CONTRACT WOLF

Security Assessment

GETE Network

Verified on 6/21/25

SUMMARY

Project	CHAIN	METHODOLOGY
GETE Network	Gete	Manual & Automatic Analysis

FILES	DELIVERY	TYPE
Single	6/21/25	Standard Audit

5	0	1	0	2	2	0
Total Findings	Critical	Major	Medium	Minor	Informational	Resolved

🟥 0 Critical	An exposure that can affect the contract functions in several events that can risk and disrupt the contract
🟧 1 Major	An opening & exposure to manipulate the contract in an unwanted manner
🟧 0 Medium	An opening that could affect the outcome in executing the contract in a specific situation
⬜ 2 Minor	An opening but doesn't have an impact on the functionality of the contract
🟦 2 Informational	An opening that consists information but will not risk or affect the contract
🟩 0 Resolved	ContractWolf's findings has been acknowledged & resolved by the project

STATUS ✔️**AUDIT PASSED**

TABLE OF CONTENTS | GETE Network

DISCLAIMER | GETE Network

ContractWolf audits and reports should not be considered as a form of project's "Advertisement" and does not cover any interaction and assessment from "Project Contract" to "External Contracts" such as PancakeSwap, UniSwap, SushiSwap or similar.

ContractWolf does not provide any warranty on its released report and should not be used as a decision to invest into audited projects.

ContractWolf provides a transparent report to all its "Clients" and to its "Clients Participants" and will not claim any guarantee of bug-free code within its **SMART CONTRACT**.

ContractWolf's presence is to analyze, audit and assess the Client's Smart Contract to find any underlying risk and to eliminate any logic and flow errors within its code.

Each company or project should be liable to its security flaws and functionalities.

SCOPE OF WORK | GETE Network

GETE Network team has agreed and provided us with the files that need to be tested (*Github, BSCscan, Etherscan, Local files etc*). The scope of audit is the main contract.

The goal of this engagement is to identify if there is a possibility of security flaws in the implementation of smart contract and its systems.

ContractWolf will be focusing on contract issues and functionalities along with the project claims from smart contract to their website, whitepaper, repository which has been provided by **GETE Network**.

AUDITING APPROACH | GETE Network

Every line of code along with its functionalities will undergo manual review to check for security issues, quality of logic and contract scope of inheritance. The manual review will be done by our team that will document any issues that they discovered.

METHODOLOGY

The auditing process follows a routine series of steps :

1. Code review that includes the following :
   - Review of the specifications, sources and instructions provided to ContractWolf to make sure we understand the size, scope and functionality of the smart contract.
   - Manual review of code. Our team will have a process of reading the code line-by-line with the intention of identifying potential vulnerabilities, underlying and hidden security flaws.

2. Testing and automated analysis that includes :
   - Testing the smart contract function with common test cases and scenarios to ensure that it returns the expected results.

3. Best practices and ethical review. The team will review the contract with the aim to improve efficiency, effectiveness, clarifications, maintainability, security and control within the smart contract.

4. Recommendations to help the project take steps to eliminate or minimize threats and secure the smart contract.

TOKEN DETAILS | GETE Network

GETE Network is the world's first cross-chain gaming public blockchain that supports the bitcoin chain.

Token Name	Symbol	Decimal	Total Supply	Chain
GETE	GETE	-	-	Gete

SOURCE

Source	*Gete CORE Network / Chain*

FINDINGS | GETE Network

5	0	1	0	2	2	0
Total Findings	Critical	Major	Medium	Minor	Informational	Resolved

This report has been prepared to state the issues and vulnerabilities for GETE Network through this audit. The goal of this report findings is to identify specifically and fix any underlying issues and errors

Title	File & Line #	Severity	Status
Fatal Crash on Index-Batch	L: 2261	Major	• Pending
Crash vs. Error Return on Write Failure	L: 2261	Minor	• Pending
Inconsistent Persistence Ordering	L: 1529	Minor	• Pending
Unbounded Loops in procFutureBlocks	L: 1070	Informational	• Pending
Resource Exhaustion (Memory & CPU)	L: 224	Informational	• Pending

FIXES & RECOMMENDATION

Resource Exhaustion (Memory & CPU)

In `NewBlockChain` you have hardcoded LRU sizes:

```
bodyCache, _    := lru.New(bodyCacheLimit)
futureBlocks, _ := lru.New(maxFutureBlocks)
```

Recommendation

Make these limits configurable and add back-pressure if they're exceeded.

```
    // MITIGATION: read from cacheConfig so operators can tune
    bodyCache, _    := lru.New(bc.cacheConfig.BodyCacheLimit)
    futureBlocks, _ := lru.New(bc.cacheConfig.FutureBlocksLimit)
```

Unbounded Loops in procFutureBlocks

If futureBlocks keeps growing faster than procFutureBlocks can drain, this goroutine may run forever, starving other work (CPU starvation) or holding references to large arrays (memory leak).

```
func (bc *BlockChain) procFutureBlocks() {
blocks := make([]*types.Block, 0, bc.futureBlocks.Len())
  for _, hash := range bc.futureBlocks.Keys() {
    //
  }
  for i := range blocks {
    bc.InsertChain(blocks[i : i+1])
  }
}
```

Recommendation

Add a per-iteration limit and schedule the next batch via timer.

```
func (bc *BlockChain) procFutureBlocks() {
 // MITIGATION: process at most N per run to avoid long bursts
 const maxProcess = 100
 blocks := make([]*types.Block, 0, maxProcess)
  for _, hash := range bc.futureBlocks.Keys() {
   if len(blocks) >= maxProcess {
     break
   }
    if block, exist := bc.futureBlocks.Peek(hash); exist {
      blocks = append(blocks, block.(*types.Block))
    }
  }
  if len(blocks) == 0 {
    return
  }
  // insertion loop unchanged
 // Schedule the next batch after a short delay to yield CPU
 go func() {
   time.Sleep(50 * time.Millisecond)
   bc.procFutureBlocks()
 }()
}
```

Inconsistent Persistence Ordering (Block vs. State)

If `blockBatch.Write()` succeeds but `state.Commit()` fails, you leave the block in the DB without its corresponding state trie, corrupting chain consistency.

```
    // ... before
    if err := blockBatch.Write(); err != nil {
        log.Crit("Failed to write block into disk", "err", err)
    }
    // Commit all cached state changes into underlying memory database.
    root, err := state.Commit(...)
```

Recommendation

Defer the on-disk block write until after the state has been successfully committed.

```
func (bc *BlockChain) writeBlockWithState(...) (status WriteStatus, err error) {
    // First, commit all cached state changes to memory DB.
    root, err := state.Commit(bc.chainConfig.IsEIP158(block.Number()))
    if err != nil {
        return NonStatTy, fmt.Errorf("state commit failed: %w", err)
    }
    // Now write block metadata atomically to disk
    if err := blockBatch.Write(); err != nil {
        return NonStatTy, fmt.Errorf("failed to write block to disk: %w", err)
    }
    // Proceed with trie commits...
```

Crash vs. Error Return on Write Failure

log.Crit forcibly exits the process, giving no chance for graceful shutdown or retry.

```
    if err := blockBatch.Write(); err != nil {
        log.Crit("Failed to write block into disk", "err", err)
    }
```

Recommendation

Replace with an error return so the caller can handle it.

```
func (bc *BlockChain) writeBlockWithState(...) (status WriteStatus, err error) {
    if err := blockBatch.Write(); err != nil {
        return NonStatTy, fmt.Errorf("disk write failed: %w", err)
    }
```

Fatal Crash on Index-Batch Write Failure in reorg

A corrupt index batch causes log.Crit, which aborts the entire node immediately, with no chance to recover or rollback.

```
    if err := blockBatch.Write(); err != nil {
        log.Crit("Failed to write block into disk", "err", err)
    }
```

Recommendation

Return the error instead, allowing the caller to handle it gracefully (retry, alert or shutdown).

```
func (bc *BlockChain) reorg(oldBlock, newBlock *types.Block) error {
    if err := indexesBatch.Write(); err != nil {
        return fmt.Errorf("failed to write reorg index cleanup: %w", err)
    }
```

AUDIT COMMENTS | GETE Network

Smart Contract audit comment for a non-technical perspective

🔴 Audited : gete公链源码/core/blockchain.go

CONTRACTWOLF

Blockchain Security - Smart Contract Audits