SPEARBIT

Overprotocol vCISO Security Review

Auditors

Dtheo, Lead Security Researcher

Jtraglia, Lead Security Researcher

Shotes, Security Researcher

Report prepared by: Lucas Goiriz

July 22, 2024

Contents

1 About Spearbit

Spearbit is a decentralized network of expert security engineers offering reviews and other security related services to Web3 projects with the goal of creating a stronger ecosystem. Our network has experience on every part of the blockchain technology stack, including but not limited to protocol design, smart contracts and the Solidity compiler. Spearbit brings in untapped security talent by enabling expert freelance auditors seeking flexibility to work on interesting projects together.

Learn more about us at spearbit.com

2 Introduction

OverProtocol is a brand new layer 1 with lightweight nodes empowering personal computers, enabling anyone to run a node on their PCs and become a validator. This is made possible by OverProtocol's layered architecture through Ethanos, which significantly decrease the resources required for block validation.

Disclaimer: This security review does not guarantee against a hack. It is a snapshot in time of kairos2.0 according to the specific commit. Any modifications to the code will require a new security review.

3 Risk classification

Severity level	**Impact: High**	**Impact: Medium**	**Impact: Low**
Likelihood: high	Critical	High	Medium
Likelihood: medium	High	Medium	Low
Likelihood: low	Medium	Low	Low

3.1 Impact

- High - leads to a loss of a significant portion (>10%) of assets in the protocol, or significant harm to a majority of users.
- Medium - global losses <10% or losses to only a subset of users, but still unacceptable.
- Low - losses will be annoying but bearable--applies to things like griefing attacks that can be easily repaired or even gas inefficiencies.

3.2 Likelihood

- High - almost certain to happen, easy to perform, or not easy but highly incentivized
- Medium - only conditionally possible or incentivized, but still relatively likely
- Low - requires stars to align, or little-to-no incentive

3.3 Action required for severity levels

- Critical - Must fix as soon as possible (if already deployed)
- High - Must fix (before deployment if not already deployed)
- Medium - Should fix
- Low - Could fix

4 Executive Summary

Over the course of 7 days in total, Overprotocol engaged with Spearbit to review the kairos2.0 protocol. In this period of time a total of **22** issues were found.

Each researcher utilized a maximum of 20 hours. The total maximum hours for this entire engagement did not exceed 60 hours between all three researchers.

Summary

Project Name	Overprotocol
Repository	kairos2.0
Commit	0ea8d2...21cba7
Type of Project	Infrastructure, Node
Audit Timeline	May 1 to May 8

Issues Found

Severity	**Count**	**Fixed**	**Acknowledged**
Critical Risk	4	0	0
High Risk	1	0	0
Medium Risk	6	0	0
Low Risk	1	0	0
Gas Optimizations	0	0	0
Informational	10	0	0
Total	**22**	**0**	**0**

5 Findings

5.1 Critical Risk

5.1.1 Malicious Restoration Servers can replay `RestoreData` messages and drain accounts

Severity: Critical Risk

Context: core/types/restore_data.go#L10

Description: The `RestoreData` struct does not have a nonce and is not rooted to the first `RestorationTX` that uses it in any way. There is also nothing in the EVM that prevents successfully processing a `RestorationTX` for a non-dormant account. These factors contribute to a situation that allows for malicious restoration servers to make as many `RestorationTX` transactions as is needed to drain the account of any user that makes a `requestRestoration` API call.

Recommendation: Either add a nonce to the `RestoreData` struct, have some other way of rooting the `RestoreData` message's signature to a single `RestorationTX`, allow only one `RestorationTX` per account per epoch, or implement some other method that prevents replays of the `RestoreData` message.

5.1.2 Prefunded contracts can permanently lose funds if account becomes dormant before code is deployed

Severity: Critical Risk

Context: core/vm/evm.go#L891

Description: Contracts that receive funds before they are deployed can permanently lose the funds if the account goes dormant before the contract is deployed. It is not an uncommon pattern for people to prefund contracts whose addresses are known ahead of time (via CREATE2). The issue is that once the contract is deployed there is no way for a `restorationTX` to restore the account because of the `common.BytesToHash(account.CodeHash) != types.EmptyCodeHash` check in `(evm *EVM) verifyRestorationProof()`. If a `restorationTX` is made on behalf of the contract's address before it is deployed then the funds will be saved. If any funds are dormant and are not restored prior to the contract deployment then they will be forever lost.

Recommendation: Consider adding support for restoring old balances to a contract. At minimum clearly document this in the Overprotocol documentation.

5.1.3 Malicious request could burn restoration server gas fees

Severity: Critical Risk

Context: internal/ethapi/api.go#L800-L848, core/vm/evm.go#L891-L894

Description: The `GetRestorationProof` function does not check if it is creating a proof for a contract address, while the `Restore` transaction will fail to `verifyRestorationProof` if `Target` is a contract address.

Without this check, a malicious actor could request restorations for a contract address that the restoration server recognizes as valid, since `GetRestorationProof` would return a valid restoration proof without checking if it's a contract. The restoration server would then send Restore transactions to the EVM which would inevitably fail due to the `Target` being a contract. This would result in the restoration server losing its gas fees without any reprecussions to the original requester.

This would result in a scenario where a malicious actor could forcibly burn the entire balance on any restoration server.

Recommendation: `GetRestorationProof` should check if the target address is a contract address as soon as possible. Ideally, this happens before any proofs are generated to prevent any unecessary computation. Right after internal/ethapi/api.go#L801-L804 seems like a good spot if the target address exists within the returned statedb. Otherwise, you could mirror the check that happens in the EVM by checking in internal/ethapi/api.go#L829-L834.

5.1.4 Malicious request can crash the restoration server

Severity: Critical Risk

Context: cmd/restoration/server.go#L175

Description: An attacker can send a request to a restoration server with a nil `Fee` which causes it to panic via nil dereference. We believe the authors were working under the assumption that the `gencodec:"required"` struct tags would ensure required fields are non-nil, but this does not apply to JSON deserialization.

Here's a simple proof of concept:

```go
package main

import (
    "fmt"
    "github.com/gofiber/fiber/v2"
    "math/big"
)

type Foo struct {
    Fee *big.Int `json:"fee" gencodec:"required"`
}

func main() {
    app := fiber.New()

    app.Post("/foo", func(ctx *fiber.Ctx) error {
        foo := new(Foo)
        if err := ctx.BodyParser(foo); err != nil {
            return ctx.Status(400).SendString(err.Error())
        }
        ret := fmt.Sprintf("foo.Fee is %v", foo.Fee)
        return ctx.Status(200).SendString(ret)
    })

    fmt.Println(app.Listen(":3001"))
}
```

A request with a null fee shows that the value can be nil:

```
$ curl -X POST \
  http://localhost:3001/foo \
  -H 'Content-Type: application/json' \
  -d '{"fee": null}'
foo.Fee is <nil>%
```

This matters because when you pass `Fee` to `Cmp`, it will panic if `Fee` is nil.

```go
func TestCmpWithNil(t *testing.T) {
    big.NewInt(64).Cmp(nil)
}
```

This test will panic with the following output:

```
=== RUN   TestCmpWithNil
--- FAIL: TestCmpWithNil (0.00s)
panic: runtime error: invalid memory address or nil pointer dereference [recovered]
    panic: runtime error: invalid memory address or nil pointer dereference
[signal SIGSEGV: segmentation violation code=0x2 addr=0x0 pc=0x1044cf52c]
```

Recommendation: Add new checks in `checkRestoreData` which ensure required fields are not nil.

5.2 High Risk

5.2.1 `EVM.Restore()` gas not consumed on invalid signature

Severity: High Risk

Context: core/vm/evm.go#L803

Description: `(evm *EVM) Restore()` does not consume all gas on `RestoreData` signatures. An attacker can generate incredibly large `restorationTX`s that will consume the nodes resources validating the restoration server's signature on a large data buffer (eg `input/proof`). If the `RestoreData` signature fails then the cost of this large verification will only be the intrinsic gas as the memory cost calculation will never be hit. This will allow for a chain-wide DOS vector that costs almost no gas to implement.

Recommendation: An honest restoration server should never create a transaction out of a `requestRestoration` request if the signature is invalid. If this happens the EVM should consume all of the `restorationTX` gas to disincentivize this behavior.

5.3 Medium Risk

5.3.1 Restoration Server `result.RestoredBalance` `nil` pointer dereference DoS

Severity: Medium Risk

Context: gethclient/gethclient.go#L159

Description: The restoration server is vulnerable to a `nil` pointer dereference panic `(panic: runtime error: invalid memory address or nil pointer dereference)` if its `gethclient` returns a null json for `restorationProofResult.RestoredBalance`. This will cause a DOS in the restoration server. While the ideal situation is that every restoration server also runs its own `gethclient` the restoration server does allow remote connections for this service.

Recommendation: Add a `if restorationProofResult.RestoredBalance == nil` check in `(ec *Client) GetRestorationProof()` and error gracefully if the pointer is `nil`.

5.3.2 `eth_getRestorationProof` API exposure possible DoS vector

Severity: Medium Risk

Context: internal/ethapi/api.go#L808

Description: The `eth_getRestorationProof` API can be computationally intensive. There is no way to prevent queries that set the `TargetEpoch` to 0 as this may be valid in some settings. The longer the chain is around the more work responding to a query like this will take. This introduces a DoS vector for public RPC servers as long as they cannot turn of this API when providing regular `eth` APIs.

Recommendation: Move the `getRestorationProof` API to a different name space that is not enabled by default like `restore_getRestorationProof`.

5.3.3 `ChainID` should not be 1

Severity: Medium Risk

Context: params/config.go#L44, params/bootnodes.go#L23-L52, cmd/clef/main.go#L103, cmd/clef/README.md#L32

Description: There are references in the codebase to Ethereum related bootnode ENRs as well as chainID defaults that are equal to Ethereum mainnet and various testnets.

Recommendation: The default `ChainID` for Overprotocol Mainnet should be changed to something unique and ENRs of all Ethereum related bootnodes should be removed from the codebase. This will prevent cross pollination with Ethereum nodes in mempools and during syncing. More importantly this will prevent the ability for transaction replays between chains.

5.3.4 Enforce `EpochLimit` value minimum size of 3 on chain creation

Severity: Medium Risk

Context: core/blockchain.go#L1438-L1444

Description: Each checkpoint block, the chain does through and deletes epochs based on the configured `EpochLimit`. Any value lower than 3 could delete required data.

Recommendation: Enforce the minimum `EpochLimit` value of 3 on chain creation, while still allowing the value of 0 for unlimited `EpochLimit`.

5.3.5 Mistake in ENR key for discovery

Severity: Medium Risk

Context: eth/protocols/eth/discovery.go#L36

Description: This function will return "over1" when it should return "over".

Recommendation: Change the return value to "over".

5.3.6 Gas checks in `Restore` should be performed as soon as possible

Severity: Medium Risk

Context: core/vm/evm.go#L812-L834

Description: Gas checks should be done as soon as possible to avoid unnecessary work. There are some operations, like a signature validation, prior to the gas checks. Additionally, if there were an error in these prior checks, it wouldn't cost the caller anything.

Recommendation: Move the gas checks to the top of the function.

5.4 Low Risk

5.4.1 `eth_getRestorationProof` will return empty proof

Severity: Low Risk

Context: internal/ethapi/api.go#L839

Description: If a `TargetEpoch` is provided to the `eth_getRestorationProof` that is above the previous `RestoredEpcoh` then the API will return an empty proof instead of erroring. This is not be caught in the restoration server logic or in the EVM processing of the `RestorationTX`. This will result in a waste of gas for the restoration server and the user requesting restoration.

Recommendation: Add logic to error if the restoration proof is empty and error before submitting the transaction. Consider adding logic to determine if the restoration will affect the users balance at all.

5.5 Informational

5.5.1 Wrong values are logged if `ckptRoot != ckptDiskRoot`

Severity: Informational

Context: trie/triedb/pathdb/journal.go#L116-L118

Description: This checks if `ckptRoot` matches `ckptDiskRoot`. If they don't match it logs `root` and `diskRoot`.

Recommendation: Replace `root` & `diskRoot` with `ckptRoot` & `ckptDiskRoot`

5.5.2 Value checks in journal should happen after each read

Severity: Informational

Context: trie/triedb/pathdb/journal.go#L110-L118

Description: In the `loadJournal` function, there are consecutive `Decode`'s from the rlp Stream followed by consecutive checks that the values are correct. While the rlp Stream likely has the data buffered from disk, it is good practice to check those values after each read. This will catch failures quickly and prevent any more reads than necessary.

Recommendation: Move the value checks at trie/triedb/pathdb/journal.go#L110-L118 up, so that they validate the values immediately after they are read.

5.5.3 Namespace overriding of `keystore` package

Severity: Informational

Context: cmd/restoration/main.go#L47

Description: The `keystore.NewKeyStore` value is assigned to variable `keystore`. This assignment overrides the keystore package namespace in the `main` function. This will result in being unable to use the `keystore` *package* in the future and may result in unexpected behavior if attempting to do so.

Recommendation: Change the variable name `keystore` so as to not interfere with the package name.

5.5.4 Nil value on initialization needs comment

Severity: Informational

Context: core/state/statedb.go#L67

Description: On initialization in the `New` function, if `currentEpoch == 0`, then the `ckptTrie` value will remain nil. This value is correctly checked for nil values everywhere it is used, but there is nothing indicating that this value could be nil.

Recommendation: Add a comment to the `StateDB` struct that indicates that `ckptTrie` is nil when `currentEpoch == 0`.

5.5.5 Unused function `isStale` can be removed

Severity: Informational

Context: trie/triedb/pathdb/checkpoint_disklayer.go#L71-L78

Description: The function `isStale` for `ckptDiskLayer` is never used, and it isn't required for implementing the `layer` interface.

Recommendation: Remove the function `isStale`.

5.5.6 Slightly misleading documentation for `SafeSub32`

Severity: Informational

Context: common/math/integer.go#L152-L156

Description: The comment above the function says that it checks for an "overflow" when it should say "underflow".

Recommendation: Replace "overflow" with "underflow" in the comment.

5.5.7 `EpochLength` could be confused with `SweepEpoch`

Severity: Informational

Context: core/rawdb/schema.go#L152

Description: `EpochLength` (size of an epoch value, `uint32`) could be confused with `SweepEpoch` (number of blocks in an epoch).

Recommendation: Add a comment to the constant which describes what it represents, or rename it so that it's obviously different.

5.5.8 Unused parameters in `checkProfitable`

Severity: Informational

Context: cmd/restoration/server.go#L230

Description: The `checkProfitable` helper function does not use `ctx` or `ethclient`.

Recommendation: Remove these two parameters from the function declaration.

5.5.9 Function unnecessarily returns an error

Severity: Informational

Context: core/vm/evm.go#L925-L930

Description: The `ProofDB::Put` function returns an error, but it will always be nil.

Recommendation: Remove the return type, so that it returns nothing.

5.5.10 Exported function with an unexported return type

Severity: Informational

Context: core/types/restore_data_signer.go#L42

Description: `NewAlpacaRestoreDataSigner` returns an instance of the `alpacaRestoreDataSigner` struct, which is not exported.

Recommendation: Change the return type to `RestoreDataSigner`.