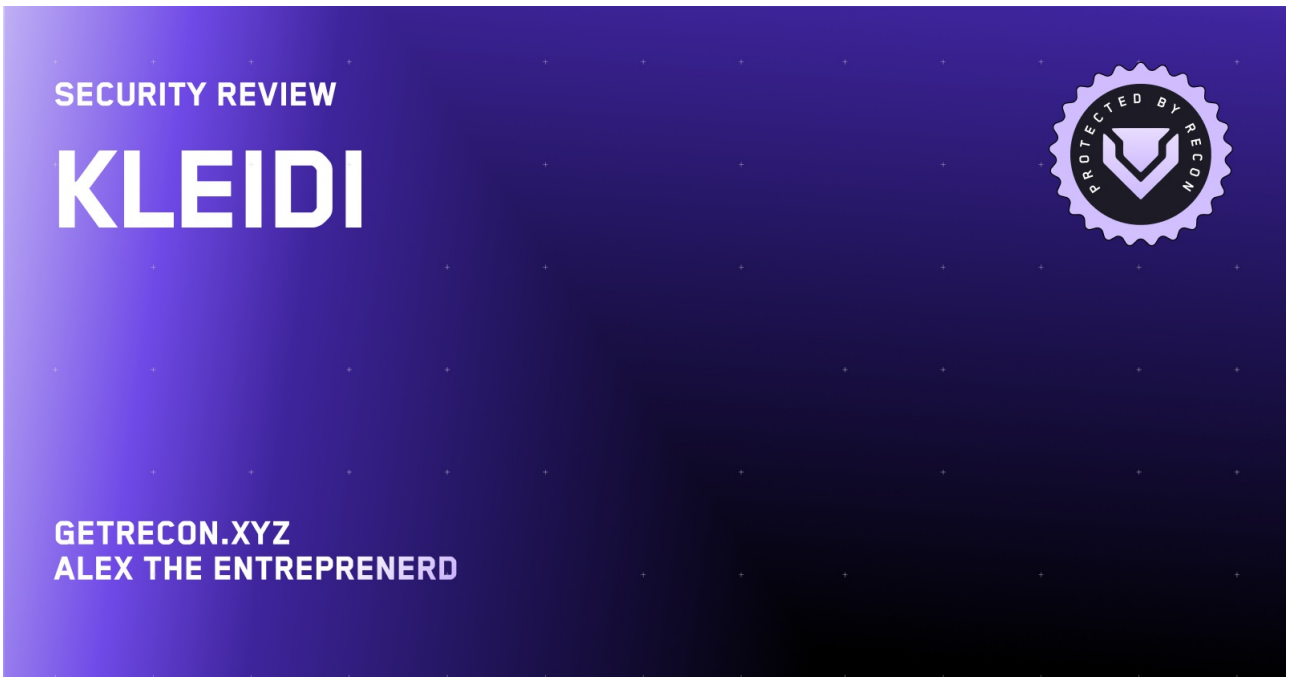

Recon Security Review

Introduction

Alex The Entreprenerd performed a 1 week review of kleidi

Repo:
https://github.com/solidity-labs-io/kleidi

Commit Hash:
1a06ac16bc99d0b4081281329d03064c3737f5e4

He additionally performed a complimentary mitigation review of fixes performed during the review, with head:
a5f6eb7e9eb5870f2f45b4f4b93e1b4da5f5cce1

This review uses Code4rena Severity Classification (https://docs.code4rena.com/awarding/judging-criteria/severity-categorization)

The Review is done as a best effort service, while a lot of time and attention was dedicated to the security review, it cannot guarantee that no bug is left

As a general rule we always recommend doing one additional security review until no bugs are found, this in conjunction with a Guarded Launch and a Bug Bounty can help further reduce the likelihood that any specific bug was missed

About Recon

Recon offers boutique security reviews, invariant testing development and is pioneering Cloud Fuzzing as a best practice by offering Recon Pro, the most complete tool to run tools such as Echidna, Medusa, Foundry, Kontrol and Halmos in the cloud with just a few clicks

About Alex

Alex is a well known Security Researcher that has collaborated with multiple contest firms such as:

- Code4rena - One of the most prolific and respected judges, won the Tapioca contest, at the time the 3rd highest contest pot ever
- Spearbit - Have done reviews for Tapioca, Threshold USD, Velodrome and more
- Recon - Centrifuge Invariant Testing Suite, Corn and Badger invariants as well as live monitoring

Table of Contents

Q-01 Suggested Next Steps

Executive Summary

The codebase is already very well tested and mature

Excluding known issues, such as performing MEV exposed operations, scenarios in which all parties turn malicious, or hypothetical scenarios in which Create2 hashing is broken, I wasn't able to identify any Medium nor High Severity risk in the codebase.

I believe the code can benefit by one more rounding of polish in terms of comments, and gas optimizations You may also opt into simplifying all functions by making them perform a single operation per function

For example addCalldataCheck currently is used to:

- Set a wildcard check
- Set a non wildcard check
- Add more hashes to a non wildcard check

The code can benefit by being simplified by turning each of these operation into a separate function

However, none of these seem to cause any particular security concern

Given my broad experience with these type of codebases, my recommendation for next steps is to perform a Audit Contest or a Bug Bounty next as I believe it will be hard to find any specific researcher that will be able to find any flaws I missed, while in aggregate, some researchers may found something that would escape the average researcher

A few possible things I may have missed:

- Is there any parameter that result in a salt that is ambigous or could cause a clash?
- Is there any specific scenario in which one of the ambigous parameters can result in a effective front-run?
- Is there a way to use SENTINEL to cause reverts for recovery spells?
- Is there any way to pass a forged payload that would result in calldata vs memory being read differently, possibly sidestepping the various check put in place in the system?

I have extensively explored these ideas, however someone with a different background or tools may find something I missed

Q-02 Timelock Permissionless execute could cause issues and out of order operations when multiple operations are ready

Impact

execute can be called by anyone

Meaning that once an operation is ready, it can be broadcasted by any caller

This also means that if more than one operation is ready at a specific time, the order of execution may be altered

This could be done for multiple reasons:

- Cause a revert
- Create MEV opportunities
- Cause a misconfiguration at the end of the sequence

Sonne Example

An extreme example is what happened with Sonne finance:
https://rekt.news/sonne-finance-rekt/
https://medium.com/@SonneFinance/post-mortem-sonne-finance-exploit-12f3daa82b06

They had queued the creation of a Compound Fork Market, the setting of 0% collateral factor, adding collateral and burning them as 3 separate operations, allowing the exploiter to only perform the creation of the market

Realistic Example

In the case of the Timelock, because certain operations such as removeCalldataCheck and removeCalldataCheckDatahash rely on an index, out of order operations may cause reverts or misconfiguration

Mitigation

It's important to document the risks of not using batch operations to end users

As long as end users batch their operations, no meaningful risk should be present

Q-03 Timelock unnecessary call of _setRoleAdmin

The call

https://github.com/solidity-labs-io/kleidi/blob/1a06ac16bc99d0b4081281329d03064c3737f5e4/src/Timelock.sol#L307-L308

```
    /// set the admin of the hot signer role to the default admin role
    _setRoleAdmin(HOT_SIGNER_ROLE, DEFAULT_ADMIN_ROLE); /// @audit prob implict
```

Is unnecessary

Since DEFAULT_ADMIN_ROLE == 0x00

Meaning that it will be admin role by default

Q-04 Timelock encoding of bytes is unambigous while strings may cause issues

Encoding (UI risk)

This test fails

```
bytes wrongCheck = "a54D3c09E34aC96807c1CC397404bF2B98DC4eFb";
bytes rightCheck = "a54d3c09E34aC96807c1CC397404bF2B98DC4eFb";

function test_bytes_checksum() public {
    bytes32 kak1 = keccak256(wrongCheck);
    bytes32 kak2 = keccak256(rightCheck);

    assertEq(kak1, kak2, "same res");
}
```

This doesn't

```
bytes wrongCheck = hex"a54D3c09E34aC96807c1CC397404bF2B98DC4eFb";
bytes rightCheck = hex"a54d3c09E34aC96807c1CC397404bF2B98DC4eFb";

function test_bytes_checksum() public {
    bytes32 kak1 = keccak256(wrongCheck);
    bytes32 kak2 = keccak256(rightCheck);

    assertEq(kak1, kak2, "same res");
}
```

Because the first one is converting the bytes to literals
While the second one is converting them from hex, which is consistent with encodePacked values

It's important that while the UI shows user friendly values, that all values passed to the smart contract are hex bytes, to avoid additional layers of encoding

Q-05 Refactoring - Timelock wildcard and non wildcard checks could be simplified

Refactoring Analysis

Fundamentally:
You either accept all (wildcard)
Or you check some
And checking some is tied to exact matches on some pieces

Each bytes check has a list of values afaict

So there is prob some risk there
But also not sure if it can be avoidable

In TS
Check {
isWildcard: boolean,
actualChecks: Check[]
}

Where each check is the segmented check

Start
End
Self Address Check
Data

Q-06 Timelock isSelfAddressCheck can be removed

https://github.com/solidity-labs-io/kleidi/blob/1a06ac16bc99d0b4081281329d03064c3737f5e4/src/Timelock.sol#L1166-L1177

```
        if (isSelfAddressCheck[i]) {
            /// self address check, data must be empty
            require(
                data[i].length == 0,
                "CalldataList: Data must be empty for self address check"
            );
            require(
                endIndex - startIndex == 20,
                "CalldataList: Self address check must be 20 bytes"
            );
            dataHash = ADDRESS_THIS_HASH;
        } else {
```

The hash is known at time of setup, so there's no advantage in having this code

Q-07 Timelock require has a typo

https://github.com/solidity-labs-io/kleidi/blob/1a06ac16bc99d0b4081281329d03064c3737f5e4/src/Timelock.sol#L1104-L1105

```
                "CalldataList: End index eqauls start index only when 4"
```

Q-08 Timelock checks for 1 instead of DONE_TIMESTAMP

This should be DONE_TIMESTAMP

https://github.com/solidity-labs-io/kleidi/blob/1a06ac16bc99d0b4081281329d03064c3737f5e4/src/Timelock.sol#L430-L431

```
    require(timestamp != 1, "Timelock: operation already executed");
```

Q-09 Timelock prevents multiple replays, but is subject to cross operation reentrancy

Impact

The code for execute is as follows:

https://github.com/solidity-labs-io/kleidi/blob/a5f6eb7e9eb5870f2f45b4f4b93e1b4da5f5cce1/src/Timelock.sol#L595-L608

```
    bytes32 id = hashOperation(target, value, payload, salt);

    /// first reentrancy check, impossible to reenter and execute the same
    /// proposal twice
    require(_liveProposals.remove(id), "Timelock: proposal does not exist");
    require(isOperationReady(id), "Timelock: operation is not ready");

    _execute(target, value, payload);
    emit CallExecuted(id, 0, target, value, payload);

    /// second reentrancy check, second check that operation is ready,
    /// operation will be not ready if already executed as timestamp will
    /// be set to 1
    _afterCall(id);
```

Before executing it will remove the id from liveProposals, making a double execution of the function impossible

There's a redundant check in _afterCall(id); that also performs a check against replay

However, the function doesn't protect against performing reentrancy between multiple operations

Meaning that mid execution of an operation, another operation could be performed

This typically can only happen when the Timelock interacts with untrusted, malicious contracts, so barring an uninteded use this shouldn't cause any major damage

Mitigation

Consider adding a reentrancyGuard or simply document this risk to end users to ensure they do not re-enter mid execution

Q-10 Timelock CalldataAdded will log all dataHashes instead of the ones being added

Event Data

https://github.com/solidity-labs-io/kleidi/blob/1a06ac16bc99d0b4081281329d03064c3737f5e4/src/Timelock.sol#L1198-L1199

```
        indexes[targetIndex].dataHashes.values() /// @audit technically adding all, also technically will emit a hash not the actual value
```

Q-11 RecoverySpell Comment only applies to benign spells

Impact

The comment:

https://github.com/solidity-labs-io/kleidi/blob/1a06ac16bc99d0b4081281329d03064c3737f5e4/src/RecoverySpell.sol#L112-L117

```
    /// no checks on parameters as all valid recovery spells are
    /// deployed from the factory which will not allow a recovery
    /// spell to be created that does not have valid parameters.
    /// A recovery spell can only be created by the factory if the Safe has
    /// already been created on the chain the RecoverySpell is being
    /// deployed on.
```

Only applies to benign spells

Malicious spells can be deployed anywhere by the compromised hot signer

Q-12 Once a RecoverySpell is deployed, all spells on all chains may be deployed

Impact

RecoverySpells are meant to be used in a time of emergency

For various reasons a recovery may need to be performed exclusively on one chain

However, calling createRecoverySpell will leak key details about the RecoverySpell

https://github.com/solidity-labs-io/kleidi/blob/1a06ac16bc99d0b4081281329d03064c3737f5e4/src/RecoverySpellFactory.sol#L44-L51

```
function createRecoverySpell(
    bytes32 salt,
    address[] memory owners,
    address safe,
    uint256 threshold,
    uint256 recoveryThreshold,
    uint256 delay
) external returns (RecoverySpell recovery) {
```

More specifically it will inform everyone about the parameters that would result in the RecoverySpell being deployed on every other chain

In the scenario in which a recovery was meant to be performed only on Chain A, for all other chains, anyone could deploy the RecoverySpell causing them to have a reduced delay for all those chains

Mitigation

In your documentation you should clarify that if recovery happens on one chain, it should probably happen on all chains

Q-13 InstanceDeployer - Could use constant for Flag Previous Owner

Could use CONSTANT instead of 1

https://github.com/solidity-labs-io/kleidi/blob/1a06ac16bc99d0b4081281329d03064c3737f5e4/src/InstanceDeployer.sol#L294-L302

```
        calls3[index++].callData = abi.encodeWithSelector(
            OwnerManager.swapOwner.selector,
            /// previous owner
            address(1),
            /// old owner (this address)
            address(this),
            /// new owner, the first owner the caller wants to add
            instance.owners[0]
        );
```

Q-14 Guard.sol comment on safe having no funds is technically inaccurate

https://github.com/solidity-labs-io/kleidi/blob/1a06ac16bc99d0b4081281329d03064c3737f5e4/src/Guard.sol#L29-L30

```
/// Refund receiver and gas params are not checked because the Safe itself
/// does not hold funds or tokens.
```

You technically cannot guarantee this

Some funds could be there and may even be used or active

Q-15 Admin Sidestep - Guard.sol DelegateCall is still possible via Modules

Delegatecall could still be performed by setting up modules

IMO this is fine, however, signers should be warned that adding a module could cause a full sidestep of the security guarantees of the project

The same can happen via a fallback handler

Q-16 ConfigurablePause._grantGuardian naming could be changed to setGuardian

Grant Guardian is setting to address(0) as well, meaning it's also a revoke, setGuardian seems to be most appropriate

Q-17 ConfigurablePause.sol some indexed parameters are not particularly useful

Not sure it makes sense to index this

https://github.com/solidity-labs-io/kleidi/blob/1a06ac16bc99d0b4081281329d03064c3737f5e4/src/ConfigurablePause.sol#L46-L47

```
event PauseTimeUpdated(uint256 indexed newPauseStartTime);
```

That would be helpful to find all events at a time, but that's a pretty niche thing to do

Same here

https://github.com/solidity-labs-io/kleidi/blob/1a06ac16bc99d0b4081281329d03064c3737f5e4/src/ConfigurablePause.sol#L51-L54

```
event PauseDurationUpdated(
    uint256 indexed oldPauseDuration, uint256 newPauseDuration
);
```

Q-18 Guard QA Report

Analysis

Fundamentally enforces that no delegate call can be called from the safe

It can still be called via the Modules and MAYBE via the fallback module

Seems to be upgrade resistant and I don't believe upgrades are "intended"

You cannot guarantee the safe will be empty
But the behaviour of the safe is pretty much normal when it comes to executing operations

This seems to remove the ability to batch operations because batches are done via delegatecalls

QA

Admin sidestep

Delegatecall could still be performed by setting up modules

AFAICT this is still allowd by the Timelock

IMO this is fine, however, signers should be warned that adding a module could cause a full sidestep of the security guarantees of the project

The same can happen via a fallback handler

Hunch - Takeover via malicious fallback handler?

https://github.com/solidity-labs-io/kleidi/blob/1a06ac16bc99d0b4081281329d03064c3737f5e4/src/InstanceDeployer.sol#L155-L156

```
        /// no fallback handler allowed by Guard
        address(0),
```

-> Front-run
-> Deploy the guard
-> Shrekt?

NITS?

Safe flow for owners, and other configs

Technically cannot guarantee this

Some funds could be there and may even be used or active

https://github.com/solidity-labs-io/kleidi/blob/1a06ac16bc99d0b4081281329d03064c3737f5e4/src/Guard.sol#L29-L30

```
/// Refund receiver and gas params are not checked because the Safe itself
/// does not hold funds or tokens.
```

Fallback missing

Likelihood of this being an issue is extremely low
Also since upgrades are not contemplated this seems fine as is

Notes

Self Calls via

https://github.com/safe-global/safe-smart-account/blob/0142ec8a4a05f03167daba9e7231b7e858aabd32/contracts/base/ModuleManager.sol#L154-L163

```
function execTransactionFromModule(
    address to,
    uint256 value,
    bytes memory data,
    Enum.Operation operation
) external override returns (bool success) {
    (address guard, bytes32 guardHash) = preModuleExecution(to, value, data, operation);
    success = execute(to, value, data, operation, type(uint256).max);
    postModuleExecution(guard, guardHash, success);
}
```

So timelock can still change settings since it's a module

Q-19 Different Threshold can result in different hash but same config when using one owner

Impact

When calling createSystemInstance, a Safe with a single owner could be deployed, while passing a threshold that is above 1

Because of the check if (instance.owners.length > 1) { the threshold will not be validated in those cases

Because threshold is part of the salt, this will result in a unique deployment, which will share the configuration with other deployments

https://github.com/solidity-labs-io/kleidi/blob/1a06ac16bc99d0b4081281329d03064c3737f5e4/src/InstanceDeployer.sol#L308-L329

```
        for (uint256 i = 1; i < instance.owners.length - 1; i++) {
            calls3[index++].callData = abi.encodeWithSelector(
                OwnerManager.addOwnerWithThreshold.selector,
                instance.owners[i],
                1
            );
        }

        /// if there is only one owner, the threshold is set to 1
        /// if there are more than one owner, add the final owner with the
        /// updated threshold
        if (instance.owners.length > 1) {
            /// add final owner with the updated threshold
            /// if threshold is greater than the number of owners, that
            /// will be caught in the addOwnerWithThreshold function with
            /// error "GS201"
            calls3[index++].callData = abi.encodeWithSelector(
                OwnerManager.addOwnerWithThreshold.selector,
                instance.owners[instance.owners.length - 1],
                instance.threshold
            );
        }
```

Mitigation

You could enforce that the threshold is met by the number of owners

However there is no particular impact to this finding

Q-20 Refactoring - Roles are not used and could be removed

Impact

Overall roles are used only for DEFAULT_ADMIN_ROLE (self) and for HOT_SIGNER_ROLE because of this, you could simply remove the roles logic and leave a way to add and remove HOT_SIGNERS

https://github.com/solidity-labs-io/kleidi/blob/1a06ac16bc99d0b4081281329d03064c3737f5e4/src/Timelock.sol#L774-L806

```
function grantRole(bytes32 role, address account)
    public
    override(AccessControl, IAccessControl)
{
    require(role != DEFAULT_ADMIN_ROLE, "Timelock: cannot grant admin role");
    super.grantRole(role, account);
}

/// @notice function to revoke a role from an address
/// @param role the role to revoke
/// @param account the address to revoke the role from
function revokeRole(bytes32 role, address account)
    public
    override(AccessControl, IAccessControl)
{
    require(
        role != DEFAULT_ADMIN_ROLE, "Timelock: cannot revoke admin role"
    );
    super.revokeRole(role, account);
}

/// @notice function to renounce a role
/// @param role the role to renounce
/// @param account the address to renounce the role from
function renounceRole(bytes32 role, address account)
    public
    override(AccessControl, IAccessControl)
{
    require(
        role != DEFAULT_ADMIN_ROLE, "Timelock: cannot renounce admin role"
    );
    super.renounceRole(role, account);
}
```

Mitigation

Delete the functions

Add a function grantHotSigner that requires calling self

If you wish to allow the Safe to also be able to grant hotSigners, you could simply add a role check to grantHotSigner

Q-21 RecoverySpell Gas Optimizations

Remove address(0) check, this is already done by OZ.ECDSA

https://github.com/solidity-labs-io/kleidi/blob/1a06ac16bc99d0b4081281329d03064c3737f5e4/src/RecoverySpell.sol#L226-L230

```
        /// will be 0 on the second retrieval and the require will fail.
        require( /// @audit Pretty sure OZ does this
            valid && recoveredAddress != address(0),
            "RecoverySpell: Invalid signature"
        );
```

Ownership check can be done in memory more cheaply

The code change is pretty annoying compared to this, but memory should be cheaper

https://github.com/solidity-labs-io/kleidi/blob/1a06ac16bc99d0b4081281329d03064c3737f5e4/src/RecoverySpell.sol#L218-L221

```
        assembly ("memory-safe") {
            valid := tload(recoveredAddress)
            if eq(valid, 1) { tstore(recoveredAddress, 0) } /// @audit Cannot recover more than once per address
        }
```

Copy owners (unnecessary if immutable)
For each of them store true in an array at their index

For each signer, find the owner in the array and check that the value was true, set the value to false

This should cost less because the cost of finding the address will tend to be lower for the average user count (less than 10)

Q-22 removeCalldataCheckDatahash could mistakenly remove a wildcard check if the wildcard check is added with an empty dataHash

Impact

_addCalldataCheck doesn't enforce that a wildcard check has no dataHashes

https://github.com/solidity-labs-io/kleidi/blob/1a06ac16bc99d0b4081281329d03064c3737f5e4/src/Timelock.sol#L1062-L1112

```
function _addCalldataCheck(
    address contractAddress,
    bytes4 selector,
    uint16 startIndex,
    uint16 endIndex,
    bytes[] memory data,
    bool[] memory isSelfAddressCheck
) private {
    require(
        contractAddress != address(0),
        "CalldataList: Address cannot be zero"
    );
    require(selector != bytes4(0), "CalldataList: Selector cannot be empty");
    require(
        startIndex >= 4, "CalldataList: Start index must be greater than 3"
    );
    require(
        data.length == isSelfAddressCheck.length,
        "CalldataList: Array lengths must be equal"
    );
    /// prevent misconfiguration where a hot signer could change timelock
    /// or safe parameters
    require(
        contractAddress != address(this),
        "CalldataList: Address cannot be this"
    );
    require(contractAddress != safe, "CalldataList: Address cannot be safe");

    Index[] storage calldataChecks =
        _calldataList[contractAddress][selector];
    uint256 listLength = calldataChecks.length;
    /// @audit If you add wildcard, you need to remove it before adding something else
    if (listLength == 1) {
        require(
            calldataChecks[0].startIndex != calldataChecks[0].endIndex,
            "CalldataList: Cannot add check with wildcard"
        );
    }

    if (startIndex == endIndex) {
        require(
            startIndex == 4,
            "CalldataList: End index eqauls start index only when 4"
        );
        require(
            listLength == 0,
            "CalldataList: Add wildcard only if no existing check"
        );

        /// @audit Add `data.length == 0` so you ensure this check is correct
    } else {
```

A Wildcard Could be added with Data by mistakes due to a lack of this check

In removing the dataHash, the wildcard would also be removed by removeCalldataCheckDatahash:

https://github.com/solidity-labs-io/kleidi/blob/1a06ac16bc99d0b4081281329d03064c3737f5e4/src/Timelock.sol#L903-L908

```
function removeCalldataCheckDatahash(
    address contractAddress,
    bytes4 selector,
    uint256 index,
    bytes32 dataHash
) external onlyTimelock {
```

POC

You can pass an empty string and the check will pass, please see this test which shows:

- Setup the check with empty bytes
- Remove the empty bytes
- Removes the wildcard check

forge test --match-test test_show_arbitraryCheck -vv

```
function test_show_arbitraryCheck() public {
    address[] memory targetAddresses = new address[](1);
    targetAddresses[0] = address(lending);

    bytes4[] memory selectors = new bytes4[](1);
    selectors[0] = MockLending.deposit.selector;

    /// compare first 20 bytes
    uint16[] memory startIndexes = new uint16[](1);
    startIndexes[0] = 4;

    uint16[] memory endIndexes = new uint16[](1);
    endIndexes[0] = 4;

    bytes[][] memory checkedCalldatas = new bytes[][](1);
    bytes[] memory checkedCalldata1 = new bytes[](1);
    checkedCalldata1[0] = hex"";
    checkedCalldatas[0] = checkedCalldata1;

    bool[][] memory isSelfAddressChecks = new bool[][](1);
    bool[] memory isSelfAddressCheck = new bool[](1);
    isSelfAddressCheck[0] = false;
    isSelfAddressChecks[0] = isSelfAddressCheck;

    vm.prank(address(timelock));
    timelock.addCalldataChecks(
       targetAddresses,
       selectors,
       startIndexes,
       endIndexes,
       checkedCalldatas,
       isSelfAddressChecks
    );

    // Check that the check is there
    timelock.checkCalldata(targetAddresses[0], abi.encodePacked(selectors[0]));

    // Remove it
    vm.prank(address(timelock));
    timelock.removeCalldataCheck(targetAddresses[0], selectors[0], 0);

    // Show that now it reverts
    vm.expectRevert();
    timelock.checkCalldata(targetAddresses[0], abi.encodePacked(selectors[0]));
 }
```

Mitigation

Add a check to enforce that data.length is 0 for wildcard checks

```
    if (startIndex == endIndex) {
        require(
            startIndex == 4,
            "CalldataList: End index eqauls start index only when 4"
        );
        require(
            listLength == 0,
            "CalldataList: Add wildcard only if no existing check"
        );

        /// @audit Add `data.length == 0` so you ensure this check is correct
```

Q-23 getFunctionSignature can be refactored to not use assembly

Impact

getFunctionSignature uses assembly as follows

https://github.com/solidity-labs-io/kleidi/blob/1a06ac16bc99d0b4081281329d03064c3737f5e4/src/BytesHelper.sol#L7-L18

```
  function getFunctionSignature(bytes memory toSlice)
      public
      pure
      returns (bytes4 functionSignature)
  {
      require(toSlice.length >= 4, "No function signature");

      assembly ("memory-safe") {
          functionSignature := mload(add(toSlice, 0x20))
      }
      return bytes4(toSlice);
  }
```

But you can simply cast bytes to bytes4 instead

Proof

```
  function getFunctionSignature(bytes memory toSlice)
      public
      pure
      returns (bytes4 functionSignature)
  {
      require(toSlice.length >= 4, "No function signature");

      assembly ("memory-safe") {
          functionSignature := mload(add(toSlice, 0x20))
      }
  }

  function test_check(bytes memory theBytes) public {
      vm.assume(theBytes.length >= 4);
      bytes4 res = getFunctionSignature(theBytes);
      bytes4 normal = bytes4(theBytes);

      assertEq(res, normal);
  }
```

Q-24 Original Hot Signer that becomes Compromised or Malicious can take over all undeployed systems

Impact

The known issues specify:

https://github.com/solidity-labs-io/kleidi/blob/1a06ac16bc99d0b4081281329d03064c3737f5e4/docs/KNOWN_ISSUES.md#L5

```
- if the hot signers are malicious or compromised, they can deploy a compromised system instance on a new chain with compromised recovery spells and malicious calldata checks that allow funds to be stolen
```

This is correct, however, it's worth highlighting that malicious / compromised hot signers may be removed from a live deployment

However, because of the logic used to deploy the system on new networks, the previously removed hot signer would still be able to deploy and compromise the system

POC

- Setup system as intended on Chain A
- Do not deploy on Chain B
- Hot Signer N key gets leaked
- Remove Hot Signer N from Chain A
- Hot Signer N can deploy on Chain B and compromise it

Meaning that the risk of compromised Hot Signers doesn't end once a Hot Signer is removed from the original chain, but only if they are removed from all chains

Mitigation

Users should:

- Deploy all instances they want to secure
- Remove all hot signers from all deployed chains
- Setup each chain at the time in which that's necessary

It may also be best to enforce that one of the owners is setting up a deployment on any other chain, however this shares a similar risk that if they are compromised they could still compromised other chains

Q-25 Docs Typos

Merge marks

https://github.com/solidity-labs-io/kleidi/blob/1a06ac16bc99d0b4081281329d03064c3737f5e4/docs/TESTING.md#L53-L59

```
so that when there is a further sanity check failure by a mutant the spec can be marked as violated.
<<<<<<< HEAD
>>>>>>> 81590cd (remove redundant spec)
=======
>>>>>>> 0cf1c3c (remove redundant spec)
>>>>>>> 664c0e1 (remove redundant spec)
=======
```

Pseudocode doesn't exist

https://github.com/solidity-labs-io/kleidi/blob/1a06ac16bc99d0b4081281329d03064c3737f5e4/docs/CALLDATA_WHITELISTING.md#L34-L45

```
This means that to supply to any market, the same function is called with different calldata parameters. This is where the array of calldata checks is used.

```solidity
    struct Index {
        uint16 startIndex;
        uint16 endIndex;
        EnumerableSet.Bytes32Set dataHashes;
    }

    contract address => bytes4 function selector => Index[] calldataChecks;
```

Broken Link

https://github.com/solidity-labs-io/kleidi/blob/1a06ac16bc99d0b4081281329d03064c3737f5e4/README.md#L7
```

- **Security**: The protocol is designed by the world class Smart Contract engineering team at Solidity Labs. It has had components formally verified, and has been audited twice with no critical or high issues discovered. It is designed to be used with Gnosis Safe multisigs, which are battle-tested and have secured tens of billions of dollars in assets. See our internal audit log [here](docs/security/AUDIT_LOG.md).

Mitigation

Should be /docs/AUDIT_LOG.md

Q-26 Create 2 Hash Collision

Impact

Because of the fact that the system uses deterministic addresses, given the fact that keccak (a 32 bytes function) is truncated down to a uint160 (20 bytes), the likelihood that a clash is found is around 2^80

This makes brute-force mining a clash plausible, although extremely expensive

While the risk of mining a collision is possible, this should not be feasible for any specific Safe in the system as the cost of mining a clash is estimated in the order of billions of dollars

Notes

It is worth going through these notes to familiarise yourself with these types of findings

https://eips.ethereum.org/EIPS/eip-3607
https://github.com/sherlock-audit/2024-06-makerdao-endgame-judging/issues/64
https://github.com/sherlock-audit/2023-07-kyber-swap-judging/issues/90

G-01 RecoverySpell save gas by setting recoveryInitiated = 0 to signify a Disabled Spell

Gas - Set it to 0 to save gas due to refund

https://github.com/solidity-labs-io/kleidi/blob/1a06ac16bc99d0b4081281329d03064c3737f5e4/src/RecoverySpell.sol#L302

```
        recoveryInitiated = type(uint256).max;
```

There is no zero timestamp so this is a safe change

G-02 ConfigurablePause.sol - Save gas by using uint48

https://github.com/solidity-labs-io/kleidi/blob/1a06ac16bc99d0b4081281329d03064c3737f5e4/src/ConfigurablePause.sol#L15-L23

```
/// @notice pause start time, starts at 0 so contract is unpaused
uint128 public pauseStartTime;

/// @notice pause duration
uint128 public pauseDuration;

/// @notice address of the pause guardian
address public pauseGuardian;
```

u48 is plenty and will allow one slot for all 3 values

Additional Services by Recon

Recon offers:

- Ongoing advisory and invariant testing - Ask about Recon Legendary
- Cloud Fuzzing as a Service - The easiest way to run invariant tests in the cloud - Ask about Recon Pro
- Security Reviews by Alex The Entreprenerd and the Recon Team