BYBIT

Blockchain Freezing Exposed

Examine The Impact of Fund Freezing Ability in Blockchain

Content

Brief

Fund freezing, in layman's terms, occurs when a foundation can lock a user's assets without their consent. This capability runs counter to the core principle of decentralization by reintroducing a central authority with control over funds — similar to a traditional bank. Therefore, when the Sui Foundation intervened to freeze assets stolen from the Cetus protocol, it sparked a heated debate within the blockchain community.

This incident prompted Bybit's Lazarus Security Lab team to launch a comprehensive investigation into whether other blockchain networks have similar powers. The team analyzed 166 different blockchains. Since most networks did not disclose such functionality in their public documentation, the team conducted in-depth code reviews of blockchain repositories. Given the vast amount of code involved, they developed a systematic, AI-assisted approach to conduct the research efficiently.

The goal of this research is to bring greater transparency to how these mechanisms operate while laying the groundwork for future studies and risk assessments in the fast-evolving digital asset and blockchain landscape.

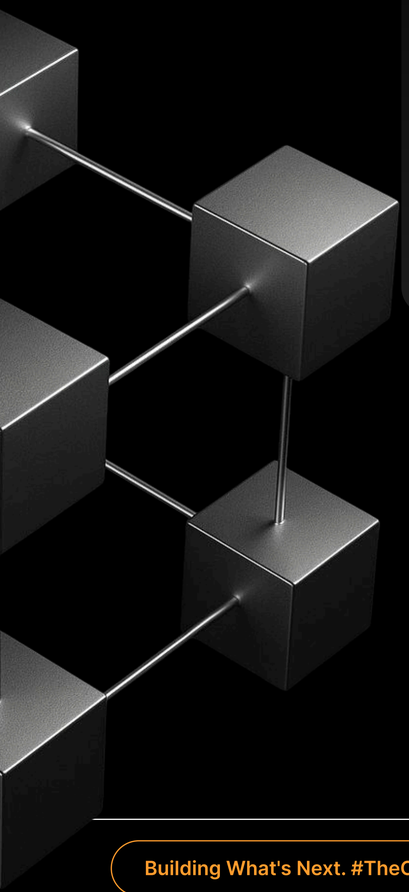
2025

Key Highlights

2025

BYBIT

Key Highlights



01.

The Lazarus Security Lab reviewed 166 prominent blockchains to assess whether they have the capability to freeze funds.

02.

The research adopted a novel approach: using a customized AI agent to filter blockchains, followed by deep manual analysis.

03.

The research confirmed that 16 blockchains currently have freezing capabilities, while an additional 19 blockchains could potentially support freezing in the future.

04

Among the 16 chains, the team identified three main freezing mechanisms:

- Hardcoded freezing
- Config file–based freezing
- On-chain smart contract freezing

05.

During the research, the team uncovered five noteworthy incidents and unique implementations:

- Cetus hack on Sui and APTOS's prompt update
- BNB hack incident followed by the blacklisting ability
- Cosmos's modular accounts with address-blocking ability
- HECO chain's blacklisting through the smart contract
- VeChain hack incident followed by the blacklisting ability

Key Statistics

	Hardcoded	Configfile	Smart Contract	Total
Freezing Ability	5	10	1	16
	CHILIZ, VIC, XDC, BNB, VECHAIN	ONE, HVH, APTOS, SUPRA, EOS, ROSE, WAXP, SUI, LINEA, WAVES	HECO	
Potential Freezing Ability	19	-	-	19
	ARBI, ATOM, AXL, BABYLON, CELESTIA, DYDX, DYM, DYMEVM, EVMOS, INITIA, KAVA/KAVAEVM, LUNA, MANTRA, Nillion, OKB, RUNE, SEI/ SEIEVM, SRCT, XION	-	-	

For more in-depth insights, read the full report.

Investigation Findings

2025

1.1 Freezing Methods

During its research, Bybit's Lazarus Security Lab identified 16 chains with protocol-level freezing capabilities. This means a blockchain's foundation or governance group can completely block specific addresses of their choosing. Once an address is blacklisted, any tokens within it become inaccessible to the original signer, and no one can access that address until it is removed from the blacklist by the foundation or governance group.

Among the 16 chains, the team found three distinct methods for freezing funds at the protocol level.

1.1.1 Hardcoded freezing method (public blacklist)

1.1.2 Config file-based freezing method (private blacklist)

1.1.3 On-chain smart contract freezing method

1.1.1 Hardcoded Freezing Method (Public Blacklist)

The hardcoded freezing method was first employed by VeChain in Dec 2019. Following a hack that stole approximately $6.6 million worth of VET tokens from its official buyback wallet, the VeChain Foundation introduced a function that blocked blacklisted addresses from signing on-chain transactions. A total of 469 addresses associated with the attackers were added to a blacklist on GitHub, effectively preventing them from interacting with the VeChain blockchain and liquidating the stolen funds.

In Oct 2022, the BNB Chain also used the hardcoded freezing method after a major security breach on its cross-chain bridge caused by a vulnerability in IAVL tree proof verification. The exploit allowed an attacker to forge withdrawal proofs and mint 2 million BNB tokens, worth about $570 million at the time. Through hardcoded blacklisting, the impact was partially contained, with only around $100–110 million successfully moved off-chain.

As the two cases illustrate, the primary advantage of a freezing ability is the swift remediation of financial damage to an ecosystem. This is achieved by preventing attackers from moving or liquidating stolen assets by validators and foundation team.

1.1.1 Hardcoded Freezing Method (Public Blacklist)

In total, five blockchains, including BNB Chain and VeChain, were found to have hardcoded freezing capabilities.

Chain	GitHub Link
CHILIZ	https://github.com/chiliz-chain/v2/blob/3e0e8a8fd5313d83c288edae3cc79cb4a6a1abcc/core/types/blacklist.go#L4
VIC	https://github.com/BuildOnViction/victionchain/blob/master/common/constants.go
XDC	https://github.com/XinFinOrg/XDPoSChain/blob/af4178b2c7f9d668d8ba1f3a0244606a20ce303d/common/constants.mainnet.go#L4
BNB Chain	https://github.com/bnb-chain/bsc/blob/e7b198c10cc8c3e32a5b6d98cd34938115f08ade/core/types/blacklist.go#L6
VECHAIN	https://github.com/vechain/thor/blob/b01ac99428e18130f57a8b4b19e005bfc6921184/thor/blocklist.go#L493

1.1.2 Config File-based Freezing Method
(Private Blacklist)

The logic behind blocking certain addresses is the same as with the hardcoded method. The difference is that the blacklist is managed and updated in local configuration files — such as YAML, ENV or TOML — that are accessible only to validators, the foundation and core developers.

On May 22, 2025, Cetus, a decentralized exchange (DEX) on Sui, was hacked, resulting in losses of roughly $223 million in digital assets. In response, Sui's validators and the Sui Foundation exercised their fund-freezing capability. They added the attackers' addresses to their configuration files and restarted the nodes, blocking those addresses from signing transactions on-chain.

Shortly after, Aptos, often referred to as Sui's "brother chain", updated its code to include a blacklisting function of its own. These actions have fueled debate over whether such foundational blockchain projects are truly decentralized or if they are essentially centralized entities operating on a distributed ledger. Further details on Aptos's response can be found in Section 1.2.1.

1.1.2 Config File-based Freezing Method (Private Blacklist)

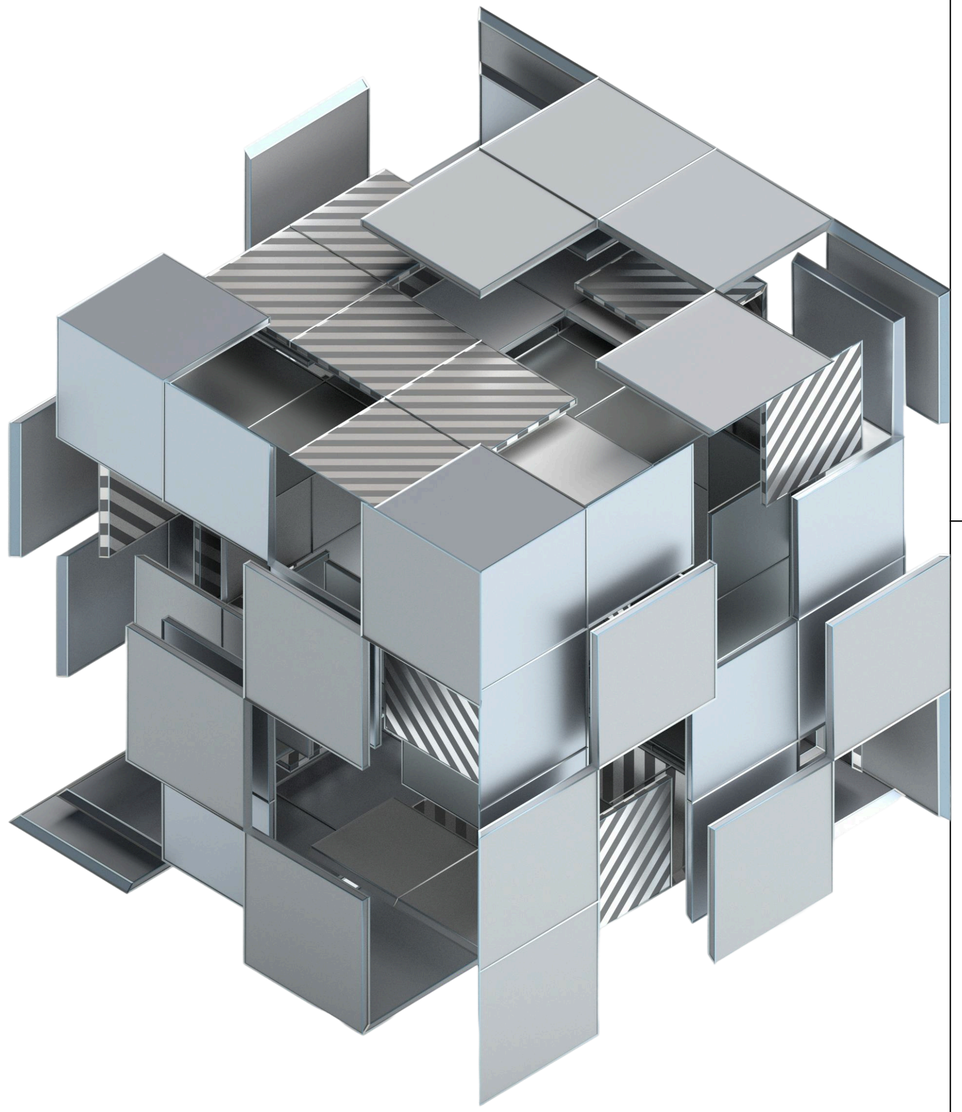
A total of ten chains, including Sui, were found to have this configuration file–based freezing capability.

Chain	GitHub Link
ONE	https://github.com/harmony-one/docs-home/blob/23d060ca3181818aeff7e9df5996e917e3d163f0/network/validators/node-setup/installing-updating/installing-node/using-binary.md?plain=1#L183
VIC	https://github.com/havah-project/goloop-havah/blob/ddeb0cf687a04ff8740267310d763fe1a39a3534/service/transaction/transactionhandler.go#L117
APTOS	https://github.com/aptos-labs/aptos-core/blob/b36e1c490b72e8cf352aa295a48636d439488760/crates/aptos-transaction-filters/src/block_transaction_filter.rs#L93
SUPRA	https://github.com/Entropy-Foundation/aptos-core/blob/ffabf2ab759dfa7e0f37033e8aa617feb54750d3/config/src/config/transaction_filter_type.rs#L156
EOS	https://github.com/AntelopeIO/spring/blob/83d6d3d6829ccb799426065844e7c7ad94da7e12/libraries/chain/controller.cpp#L4797-L4882

Chain	GitHub Link
ROSE	https://github.com/oasisprotocol/oasis-core/blob/2a01cbe4668099eaf028e73e21fe35db2c94b879/go/consensus/cometbft/full/common.go#L129
WAXP	https://github.com/worldwide-asset-exchange/wax-blockchain/blob/8e8e0a24784abfb8dbf874788c7ddd75832bb90d/plugins/producer_plugin/producer_plugin.cpp#L1529
SUI	https://github.com/MystenLabs/sui/blob/93a61faf954249122a97f404812101362b6a86e2/crates/sui-config/src/transaction_deny_config.rs#L104
LINEA	https://github.com/Consensys/linea-monorepo/blob/19817c877d962756a380381950f33be8634c7c1f/besu-plugins/linea-sequencer/sequencer/src/main/java/net/consensys/linea/sequencer/txpoolvalidation/validators/AllowedAddressValidator.java#L66
WAVES	https://github.com/wavesplatform/Waves/blob/769f6d70e127af020cc3c4dc83b9130724cd1a13/node/tests/src/test/scala/com/wavesplatform/utx/UtxPoolSpecification.scala#L33

1.1.3 On-Chain Smart Contract Freezing Method

Managing a blacklist through an on-chain smart contract is a unique approach used exclusively by the HECO chain, also known as the Huobi Eco Chain. To enable prompt blocking and avoid the need to restart nodes to make the blacklist effective, HECO manages its blacklist via the on-chain smart contract. Validators check the list by querying the smart contract's ABI. More details are available in Section 1.2.4.



1.2 Notable Research Findings

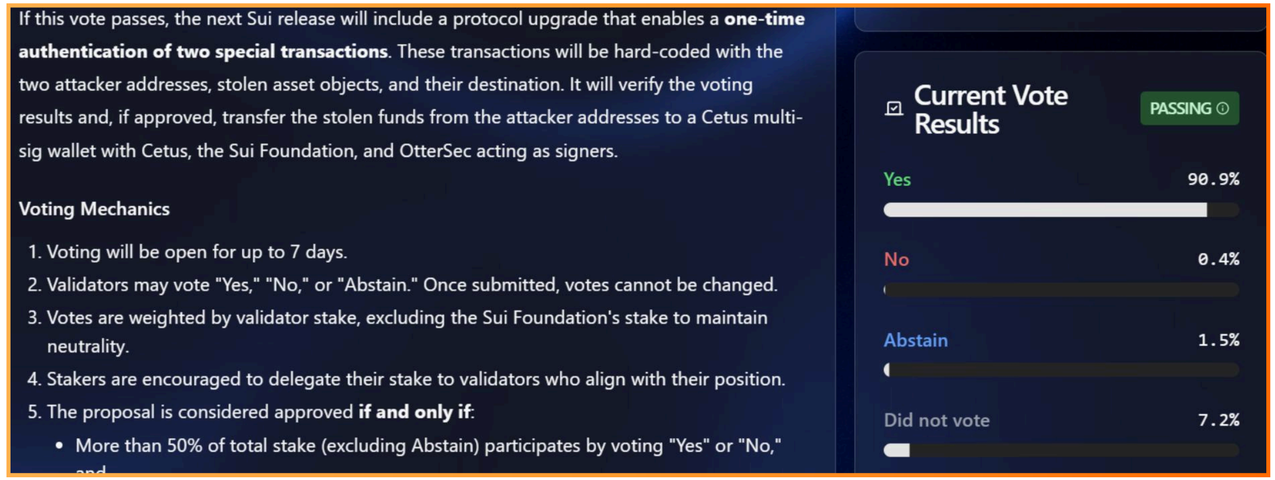
1.2.1 Cetus hack on Sui and APTOS's prompt update

Cetus Hack

Cetus, a decentralized exchange (DEX) on Sui, was exploited on May 22, 2025, resulting in losses of about $223 million in digital assets. The attacker exploited a vulnerability in Cetus's math library, which allowed them to manipulate the prices of liquidity pools using fake tokens.

Given the potential damage to Sui's ecosystem, the Sui Foundation and validators exercised their protocol-level fund-freezing ability, successfully freezing $162 million of the stolen funds.

After freezing,the SUI team performed an additional action: recover the funds from hacker's address. Sui community successfully passed a governance vote with 90.9% approval from validators to recover $162 million in assets frozen after a hack that affected the Cetus protocol. The vote's approval allows for frozen funds, which were in a hacker's address, to be transferred to a specific Cetus multisignature wallet. From there, the funds will be held in trust and eventually returned to the affected users.



1.2.1 Cetus hack on Sui and APTOS's prompt update

APTOS update after the Cetus hack

Aptos and Sui inherited the Move programming language, designed as a secure and universal language for smart contracts, emphasizing core principles such as resource ownership and scarcity.

However, our recent research highlights a key difference in protocol-level fund freezing. While Sui had this capability in place since Apr 2023, Aptos did not implement it until after Sui exercised the feature in response to the Cetus hack.

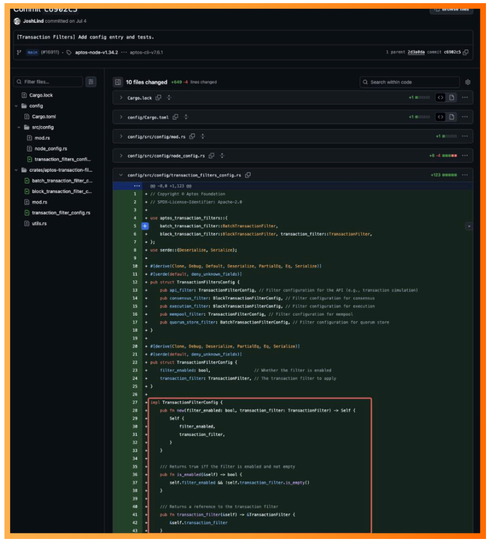
This development only became known to the public when Sui used it to block the hacker's fund transfers.

Our findings show that Aptos introduced support for TransactionFilter on Jul 4, roughly one month after the Cetus hack on May 22.

The update introduces functionality similar to that of the Sui blockchain, allowing transactions to be denied based on blacklisted addresses. The blacklist can be updated through YAML or TOML files, but implementing changes requires a node restart.

Conclusion

The cornerstone of blockchain is its ability to process transactions in decentralized way. The function used in SUI, and those recently updated in Aptos, demonstrates its resilience to hacking and prompt risk management methods to recover their lost funds. However, it also shows centralization power to the blockchain community.



https://github.com/aptos-labs/aptos-core/blob/fb06376c4461795699cb7d49605b54ac494a5972/config/src/config/transaction_filters_config.rs#L22

1.2.2 BNB hack incident followed by the blacklisting ability

Background

On Oct 6, 2022, the BNB Chain suffered a major security breach on its cross-chain bridge caused by a vulnerability in the IAVL tree proof verification process. This flaw allowed a hacker to forge withdrawal proofs and mint 2 million BNB tokens, resulting in losses of around $570 million at the time.

The IAVL tree's proof verification process is designed to validate cross-chain transactions between the BNB Chain and the BSC Token Hub. The vulnerability arose because the verification process skipped checks on the right child. Exploiting this gap, the attacker created a malicious proof that inserted a new leaf into the tree, bypassing verification and minting BSC tokens out of thin air.

In response, the BNB Chain core team, working with validators, quickly paused the blockchain to prevent further asset drain. They also rolled out a software update that hardcoded the attacker's wallet address into a blacklist at the protocol level, effectively freezing the stolen funds still on the chain. Like Sui, the BNB Chain also introduced protocol-level fund-freezing capabilities after the hack.

1.2.2 BNB hack incident followed by the blacklisting ability

BNB

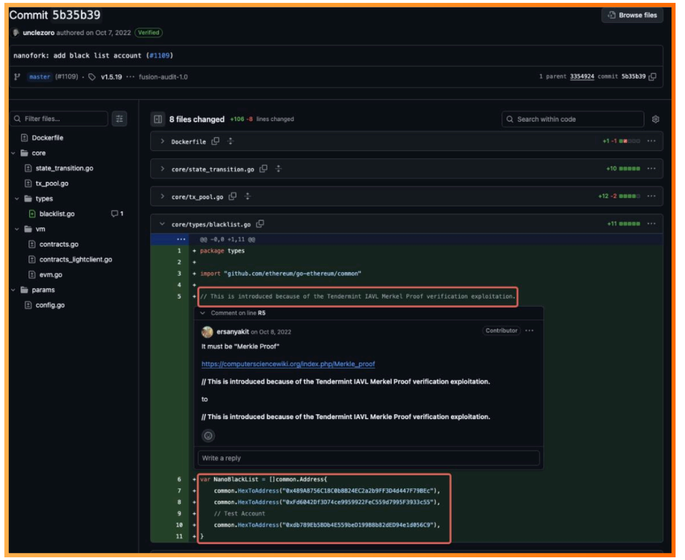
Key Points

Like Sui, the BNB Chain also introduced a protocol-level freezing ability. However, there is one key difference: BNB Chain maintains a public list of blacklisted addresses.

In Sui, blacklisted addresses are managed in the local configuration files of validators (and potentially the Sui Foundation), whereas on BNB Chain, the full blacklist is visible to everyone.

Conclusion

As seen with both Sui and BNB Chain, blockchains are ultimately created and managed by business organizations (unlike Bitcoin). When a severe hack occurs, these organizations can — depending on the scale of financial damage — quickly introduce protocol-level address blacklisting to minimize losses.



https://github.com/bnb-chain/bsc/blob/e7b198c10cc8c3e32a5b6d98cd34938115f08ade/core/types/blacklist.go#L6

2025

1.2.3 Cosmos's modular accounts with address-blocking ability

Cosmos uses special module accounts, which are accounts created by Cosmos SDK modules to manage funds and carry out module-specific operations. Unlike regular user accounts, these accounts are controlled by module logic rather than private keys.

These module accounts are typically blocked from performing normal transactions, such as sending native tokens through standard user-initiated messages. This is because they serve specific internal protocol functions rather than general user activities.

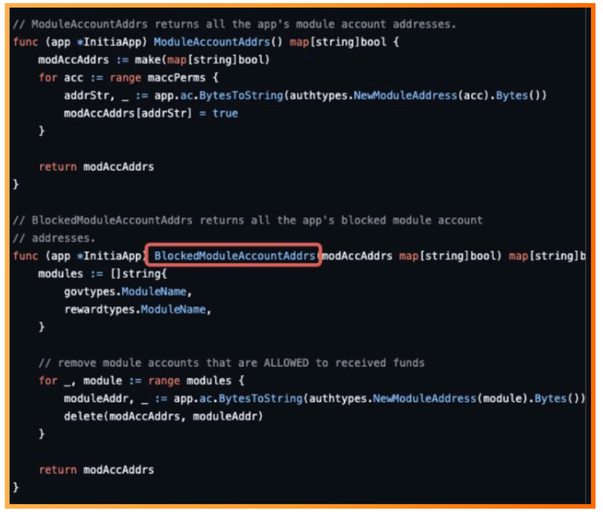
Each module maintains a list called "blockedAddrs". By default, this list includes all module accounts (those used by on-chain modules for staking, governance, distribution, etc.). The purpose is twofold: to prevent users from accidentally sending tokens directly to these accounts and to stop module accounts from transferring funds out in the event of a hack.

1.2.3 Cosmos's modular accounts with address-blocking ability

COSMOS

This function could, in theory, be modified in the future to add a hacker's address, but so far none of the blockchains in the Cosmos ecosystem have used it in this way. Moreover, implementing such a change would require a hard fork along with minor adjustments — likely in the anteHandler file — or additional code modifications.

Such function can later be modified to add hacker's address in future, but currently none of blockchains in Cosmos chain as used the function in such way. Moreover, adding such address requires hardfork and additional minor change (probably under file anteHandler) or addition of code is required.



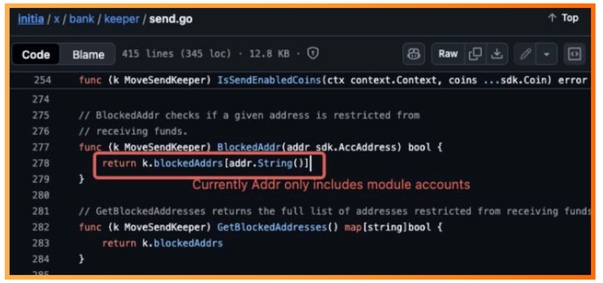
```go
// ModuleAccountAddrs returns all the app's module account addresses.
func (app *InitiaApp) ModuleAccountAddrs() map[string]bool {
    modAccAddrs := make(map[string]bool)
    for acc := range maccPerms {
        addrStr, _ := app.ac.BytesToString(authtypes.NewModuleAddress(acc).Bytes())
        modAccAddrs[addrStr] = true
    }

    return modAccAddrs
}

// BlockedModuleAccountAddrs returns all the app's blocked module account
// addresses.
func (app *InitiaApp) BlockedModuleAccountAddrs(modAccAddrs map[string]bool) map[string]b
    modules := []string{
        govtypes.ModuleName,
        rewardtypes.ModuleName,
    }

    // remove module accounts that are ALLOWED to received funds
    for _, module := range modules {
        moduleAddr, _ := app.ac.BytesToString(authtypes.NewModuleAddress(module).Bytes())
        delete(modAccAddrs, moduleAddr)
    }

    return modAccAddrs
}
```



initia / x / bank / keeper / send.go ↑ Top

Code Blame 415 lines (345 loc) · 12.8 KB · Raw

```go
254  func (k MoveSendKeeper) IsSendEnabledCoins(ctx context.Context, coins ...sdk.Coin) error
274
275  // BlockedAddr checks if a given address is restricted from
276  // receiving funds.
277  func (k MoveSendKeeper) BlockedAddr(addr sdk.AccAddress) bool {
278      return k.blockedAddrs[addr.String()]
279  }
280
281  // GetBlockedAddresses returns the full list of addresses restricted from receiving funds
282  func (k MoveSendKeeper) GetBlockedAddresses() map[string]bool {
283      return k.blockedAddrs
284  }
```

Currently Addr only includes module accounts

https://github.com/initia-labs/initia/blob/7947f7cf73cfdd7973fe4f69140c24043264444b/x/bank/keeper/send.go

2025

1.2.4 HECO chain's blacklisting through the smart contract

Unlike other blockchains with protocol-level freezing capabilities, HECO (also known as the Huobi ECO Chain) takes a unique approach to blacklisting. Instead of requiring a hard fork, HECO allows an admin address to add any address to the blacklist directly, with the update taking effect immediately.

How the blacklist is retrieved

I. **Cache check:**
The system first attempts to retrieve the blacklist from an in-memory LRU cache (c.blacklists), keyed by the parent block hash.

II. **Fast path for recent updates:**
If the last update occurred a long time ago and the cache doesn't contain the list, the system recursively checks the blacklist of the parent block (a fast path for blocks with no recent changes).

III. **Contract Call**
If the blacklist isn't found in the cache, the system queries the system contract directly on the blockchain state using the contract ABI.

IV. **Go**

```
alABI := c.abi[systemcontract.AddressListContractName]
get := func(method string) ([]common.Address, error) {
  ret, err := c.commonCallContract(header, parentState, alABI,
systemcontract.AddressListContractAddr, method, 1)
  ...
  blacks, ok := ret[0].([]common.Address)
  ...
  return blacks, nil
}
froms, err := get("getBlacksFrom")
tos, err := get("getBlacksTo")
```

- It calls two methods on the AddressListContract: getBlacksFrom and getBlacksTo.
- These return lists of addresses are blacklisted either as senders or receivers.

I. **Mapping Construction:**
It constructs a map of addresses to blacklistDirection (from, to or both) based on the results of the contract calls.

II. **Caching:**
The results are then stored in the LRU cache to enable faster retrieval in future lookups.

1.2.4 HECO chain's blacklisting through the smart contract

How the blacklist is updated

The blacklist is managed by the on-chain AddressListContract at address 0×0000000000000000000000000000000000000F004. This system contract includes built-in methods for managing blacklisted addresses:

- The contract provides functions such as getBlacksFrom and getBlacksTo to read the blacklist.
- To add or remove an address, you interact with the contract via a transaction that calls its management (admin) functions (e.g., addBlackFrom(address) or addBlackTo(address)).
- Only the contract admin — typically a multisig or governance account — has permission to update the blacklist.
- Nodes automatically read updates from the contract, so no restart or fork is required.

Conclusion

HECO has a design which allows the team to promptly add any address to blacklist. As no hard fork is required to take an effect, adding and taking effect all depends on how fast multi signature is processed.

1.2.5 VeChain hack incident followed by the blacklisting ability

On Dec 13, 2019, the VeChain Foundation suffered a hack in which roughly $6.6 million worth of VET tokens were stolen from its official buyback wallet. In response, the team introduced a blacklisting mechanism targeting the attacker's addresses.

A total of 469 addresses associated with the hack were added to a blacklist on GitHub, effectively preventing them from interacting with the VeChain blockchain and liquidating the stolen funds.

https://github.com/vechain/thor/blob/b01ac99428e18130f57a8b4b19e005bfc6921184/thor/blocklist.go#L493

Patterns Across Chain Groups

2025

2. Patterns Across Chain Groups

In our manual review, we found that the core logic for protocol-level freezing is usually located under tx_pool or validators code.

Furthermore, blockchains within the same family (e.g., EVM, Cosmos, UTXO) tend to show similar characteristics, influenced by their overall design and the programming languages used in their development.

Chain Family	Dominant Programming Language	Blacklist Management	Freezing Logic
EVM	Go	Most commonly uses a hardcoded blacklist, visible to the public. Notable exceptions include HECO, which manages its blacklist via a smart contract, and HVH, which uses a config file.	Core logic is typically embedded in the tx_pool or validators code. The approach is heavily influenced by BNB's method.
Object-Based	Rust	Managed in a local file by validators or blockchain organizations; not visible to the public.	Core logic resides in a dedicated Rust file (often named "transaction deny config") and is executed as part of the node code.
Cosmos	Go	If present, blacklists are generally hardcoded and visible to the public.	Currently, Cosmos chains lack a fully functional freezing mechanism at the protocol level, though one could be introduced by modifying code used for modular addresses.
Other	Varies	Methods vary depending on the chain, ranging from hardcoded lists to config files.	The implementation differs across consensus mechanisms and architectures, but the underlying fund-blocking logic is usually found in the tx_pool or validators code.

Methodology

2025

3.1 Brief

When we discovered that some blockchains include built-in address-freezing mechanisms, we faced a significant challenge: analyzing 166 Bybit-supported networks to assess their censorship capabilities. The sheer volume of code made manual review impractical, so we developed a systematic, AI-assisted detection pipeline. Our workflow unfolded in three key phases:

1. Test Data Preparation

We selected the Sui blockchain as our initial test case after identifying, through business operations, that it included address-freezing mechanisms.

This real-world example served as the ideal validation dataset for our AI detection methodology.

2. Experimental Prompt Development

Using Sui's codebase, we ran extensive tests with multiple prompt variations to determine the most effective detection approach.

Through iterative experimentation, we compared prompt performance in accurately identifying freezing mechanisms and ultimately selected the version that achieved the highest accuracy based on observed outputs.

3. Large-Scale Application

After validating our approach on Sui, we deployed the optimized detection system (prompt + AI model: Claude-4.1 Opus) across all 166 blockchain networks, systematically identifying which chains included similar freezing capabilities.

3.2 Our AI Detection Methodology

Our prompt development process began with an expert analysis of Sui's freezing mechanisms, addressing a key challenge: accurately detecting freezing capabilities across 166 blockchains, each with its own implementation, would be prohibitively time-consuming if done manually. To overcome this, we focused on identifying the fundamental characteristics that define freezing capabilities, ensuring our methodology could capture diverse implementations. This analysis revealed three critical characteristics that determine a blockchain's ability to freeze transactions:

1. Initial Feature Extraction

Through a manual expert review of Sui's codebase, we identified the essential components:

- Address blacklist configuration: mechanisms such as deny_list or address_blacklist.
- Real-time configuration loading: Allowing key transaction processors to read configurations during runtime.
- Transaction filtering during processing: Rejecting blacklisted transactions at various points in the transaction lifecycle.

2. Iterative Prompt Testing

We tested multiple prompt variations against Sui's codebase, refining our approach based on accuracy. Each iteration helped us better define the detection criteria, moving from broad searches to precise capability identification applicable across different blockchain architectures.

3. Final Prompt Optimization

After extensive refinements validated against Sui's known freezing capabilities, we finalized a prompt that accurately detects both Layer 1 native token restrictions and Layer 2 project token controls. The prompt instructs the AI to analyze blacklist modules, configuration-loading mechanisms and transaction-filtering logic, while distinguishing between native and project token freezing, ensuring comprehensive detection across diverse blockchain architectures.

3.2 Our AI Detection Methodology

This systematic approach turned what would have been an overwhelming manual review into an efficient, AI-powered analysis, allowing us to thoroughly assess censorship risks across the blockchain ecosystem.

```
Please play the role of a blockchain code analysis expert and review all the code of this project. The goal is to determine whether this blockchain has the capability to **"quickly freeze key token transactions"**

Background definition is as follows:

## Main Analysis Objectives

We are concerned with **whether this chain can freeze the transfer or payment transactions of key tokens**. For different types of blockchain projects, the definition of key tokens is as follows:

- **Layer 1 Main Chain:** Only focus on "native tokens of this chain"
- **Layer 2 Project Chain:** Focus on both "native tokens of this chain" and "project tokens"

Note: We do not care about other tokens (such as ERC20, third-party tokens, etc.). Based on the Layer, we only focus on native coins or only focus on native coins and project coins themselves.

If a project meets the following structural characteristics, it can be determined to have **freezing capability:**

- Address blacklist configuration support: There exists some configuration mechanism (such as deny_list / address_blacklist) that allows setting restricted addresses;

- Configuration quick loading capability: Key transaction processing modules (such as sequencer, executor, sorter, consensus process, etc.) support reading this configuration at runtime or after restart;

- Transaction filtering mechanism exists: In the transaction lifecycle (such as packaging, signature verification, sorting, execution, etc. stages), it can identify and reject transactions from specified addresses;

- The target must be transaction types of key tokens:

  - Layer 1: Core behaviors of native tokens (such as SUI, APT, ETH, SEI, etc.) like transfers, payments, fuel payments, etc.

  - Layer 2: Transfer, payment, etc. behaviors of native tokens + project tokens

Please answer the following questions based on the above criteria:

## Freezing Capability Analysis

1. Does this project have blacklist, address restriction, dynamic configuration loading and other modules? If yes, please list related functions, modules, file paths.

2. Does there exist logic for address-level filtering of core transaction types for key tokens?
 a. Native tokens: Will transfers, payments, gas consumption, etc. be rejected for execution, rejected for packaging, or actively discarded by nodes
 b. Project tokens: (Layer 2 only) Are transfers, payments, etc. of project tokens similarly restricted

3. Does it support quickly controlling whether to reject these key token transactions through configuration? Do hot-loading mechanisms for configuration or loading mechanisms after restart exist?

4. At what stage does the transaction filtering logic take effect? For example, transaction signature verification, pre-packaging checks, rejection during transaction execution?

5. For Layer 2 projects, please analyze whether the freezing capabilities of project tokens and native tokens are consistent?
```

Example of the prompt we use.

3.3 AI Analysis Limitations and Practical Considerations

While AI greatly improved our efficiency in analyzing 166 blockchain projects for freezing capabilities and security vulnerabilities, we also encountered several practical challenges worth sharing with the research community.

These issues highlight methodological pitfalls that researchers should keep in mind when conducting large-scale blockchain codebase analysis.

3.3.1 Role-based Freezing Authority Misidentification

Background

We used an optimized prompt together with Claude-4.1-Opus to analyze blockchain core code repositories. During this automated scanning phase, the AI appeared to interpret freezing capabilities without issue. However, discrepancies emerged during manual review, where we found that the AI's understanding of "freezing ability" did not fully align with the criteria our analysts required.

Technically speaking, the AI's identification of freezing capabilities was not incorrect, but it exposed a limitation in how the prompt had been designed — specifically around the scope and authority level of freezing mechanisms. This underscores why manual review remains an essential part of our analysis pipeline.

Observation

For example, in the case of Symbol Chain, the AI flagged potential freezing capabilities and returned the following response to our questions:

```
Does this project have blacklist, address restriction, dynamic
configuration loading and other modules?
Model Replied:Yes

Does there exist logic for address-level filtering of core
transaction types for key tokens?
Model Replied:Yes, through AccountAddressRestrictionTransaction and
XYM support mechanisms (XYM is the native token of Symbol Chain)

Does it support quickly controlling whether to reject these key token
transactions through configuration? Do hot-loading mechanisms for
configuration or loading mechanisms after restart exist?
Model Replied:No hot update mechanism found. Restrictions are set
dynamically through AccountAddressRestrictionTransaction rather than
static configuration files. Auto-recovers after node restart

At what stage does the transaction filtering logic take effect? For
example, transaction signature verification, pre-packaging checks,
rejection during transaction execution?
Model Replied:During transaction signature verification

For Layer 2 projects, please analyze whether the freezing
capabilities of project tokens and native tokens are consistent?
Model Replied:No, it is layer 1

Conclusion: Yes - Symbol has the capability to rapidly freeze
critical token transactions.
```

Upon manual review, we found that this was not a system-wide freezing function but rather a personal restriction feature, allowing individual users to block specific addresses from sending them tokens. In essence, it operates more like a "block" function on social media platforms.

3.3.1 Role-based Freezing Authority Misidentification

Analysis

This mechanism is fundamentally different from the network-wide administrative freezing powers that were the true focus of our investigation. While the AI correctly detected address-filtering logic in the codebase, it failed to distinguish between user-level blocking and administrator-level freezing. At the code level, both features share similar implementation patterns — address validation, transaction filtering and conditional execution — but their scope and impact differ significantly. User-level blocking affects only the wallets of individuals, whereas administrative freezing has the power to stop token transfers across the entire network.

Solution

This is exactly why we treat manual review as a critical second layer of verification. No single test dataset can capture every variation in how freezing might be implemented across different chains. What matters is that all chains with true freezing capabilities share certain fundamental traits. Analysts use test data to confirm that the AI understands the essential nature of freezing capabilities, but edge cases — such as differences in authority levels — will inevitably appear. This particular case provides a valuable reference point. By incorporating authority-level distinctions into our prompts, we can refine detection and apply them as filtering criteria during manual verification.

3.3.2 Lacks Comprehensive Code Execution Path Analysis

Background

We applied the same optimized prompt with Claude-4.1-Opus but broadened the detection scope to include IAVL library vulnerabilities. This was necessary after discovering that IAVL had introduced critical flaws in the BSC bridge hack, where attackers exploited proof validation processes. As BSC's freeze functionality was tied to vulnerable IAVL usage, we enhanced our prompt to detect both IAVL implementations and their corresponding hotfixes as key investigation targets.

Observation

Take the Sei blockchain as an example:
Our AI identified extensive IAVL-related code across the codebase and concluded that "Sei is still using the IAVL library". In reality, Sei had migrated to the memIAVL implementation via a runtime configuration flag (sc-enable = true). Instead of rewriting all existing IAVL code, Sei introduced a switching mechanism that redirects IAVL operations to the more secure memIAVL whenever the flag is enabled.

Analysis

This case highlighted a core limitation of AI: surface-level code scanning without deep runtime analysis. The AI only performed superficial detection by identifying IAVL imports and function calls in the codebase, but it failed to examine the switching mechanisms that govern actual execution. It saw the presence of IAVL code and immediately assumed the vulnerability existed, without considering configuration flags, conditional logic or runtime redirections that might mitigate the risk. This shallow approach overlooks important architectural patterns where projects retain legacy code for compatibility while using runtime switches to redirect execution to more secure alternatives.

Solution

To address this limitation, prompts should be refined to explicitly ask about switching mechanisms, configuration flags or replacement code for IAVL functionality — an area we've already identified as a gap. Manual verification through GitHub commit history can also provide conclusive evidence of implemented fixes. A dual approach — combining AI detection that accounts for runtime logic with manual history review — offers the most reliable way to capture both surface-level usage and the deeper architectural choices that determine real vulnerability exposure.

3.4 Critical Reminders

In our initial analysis, we overlooked an important detail: some blockchain projects distribute their core functionality across multiple repositories. Our early approach focused only on the main repository, without considering that critical components might be located in separate repos.

Take Plume Chain as an example:

- **plume-go-ethereum:**
  Handles the core execution layer.

- **plume-nitro:**
  Manages rollup infrastructure and sequencing.

When we analyzed only the plume-go-ethereum repository, we mistakenly concluded that Plume lacked comprehensive freezing capabilities. In reality, the freezing mechanisms we were investigating were implemented in the nitro repository. If one only analyzed the go-ethereum repo, the full picture would be missed.

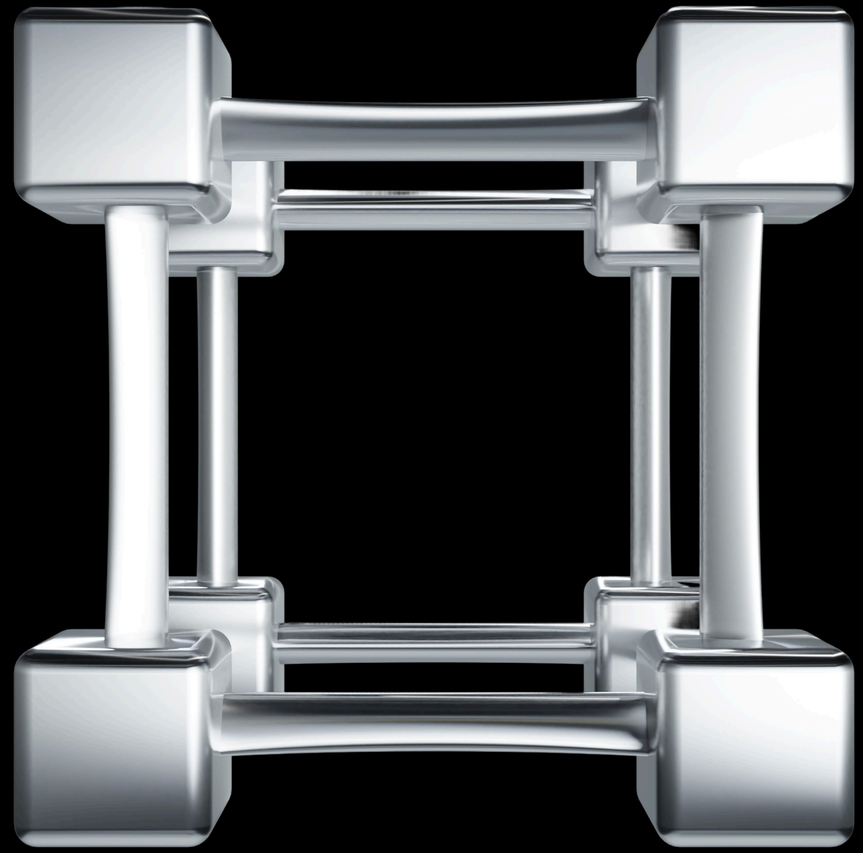
This kind of architectural split is common in modern L2 solutions, yet our initial approach treated each repo as a standalone system rather than recognizing them as interconnected parts of a larger blockchain infrastructure. Accounting for these multi-repository relationships is a nuanced but critical adjustment we need to make.

Conclusion

2025

Investigation, prompted by the SUI Foundation's intervention in the Cetus hack, has provided a critical analysis of censorship capabilities across 166 blockchain networks. The investigation was done with adequate combination of customized AI agents and deep-level manual code analysis. Our findings confirm that a total of 16 chains possess built-in freezing abilities, while an additional 19 have the potential for such functionalities. The research identified three distinct methodologies for asset freezing: hardcoded freezing, config file freezing, and on-chain smart contract freezing.

The presence of these mechanisms fundamentally challenges the foundational principles of a decentralized ecosystem and necessitates further discourse within the blockchain community, but it has prevented hackers from stealing funds. The study contributes to a more transparent understanding of these capabilities, providing a foundation for future research and risk assessment in the rapidly evolving digital asset space.



Chain								
ENJ	LKY	SOL	LTC	ONE	HBAR	MERLIN	MANTRA	PLASMA
MOVR	ZIRCUIT	MILKYWAY	BCH	BASE	LINEA	ZETAEVM	FIL	SEIEVM
CELO	QTUM	OAS	KDA	KON	HECO	VIC	ATOM	KLAY(KVM)
OP	ZETA	BTG	LUNA	MANTA	FLR	GUNZ	IMMUTABLE	TRX(TVM)
XDC	KLAY	RVN	LUNANEW	BB	B2	RUNE	MINA	ZIL(ZVM)
SONIC	STX	APTOS	INJ	CORN	VANA	KSM	XEC	CAMP
FTM	CHZ	EOS	EGLD	WAVES	PLUME	KASPA	CELESTIA	SCRT
ZKFAIR	NIBI	MATIC	STARKNET	MANTLE	AXL	HYPE	ETH	TENETCVM
MODE	ICX	TENET	DGB	NEAR	TON	ABSTRACT	VENOM	RUNE
ZKLINKNOVA	DASH	ARBI	BABYLON	POKT	LUK	ADA	NEM	AO
STORY	GLMR	CHILIZ	ARBINOVA	NILLION	ETHW	XRP	ALEO	CSPR2.0
BERA	KROMA	OPBNB	ZKSYNC	CAVAX	AVAIL	INITIA	HYPEEVM	EVMOS
ICP	ZEC	ZEN	BLAST	TAIKO	ETC	MOVEMENT	AR	NEON
BSC	SEI	DOT	DOGE	FITFI	XION	SUI	OKT	SOPHON
XTZ	FLOW	XYM	ROSE	ZKV2	WAXP	ACA	DCR	NUBIT
SCROLL	THETA	OMEGA	VECHAIN	XLM	DYM	DYMEVM	KAVA	EROSE
ZIL	WEMIX	BTC	CORE	XTER	DYDX	XAI	TRX	METIS
SUPRA	XAVAX	SC	HVH	CSPR	ALGO	CMP	CODEX	WORLD CHAIN
OSMO	KMA	ORDINAL	KAVAEVM					