hashes that are uploaded simultaneously to the blockchain via their RFID reader. The RFID technology can be used in various sectors requiring identification (credit cards, mobile payments, magnetic keys, etc.) This translates to enhanced operational efficiency, especially for supply chain use cases such high-end clothing identification, food & drug traceability, and logistics tracking.

Waltonchain has been implementing its RFID and other technical applications in the clothing, food, collection, logistics, and other industries. The team mainly focuses on two aspects: data reliability and data value circulation. By seamlessly integrating Blockchain and FRID to ensure data reliability right from the source, Waltonchain brings outstanding effectiveness of data sharing by leveraging its cross-chain ecosystem.

Scope of Work

CertiK was chosen by WaltonChain to audit the design and implementation of golang version blockchain, which is a fork of `go-ethereum 1.7.1` and its cross chain smart contracts. To ensure comprehensive protection,the source code has been analyzed by the proprietary CertiK manually reviewed by our blockchain and smart contract experts and engineers. That end-to-end process ensures proof of stability as well as a hands-on, engineering-focused process to close potential loopholes and recommend design changes in accordance with the best practices in the space.

	File Modified
1	accounts/abi/bind/backends/simulated.go
2	accounts/keystore/key.go
3	build/ci.go
4	cmd/bootnode/main.go
5	cmd/geth/chaincmd.go
6	`cmd/geth/dao_test.go`
7	cmd/geth/main.go
8	cmd/geth/misccmd.go
9	cmd/geth/usage.go
10	`cmd/puppeth/wizard_genesis.go`
11	`cmd/puppeth/wizard_node.go`
12	cmd/utils/flags.go
13	common/big.go
14	common/now.go
15	consensus/consensus.go
16	consensus/ethash/algorithm.go
17	consensus/ethash/consensus.go
18	consensus/ethash/ethash.go
19	`consensus/ethash/ethash_test.go`

20	consensus/ethash/sealer.go
21	consensus/misc/dao.go
22	console/console.go
23	core/block_validator.go
24	core/blockchain.go
25	core/chain_makers.go
26	core/dao_test.go
27	core/evm.go
28	core/genesis.go
29	core/headerchain.go
30	core/state/dump.go
31	core/state/journal.go
32	core/state/state_object.go
33	core/state/statedb.go
34	core/state_processor.go
35	core/state_transition.go
36	core/tx_journal.go
37	core/tx_pool.go
38	core/types/block.go
39	core/types/block_test.go
40	core/types/transaction.go
41	core/vm/evm.go
42	core/vm/instructions.go
43	core/vm/interface.go
44	core/vm/runtime/runtime.go
45	crypto/crypto.go
46	crypto/x/x.go
47	crypto/x/x_test.go
48	eth/api_backend.go
49	eth/backend.go
50	eth/config.go
51	eth/db_upgrade.go
52	eth/downloader/downloader.go
53	eth/handler.go
54	eth/handler_test.go
55	eth/sync.go
56	ethdb/database.go
57	internal/webext/webext.go
58	les/api_backend.go
59	les/backend.go
60	les/fetcher.go
61	les/handler.go
62	light/lightchain.go
63	miner/agent.go
64	miner/miner.go
65	miner/remote_agent.go
66	miner/unconfirmed.go
67	miner/worker.go
68	node/config.go

69	node/node.go
70	pp/discover/node.go
71	pp/discover/udp.go
72	pp/server.go
73	params/bootnodes.go
74	params/config.go
75	params/protocol_params.go
76	params/version.go
77	rpc/server.go
78	tests/block_test_util.go
79	tests/init.go
80	tests/state_test_util.go
81	tests/vm_test_util.go

Source Of Truth

CertiK used the following source of truth to enhance the understanding of WaltonChain's systems:

1. WaltonChain Whitepaper[1]
2. WaltonChain Website[2]
3. WaltonChain Github[3]
4. WaltonChain Business Process[4]

All listed sources act as specification. For any inconsistency discovered between the actual code behavior and the specification, CertiK would consult with the WaltonChain team for further DICUSSION and confirmation.

Blockchain audit checklist

The blockchain source code audit will conduct and focus on answering the following listed areas, checkpoints and harm level. (✓ indicates satisfaction; × indicates unsatisfaction; − indicates partial satisfaction)

Key Management

Ensuring the all key generation meet following checkpoints:

✓ Private Key & Mnemonic Generation Correctness

✓ Private Key & Mnemonic Storage Safety Management

✓ Private Key & Mnemonic Confidentiality

[1]Whitepaper: https://www.waltonchain.org/Uploads/2019-04-25/5cc1721e671c9.pdf
[2]Website: https://www.waltonchain.org/
[3]Github: https://github.com/WaltonChain/WaltonChain_Gwtc_Src
[4]Business Process: https://www.waltonchain.org/en/sys/cate/48.html

Cryptography

Ensuring the correctness and fairness of the hash functions, signatures, and keys generation

- ✓ Random hash algorithm correctness in terms of hash function, and signature
- ✓ Hash generation within normal distribution probability

Consensus

Ensuring the consensus model design, and model correctness as described in the Source of trusts list

- – Consensus model design rationality
- – Consensus implementation correctness as described design

RPC

Ensuring the RPC interface, port, and exposure with restricted access for preventing vulnerability signature attack, and sensitive information leaking

- ✓ Sensitive information & interface accessibility

Chain Processing

Ensuring the chain is processing and synchronizing with longest selection and switching algorithm appropriately.

- ✓ Longest Chain selection and switching algorithm
- ✓ Chain synchronization logic/workflow correctness

Block Processing

Ensuring the block hash function, signature, and mertkle tree structure correctness for malicious activities caused by faking block data

- – Block processing resource limitation as for orphan block pool, verification calculation etc
- ✓ Block signatures and merkle tree root construction correctness

Transaction Processing

Ensuring the correctness of transaction process for minimizing the chances of lost transaction, low cost DoS attack, double spending attack, and replay attack

- ✓ Transaction fee model design
- ✓ Transaction processing logic & workflow
- ✓ Hash collision

Misc

× Compatibility with latest go-version.
This is always a good practice for keeping programming language to the latest version, it benefits with toolchain, compiler, packages and standard library new features and fixes.

× Update the vendor folders for obtaining the latest version of the third party libraries with its new features and bug fix updates.

× High Test Coverage.
Implementing the unit test as many as possible for ensuring function behavior is meeting its specification. A lot of time, unit-test can discover many unexpected vulnerabilities at the early stage of product release before the lost.

× Provide System testing or End-to-End testing scripts.
Strongly recommend to development some test workflows and scenarios covering the critical business process for capturing the errors at the beginning. This also would benefit for software update and release process later.

– Project github documentation with latest update.
Strongly recommend update the project Readme file for better guideline to other audiences.

– Whitepaper and technical design documentation with latest update.
Strongly recommend update the whitepaper and technical design documents with the latest changes make due to the business model and requirement changes.

Golang Finding Summary

Go Version Incompatible

- MINOR According to the project `Readme.md`, WaltonChain support Go1.7+, however the current project won't be able to compile with Go1.10+.
  - WALTONCHAIN: For now, the project supports w/ Go1.7 && Go1.8. Other go version supports will be corporate later.

```
Go version 1.10+ compatible issue:
    While installing the WaltonChain with go version 1.10+, below error
        return:
    \# github.com/wtc/go-wtc/vendor/github.com/rjeczalik/notify
    vendor/github.com/rjeczalik/notify/watcher_fsevents_cgo.go:51:52: cannot
         use nil as type _Ctype_CFAllocatorRef in assignment
    vendor/github.com/rjeczalik/notify/watcher_fsevents_cgo.go:162:40:
        cannot use nil as type _Ctype_CFAllocatorRef in argument to
        _Cfunc_CFStringCreateWithCStringNoCopy
    vendor/github.com/rjeczalik/notify/watcher_fsevents_cgo.go:162:55:
        cannot use nil as type _Ctype_CFAllocatorRef in argument to
        _Cfunc_CFStringCreateWithCStringNoCopy
    vendor/github.com/rjeczalik/notify/watcher_fsevents_cgo.go:163:40:
        cannot use nil as type _Ctype_CFAllocatorRef in assignment
```

  - INFO: The issue can be addressed by updating the vendor folder with following changes:

```
line51:
    var source = C.CFRunLoopSourceCreate(C.kCFAllocatorDefault, 0, \&C.
        CFRunLoopSourceContext{
        perform: (C.CFRunLoopPerformCallBack)(C.gosource),
    })
    line:162:
    p := C.CFStringCreateWithCStringNoCopy(C.kCFAllocatorDefault, C.CString(
        s.path), C.kCFStringEncodingUTF8, C.kCFAllocatorDefault)
    line:163:
    path := C.CFArrayCreate(C.kCFAllocatorDefault, (*unsafe.Pointer)(unsafe.
        Pointer(\&p)), 1, nil)
```

- MINOR `build/ci.go` Consider using library for go-version checking.
- MINOR `accounts/keystore/key.go` Missing the sanity check for `keyAddr == nil` and `keyAddrr[:8]` possible lead to index out of range Update the comment for `keyFileName` , the behaviour is not same as described.

Consensus

- DICUSSION WaltonChain is implementing a customize X11 algorithm, how is the algorithm works and its purpose?
  - WaltonChain The main difference between `myX11` and other X11 algorithms that on the market is the algorithm replacement policy. The intention of this design is to keep the hash-rate at a mineable rate for attracting more miners to work on the at the beginning. Due to the WaltonChain business intention, it requires with a block rate of 30s. The replacement policy work as following:
    * The 11 algorithms are classifying into two subsets: `metaDiscard` and `metaReplace` .
    * If any of the selected hash function is contained within the `metaDiscard`, then it will replace by hash function from `metaReplace`.
- INFO The x11 hash functions relative time can be projected as following:
  - A=blake
  - B=bmw
  - C=groestl
  - D=jh
  - E=keccak
  - F=skein
  - G=luffa
  - H=cubehash
  - I=shavite
  - J=simd

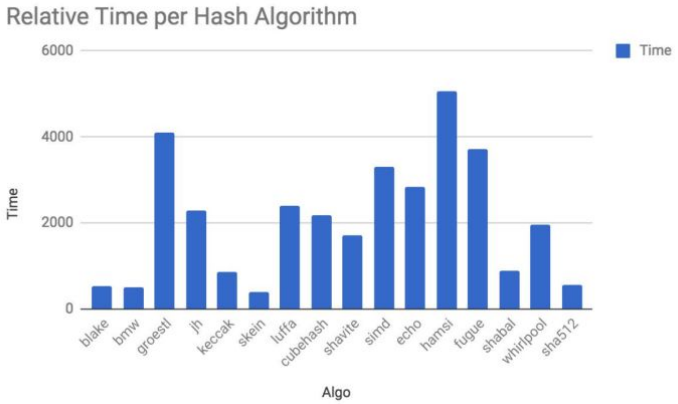

Figure 1: Relative Time per Hash Algorithm

  - K=echo

- Individual Hash function time consumption (ordering from low to high):
  - skein(F), bmw(B), blake(A), keccak(E), shavite(I), cubehash(H), jh(D), luffa(G), echo(K), simd(J), groestl (C)
  - `metaDiscard`: **CFIJK**
  - `metaReplace`: **AHDGEB**

  Over the 7 days, the `miningpoolstats` statistic indicated that the hashrate is in a average range of 235.33 GH/s

- DICUSSION Why use header number hash X11 order?
  - WaltonChain The reason of using the header number hash is because of someone could change other fields in the header (e.g. extra) to manipulate X11 order. keccak256 (change extra field) vs. X11 (increasing nonce).

- DICUSSION How is the POS consensus protocol work in WaltonChain?
  - WaltonChain Compare to traditional POS, WaltonChain's POS offers mining with staking economic model. For those who staking WTC token miners, they can obtain more rewards and 75% discount regarding the difficulty while mining, also no slashing and penalty More more detail of the **Waltonchain Progressive Mining Reward Program** can be found here.

- DICUSSION Why the coinage is removed from the codebase?
  - WaltonChain The concept of coinage is deprecated after the malicious attack. The attackers manipulate the coinage for receiving more mining rewards. The issue is addressed, and the patch has been released. For more, the mainnet upgrade press released by June 29, 2019 can be found here.
  - INFO The change commit between v1.1.2 to v1.1.3:

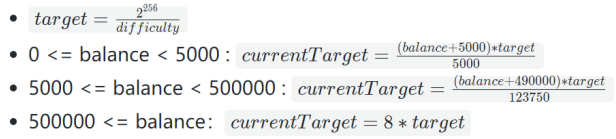

Figure 2: PoS Algorithm

1. Fixed the usage of sha256 on header.Number in `consensus/ethash/consensus.go` and `consensus/ethash/sealer.go`. (edited)
2. Divided the calculation of difficulty into two stages in `consensus/ethash/consensus.go`. (edited)
3. Reduced the minGasLimit to be compared with in `params/protocol_params.go` and updated the comparison in `consensus/ethash/consensus.go` and `core/block_validator.go`.
4. The PoS algorithm updated with decreasing the mining difficulty along with the increase of WTC balance in the mining address.

- DICUSSION: The codebase unused the uncle block, however in ethereum `uncle block` means more than just a reward block, it also benefit for minimizing the centralize problem that POW will facing in the long run.
  - Ethereum's average block time at the time of this writing is around 13-17 seconds. How would ethereum reduce the wastage with such a low block time, and also reduce the chance of multiple, frequent forks. Greedy Heaviest Observed Subtree (GHOST) protocol was created for solving this issue. The main challenge in shorter block time is, there will more miners producing the same block, and end up with no economic incentives — and waste a lot of computational power with no impact towards the stability of the network. Further, this will result in more frequent forks. When a fork happens, the network proceeds by finding the longest blockchain and every miner will switch to it. GHOST solves the problem by including these stale/orphan blocks in the calculation of finding which chain is the longest chain — and also rewarding them. In ethereum these blocks are known as `uncle blocks`. An `uncle block` receives some percentage of the normal block reward — so the computational power spent on mining the stale blocks are not wasted with no economic incentives.
  - Not all uncle blocks can be rewarded. In ethereum, a stale block can only be included as an uncle by up to the seventh-generation descendant of one of its direct siblings, and not any block with a more distant relation. This was done for several reasons.

    Reference

- `consensus/ethash/algorithm.go`
  - MAJOR `Sqrt()`, is not behavoiur as the function named, please renamed the function to nth-Root(), Please review this reference for calculating a precise n-th root function.

    * WaltonChain the function is taking 6 times sqrt root of a given number.
  - MAJOR `Sqrt()`, using big.Int might caused issue in number precision, consider using the big.Float instead.
  - MINOR `log2()` Recommend to removed if not used. The function behavior as returning log(n) + 1

Best practice

Smart contract development requires a particular engineering mindset. A failure in the initial construction can be catastrophic, and changing the project after the fact can be exceedingly difficult.

To ensure success and to avoid the challenges above smart contracts should here to best practices at their conception. Below, we summarized a checklist of key points that help to indicate a high overall quality of the current WaltonChain cross-chain project. (✓ indicates satisfaction; × indicates unsatisfaction; − indicates inapplicablility)

General

- ✓ Corrent environment settings, e.g. compiler version, test framework
- ✓ No compiler warnings
- ✓ Provide error message along with `assert`
- × Use events to monitor contract activities
- ✓ Correct time dependency

Solidity Specific

- − Safe external call
- − No reentrancy pattern
- ✓ Correct handling of integer overflow and underflow
- ✓ Correct visibility for state variables

Privilege Control

- ✓ Provide pause functionality for control and emergency handling
- ✓ Provide time buffer between certain operations
- ✓ Provide proper access control for functions
- ✓ Establish rate limit for certain operations
- ✓ Restrict access to sensitive functions
- ✓ Restrict permission to contract destruction

Documentation

✓ Provide project README and execution guidance

✓ Provide inline comment for function intention

✓ Provide instruction to initialize and execute the test files

Testing

× Provide migration scripts

× Provide test scripts and coverage for potential scenarios

Smart Contract

The smart contracts are designed for transfering the facilitator of data transfers between the various sidechains to WaltonChain parent chain. Side chain collect transactions and bundle 50 txns and pushing to parent chain. The WTC tokens are required for the transactions to be complete.

- `Ballot.sol`: 737383ee58c5511b3188fcc8675f528b6b6cf482214a7fd41a72561ce8c2f5ca
- `PermissionManageContract.sol`: 7fd97c86123affb44eac2b2fb05dc436dc29617cc9a7783dce3d4dfb855dcd35
- `SubchainManageContract.sol`: 339f50199d2964190be0d254097512ba956e0fd3575ab7cae969f3f0cc8056e9
- `TemplateManageContract.sol`: c2cac36efd30ba049432fe2dcf302549be2988344913c5e1e91a58d22a605f7a

Ballot.sol

- INFO Consider using `SafeMath` Library for all math operations to prevent overflow when working with `uint`
  - `vote()`: line 67, line 69
  - `delegete()`: line 94, line 99, line 101
- INFO: Recommend to separate a private function for handling the conditions for `passCount` , `vetoCount`. In this case, the below function can be reused in both `vote()`, `delegate()`

```
function updateIsPass() private {
   if (passCount >= passThreshold) {
       isPass = true;
   } else if (vetoCount >= vetoThreshold) {
       isPass = false;
   }
}
```

- INFO: `judgeVoterThreshold()` recommend to use safe math.
- INFO: `CommonProposal()` Recommend to ensure sender is not a zero address, feature is a non-empty string.
- MINOR: Consider declaring functions as `external`, when there is no need for an internal access from contract itself. It could benefit for gas consumption. `public` modifier consumed 496 gas vs `external` consumed 261 gas only This is due to the fact that Solidity copies arguments to memory on a public function, while external read from calldata which a cheaper than memory allocation.
  - Vote.vote
  - Vote.delegate
  - Proposal.voterVote
  - Proposal.voterDelegate
  - Proposal.judgmentVote
  - Proposal.judgmentDelegate
  - Ballot.addJMNProposal
  - Ballot.addSMNProposal
  - Ballot.systemProposalCount
  - Ballot.commonProposalCount
  - Ballot.startSystemCallback
  - Ballot.stopSystemCallback
  - Ballot.updateSystemCallback
  - Ballot.start
  - Ballot.stop
  - Ballot.update
- INFO: Recommend implementing another private function for different conditions handling and code reusability in both `vote()`, `delegate()`

```
function updateIsPass() private {
   if (passCount >= passThreshold) {
       isPass = true;
   } else if (vetoCount >= vetoThreshold) {
       isPass = false;
   }
}
```

- MINOR: Potential lead to Re-entrancy vulnerability of following functions and consider applying following changes:
  - `voterVote()`

```
function voterVote(bool _decision) public OnlyVote {
       updateStageVote();
       vote.vote(msg.sender, _decision);
   }
```

- `voterDelegate()`

```
function voterDelegate(address _delegateAddress) public OnlyVote {
      updateStageVote();
      vote.delegate(msg.sender, _delegateAddress);
  }
```

- `judgementVote()`

```
function judgmentVote(bool _decision) public OnlyJudgment {
       updateStageJudgment();
       judgment.vote(msg.sender, _decision);
   }
```

- `judgementDelegate()`

```
function judgmentDelegate(address _delegateAddress) public OnlyJudgment {
      updateStageJudgment();
      judgment.delegate(msg.sender, _delegateAddress);
  }
```

- MINOR: `addSystemProposal()`,`addJMNProposal()`, `addSMNProposal` be more effective swap the mapping and push operation.

```
systemProposalMapping[address(systemProposal)] = systemProposals.length;
       systemProposals.push(systemProposal);
```

PermissionManageContract.sol

- MINOR: `constructor()` Consider adding zero address checking for `_owner`.
- INFO: Consider declare below functions to external for gas optimization:
  - PermissionManager.addJMN
  - PermissionManager.addSMN
  - PermissionManager.jmnCount
  - PermissionManager.smnCount
  - PermissionManager.deleteJMN
  - PermissionManager.deleteSMN
  - PermissionManager.addWorker
  - PermissionManager.deleteWorker
  - PermissionManager.workerSize
  - PermissionManager.getSMNAddress

SubchainManageContract.sol

- INFO `emptyEndorsement`: Consider to remove never used variable.
- MINOR: `addEndorsement()`: Possible contract may occur reentrancy vulnerabilities, consider to use check-effect-interact pattern to minimize the impact:

  - Below state variables changed, after the external call(s) to `ITemplateManager.getContractTemplate()`, `ITemplateManager.deployContract()`.

- MINOR `endorsementContractAbi`: Is it possible return an empty data
- MINOR `endorsementContractAddress`: Is it possible return a zero address
- INFO: Consider declare below functions to external for gas optimization:
  - SubchainManager.addSubchain
  - SubchainManager.addEndorsement
  - SubchainManager.getSubchain
  - SubchainManager.getEndorsement
  - SubchainManager.subchainSize
  - SubchainManager.endorsementSize
  - SubchainManager.addSMNAttention
  - SubchainManager.smnAttentionSize
  - SubchainManager.deleteSMNAttention

TemplateManageContract.sol

- INFO: `constructor()`: Consider checking the `_owner` is not a zero address.

```
constructor(address _owner) public {
      require(_owner != address(0), ``_owner is a zero address");
      owner = _owner;
      contractTemplates.push(emptyContractTemplate);
   }
```

- MAJOR: `deployContract()`: Consider adding code for ensuring the a new contract is created.

```
assembly {
          deployContractAddress := create(0, add(bytecodeWithAddress, 0x20),
              mload(bytecodeWithAddress))
          let codeSize := extcodesize(deployContractAddress)
          if eq(codeSize, 0) { revert(0, 0) }
      }
```

- MINOR: `addEndorsement()`: Possible contract may occur reentrancy vulnerabilities, consider to use check-effect-interact pattern to minimize the impact: Below state variables changed, after the external call(s) to [`ITemplateManager.getContractTemplate()` && `ITemplateManager.deployContract()`]: endorsements subchainEndorsementsMapping
  - `endorsementContractAbi`: Is it possible return an empty data
  - `endorsementContractAddress`: Is it possible return a zero address

- INFO: Consider declare below functions to external for gas optimization:
  - SubchainManager.addSubchain
  - SubchainManager.addEndorsement
  - SubchainManager.getSubchain
  - SubchainManager.getEndorsement
  - SubchainManager.subchainSize
  - SubchainManager.endorsementSize
  - SubchainManager.addSMNAttention
  - SubchainManager.smnAttentionSize
  - SubchainManager.deleteSMNAttention

Best practice

Smart contract development requires a particular engineering mindset. A failure in the initial construction can be catastrophic, and changing the project after the fact can be exceedingly difficult.

To ensure success and to avoid the challenges above smart contracts should here to best practices at their conception. Below, we summarized a checklist of key points that help to indicate a high overall quality of the current WaltonChain cross-chain project. (✓ indicates satisfaction; × indicates unsatisfaction; − indicates inapplicablility)

General

- ✓ Corrent environment settings, e.g. compiler version, test framework
- ✓ No compiler warnings
- ✓ Provide error message along with `assert`
- × Use events to monitor contract activities
- ✓ Correct time dependency

Solidity Specific

- − Safe external call
- − No reentrancy pattern
- ✓ Correct handling of integer overflow and underflow
- ✓ Correct visibility for state variables

Privilege Control

- ✓ Provide pause functionality for control and emergency handling
- ✓ Provide time buffer between certain operations
- ✓ Provide proper access control for functions
- ✓ Establish rate limit for certain operations
- ✓ Restrict access to sensitive functions
- ✓ Restrict permission to contract destruction

Documentation

- ✓ Provide project README and execution guidance
- ✓ Provide inline comment for function intention
- ✓ Provide instruction to initialize and execute the test files

Testing

- × Provide migration scripts
- × Provide test scripts and coverage for potential scenarios

Overall we found the smart contracts to follow good practices. With the final update of source code and delivery of the audit report, we conclude that the contract is structurally sound and not vulnerable to any classically known anti-patterns or security issues. The audit report itself is not necessarily a guarantee of correctness or trustworthiness, and we always recommend to seek multiple opinions, keep improving the codebase, and more test coverage and sandbox deployments before the mainnet release.