

# You build, we defend.



**Source Code Audit** 

Phase 1

October 2024



# Phase 1 Incremental Review Source Code Audit

Version: v241029 Prepared for: Babylon October 2024

# **Security Assessment**

- 1. Executive Summary
- 2. Summary of Findings
  - 2.1 Solved issues & recommendations
- 3. Scope
- 4. Assessment
  - 4.1 Security assumptions
- 5. Detailed Findings
  - BP1-013 Unsafe confirmation depths allowed
  - BP1-014 Attacker on finality's provider system can observe passphrase
  - BP1-015 Hardcoded DB credentials increase the risk of leaking passwords

© Coinspect 2024 1 / 18

BP1-016 - Attacker can force usage of paid ordinals API

6. Disclaimer

© Coinspect 2024 2 / 18

### 1. Executive Summary

In September 2024, **Babylon Labs** engaged <u>Coinspect</u> to perform a Source Code Audit of changes made to the Phase 1 components of the Babylon mainnet. The objective of the project was to evaluate the security of the changes in the Babylon Phase 1 systems.

In October 2024, Coinspect presented the review's definitive report after reviewing Babylon's team fixes for the issues described in the report.

Solved	Caution Advised	Resolution Pending
High O	High <b>O</b>	High O
Medium <b>1</b>	Medium <b>O</b>	Medium <b>O</b>
Low 1	Low O	Low
No Risk	No Risk	No Risk O
Total 4	Total	Total <b>O</b>

BP1-016 describes how an attacker can force a Babylon Staking API operator to lose funds by abusing a fallback mechanism in the Ordinals detection API. BP1-013, BPI-014 and BPI-015 all describe possible improvements to the safety of the system under specific scenarios, but are of low risk or informational value.

© Coinspect 2024 3 / 18

## 2. Summary of Findings

This section provides a concise overview of all the findings in the report grouped by remediation status and sorted by estimated total risk.

#### 2.1 Solved issues & recommendations

These issues have been fully fixed or represent recommendations that could improve the long-term security posture of the project.

ld	Title	Risk
BP1-016	Attacker can force usage of paid ordinals API	Medium
BP1-014	Attacker on finality's provider system can observe passphrase	Low
BP1-013	Unsafe confirmation depths allowed	None
BP1-015	Hardcoded DB credentials increase the risk of leaking passwords	None

© Coinspect 2024 4 / 18

### 3. Scope

The engagement was set to last for 10 days and involved the changes made by the Babylon team since <u>Coinspect's last review</u> to 12 repositories, all related to Phase 1 functionality:

- 1. https://github.com/babylonlabs-io/babylon at commit 009cd29a425d85faf8f6dae0a6ecf4e375aa8211 (last reviewed: add420f074751cf53edea5b7a55cca3d34291f5b)
- 2. https://github.com/babylonlabs-io/btc-staker at commit 4530202f31d3b86a4d4742b294a6d05933db5245 (last reviewed: f22ee2fbd207b29fa428f12a228d46819565306a)
- 3. https://github.com/babylonlabs-io/btc-staking-ts at commit e9438565f32267a54fc2033ab87ba69aa43ac474 (last reviewed: 6494df2b9f2c7a80578356659b1d24302e69dda2)
- 4. https://github.com/babylonlabs-io/cli-tools at commit a3c8cd5ccfbd1ef72bc4fc553e50461fbd9b47cb (last reviewed: d3921efd97bed74dbe9a3b8b578ab320e3460a52)
- 5. https://github.com/babylonlabs-io/covenant-signer at commit 878bbfed57cca1d97aad019f536543207b900b52 (last reviewed: 91e4744bbe0bb440344354e380959d8126d9b82b)
- 6. https://github.com/babylonlabs-io/finality-provider at commit fdc18c897d539c687fa141ab5c21614fe981db58 (last reviewed: dbfe3632bb213560af71e0323a45bbccc1d66000)
- 7. https://github.com/babylonlabs-io/networks at commit 023355a0ee7ce2bf006b972458870e22fd9704f2 (last reviewed: -)
- 8. https://github.com/babylonlabs-io/simple-staking at commit 2c2c1a8289873819e473e2ec44cea8c44b7f8cb1 (last reviewed: 9040c942d0b811e880d284a69d8abbca0572614f)
- 9. https://github.com/babylonlabs-io/staking-api-service at commit 6c111f360dcf5afb790e5a1d1685680a06966c46 (last reviewed: 4e6033a0860df23400611bad24ec72934545f374)
- 10. https://github.com/babylonlabs-io/staking-expiry-checker at commit 0311e3adf58110a6ea0505582918ac8321aaa5b6 (last reviewed: c04e2af4b38e363554b4a4b28485d484b837dbe3)
- 11. https://github.com/babylonlabs-io/staking-indexer at commit ce502f5506e7fefc2aa449a10e56b98b70cb5436 (last reviewed: c13b4f0dd1a57f5f327e5fee613bd41e1b923062)
- 12. https://github.com/babylonlabs-io/staking-queue-client at commit 6b9bb1d59a7d6c5c19ab534f705cd7a5d61ebf91 (last reviewed: 3f07eacc102a7ea9861689a4028c825d4a67e854)

Some repositories had further constraints:

© Coinspect 2024 5 / 18

- 1. babylon repository had only its btcstaking package in scope
- createStakingTxCmd, createUnbondingTxCmd and createWithdrawTxCmd are considered test commands and as such out of scope for cli-tools repository.
- 3. btc-staker repository had only the cmd/stakercli/transaction command in scope
- 4. finality-provider repository had only the commands eotsd init, eotsd keys add, eotsd sign-schnorr and eotsd verify-schnorr-sig of the eotsmanager in scope.
- 5. networks repository only had its parameters directory in scope

© Coinspect 2024 6 / 18

#### 4. Assessment

The Babylon Phase 1 protocol is a set of systems designed to allow stakers to safely and easily lock their coins into a Bitcoin script that ensures that the account cannot be slashed except under certain <u>slashable conditions</u>. During the Phase 1, it should be <u>impossible</u> for slashing to occur, as it is impossible for slashing to happen while there is no Babylon chain active. Another condition of the Babylon Phase 1 protocol is that stakers should be able to, at will, unlock their stake and receive it back after some parametrized unbonding time.

Coinspect has already performed a review of the systems involved for the Phase 1. The threat model in this review is then, essentially, the same. The most prominent risks in scope are, in a rough order of potential severity:

- 1. Problems related to the locking-scripts which would allow an adversary to steal coins from stakers
- 2. Problems related to the locking-scripts which would prevent the staker from unbonding
- 3. Problems in the covenant committee servers which allow an adversary to interrupt the covenant-signing process
- 4. Problems in the frontend or CLI UIs which can make stakers make wrong or unsafe decisions

It is important to note the operational risks such as covenant committee private key safety, frontend delivery and other supply-chain risks are not insignificant; but cannot be covered by a source code review. Another risk that needs to be highlighted is the wallet's responsibility to protect the user's signature, private key and their interactions with potentially adversarial websites. The user-facing application in no way handles the user's private keys, that is the wallet's responsibility.

There are two exceptions in the repository list: one is the networks repository; which has not been reviewed before. It is nevertheless a very small script with an extremely narrow threat model, as it only contains a parser for the configuration of the chain. While bugs *could* be present here, they would be apparent immediately. Furthermore, it would be impossible for an attacker to leverage this repository for an attack, except as part of a supply chain attack. This repository was reviewed for conformity with the specification provided in the README.md file of the bbn1/parameters directory.

The other exception is the finality-provider repository, which was not reviewed by Coinspect together with the rest of the Phase 1 components, but it was reviewed as part of another previous engagement. The logic in scope for this review includes only a few commands related to One-Time Extractable Signatures,

© Coinspect 2024 7 / 18

The threat model for this repository includes best practices around key management, as well as the correctness and safety of the signatures and the verification of them; assuming the underlying cryptographic primitives work correctly. The underlying cryptographic protocol was not in scope for this review.

It is worth highlighting that among the changes reviewed are those made to support Cactus Wallet. While reviewers analyzed Babylon integration-code, it was impossible to perform dynamic tests to check how the wallet behaved as the wallet is not open to users and needs a corporate account before any interactions with it are permitted. In any case, the specific wallet operation is outside the scope of this audit and the Assumptions section note that it was assumed that wallet software protects the user's signatures and private keys.

Another feature introduced in the changes is the attempt to detect and prevent UTXOs containing ordinals from being used. While the feature is well-documented as a *best effort* and thus potential issues with correctness not made a priority, Coinspect found a way in which one could abuse the fallback API to cost funds to operators of the API, described in BP1-016.

#### 4.1 Security assumptions

During the review, Coinspect made several assumptions. The exact same assumptions were needed for Coinspect's previous review:

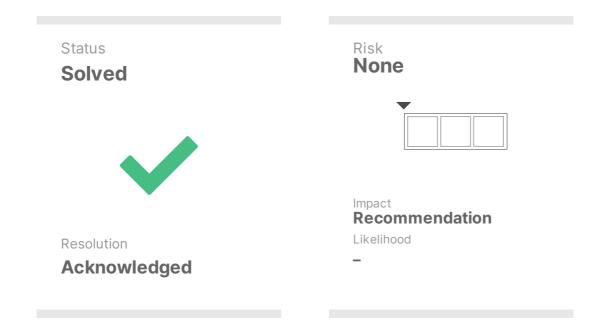
- 1. The Bitcoin network is safe and live.
- 2. A majority of covenant emulation committee members are online and respond to signing requests in a timely manner.
- 3. The provider that hosts the frontend and API components of the web applications is trusted by the user.
- 4. The wallet providers correctly protect the user's signature and private key and don't modify the user's transaction.
- 5. The provider that hosts the components connects the indexer to a Bitcoin node that reports the actual mainchain data.
- 6. The confirmation\_depth parameter is bigger or equal than six and the covenant emulation committee has more than a single member.

© Coinspect 2024 8 / 18

### 5. Detailed Findings

#### **BP1-013**

### **Unsafe confirmation depths allowed**



Location

networks/parameters/parser/ParamsParser.go

#### **Description**

The parameters specification and implementation allow the ConfirmationDepth to be arbitrarily small as long as it is positive. Restricting it to safe values of six or more can remove an assumption (see Assumptions) item from the security reviews, as it the safe values now become an software-enforced invariant of the system.

© Coinspect 2024 9 / 18

#### Recommendation

Restrict the ConfirmationDepth to safe values.

#### **Status**

The Babylon team chose to allow potentially unsafe values as it aids development and testing. This poses no risk as long as care is taken so that the ConfirmationDepth is not unsafe in production-ready configurations.

© Coinspect 2024 10 / 18

#### **BP1-014**

# Attacker on finality's provider system can observe passphrase



#### Location

finality-provider/eotsmanager/cmd/eotsd/daemon/keys.go
btc-staker/cmd/stakercli/main.go

#### **Description**

An attacker positioned on the finality's provider system but without permissions to read the finality's provider configuration or interact with the finality provider system itself can still observe the passphrase used to encrypt and decrypt the Schnorr keys.

This is because the eotsd add and eotsd sign-schnorr commands accept the -passphrase argument via the command line, making it visible for other users via the ps command. The /proc/PID directory from which the ps command reads is readable by all users by default.

The same problem is present in the btc-staker repository but for the Bitcoin RPC password.

© Coinspect 2024 11 / 18

Note that this also leaks the passphrases to the user's .bash\_history or equivalent depending on their shell by default.

#### Recommendation

Provide sensitive data to the system via an interactive prompt, like the sudo command.

#### **Status**

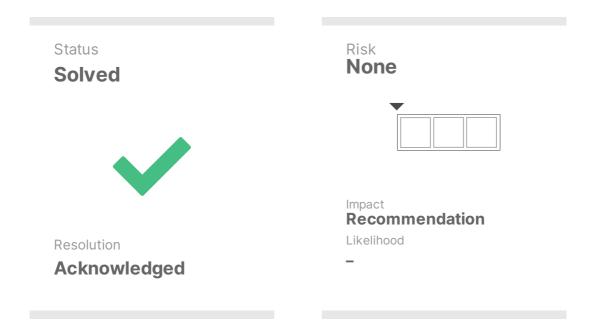
Fixed for finality-provider repository on commit 5118dc565a6c547c29bd7aa45a75919916ae4875. The eotsd add and eotsd sign-schnorr commands now use Cosmo's key management.

The btc-staker repository got the flags removed in commit 4a74320e143a5a8cfeb405ee16fb8d89f2779d09.

© Coinspect 2024 12 / 18

#### **BP1-015**

# Hardcoded DB credentials increase the risk of leaking passwords



#### Location

cli-tools/bin/init-mongo.sh
staking-api-service/bin/init-mongo.sh
staking-expiry-checker/bin/init-mongo.sh

#### **Description**

Hardcoded credentials in the init-mogo.sh script increase the risk of leaking them to untrusted third parties and complicate rotating them in case it is needed. The init-mongos.sh script can be found in the cli-tools, staking-apiservice and expiry-checker repository.

It is worth highlighting that no credentials were leaked to reviewers, as the scripts had placeholder credentials (password: example). Nevertheless, the script was not prepared to read credentials from secure locations.

© Coinspect 2024 13 / 18

#### Recommendation

Store credentials in environmental variables and prepare the scripts to read from them. When possible, use credentials manager services such as <u>AWS</u> <u>Secrets Manager</u> to be able to generate, track and rotate credentials.

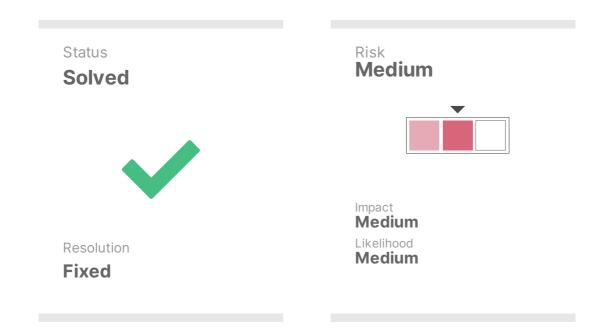
#### **Status**

This issue was considered low risk originally, but the Babylon team stated that these scripts were only used in development and testing. This makes the issue pose no real risk to the system.

© Coinspect 2024 14 / 18

#### **BP1-016**

# Attacker can force usage of paid ordinals API



Location

staking-api-service/internal/clients/ordinals/ordinals.go

#### **Description**

An attacker can force the spending of operator's funds and possibly denyservice to the ordinal-checking service by forcing requests to the freeordinals API to fail, making the system use the pay-to-use fallback option.

To understand the issue, consider that among the changes relevant in the system for this review is a new system that attempts (on a best-effort basis) to detect UTXO that contains ordinals and avoid using them for staking purposes.

To do this, the staking-api-service now implements an HTTP Client (see base\_client.go) and performs two requests to two difference services in the worst case scenario: the first one to a <u>ordinal server</u>, and if that one fails, to a <u>Unisat API</u>. The ordinal server is expected to be ran by the operator, while the Unisat (or any other fallback) is expected to be a payed API.

© Coinspect 2024 15 / 18

The issue is that an attacker can force failure on the free API to force the system to go the secondary, paid API. The attacker can do this by sending a request for a transaction they know does not exist forcing a 404 to be returned by the free API. They can also force a 500 by sending a non-existent txid (see Proof of Concept section).

The impact varies depending on how the operator has paid for the Unisat service. In the worst case scenario, the operator is using the <u>Pay as you go</u> plan, making it possible for the attacker to spend an arbitrary amount of operator's funds.

Note that while the staking-api-service recommends implementing some kind of rate limit for the system, it does not implement any application-level limit, making the issue easier to exploit.

#### Recommendation

Use the fallback API only when the request fails not due to user error, but because the Ordinals server is offline. Because the ordinals API seems to return 500 even when a more appropriate status would be 400 (as the user provided a non existent), the best way to detect an unresponsive Ordinals servers is just to check if the request failed due to the timeout. If the request went through, assume the Ordinals API is working; and do not retry with the paid API.

Alternatively, implement a specific rate-limit for the secondary API, make sure to inform operators of this risk, discourage the use of the Pay as you go Unisat pricing and warn about the potential misuse of the fallback option.

#### **Status**

Fixed in commit c76eadfc98771d30fb4b09ba87e563f4dac9e55e. The secondary, paid-to-use API has been removed; rending this issue impossible to exploit.

#### **Proof of concept**

To confirm the issue, Coinspect leveraged an Ordinal Wallet API provided by Babylon and made a simple Python script to showcase that an attacker can send an arbitrary amount of requests to the server.

```
import requests
srv = "http://localhost:8092/"
end = f"{srv}v1/ordinals/verify-utxos"
```

© Coinspect 2024 16 / 18

```
def send_evil_force_500():
    r = requests.post(end, json={"address":
    "bc1qar0srrr7xfkvy516431ydnw9re59gtzzwf5mdq", "utxos": [{"txid":
    "9f7865756c1e2651a260abebe1b0d1d37b0d73af8b77759fd8ef2060626e25c0",
    "vout": 30}]})
    print(r.json())
    return r

def send_evil_force_404():
    r = requests.post(end, json={"address":
    "bc1qar0srrr7xfkvy516431ydnw9re59gtzzwf5mdq", "utxos": [{"txid":
    "bc4c30829a9564c0d58e6287195622b53ced54a25711d1b86be7cd3a70ef61ed",
    "vout": 13}]})
    print(r.json())
    return r
```

© Coinspect 2024 17 / 18

#### 6. Disclaimer

The contents of this report are provided "as is" without warranty of any kind. Coinspect is not responsible for any consequences of using the information contained herein.

This report represents a point-in-time and time-boxed evaluation conducted within a specific timeframe and scope agreed upon with the client. The assessment's findings and recommendations are based on the information, source code, and systems access provided by the client during the review period.

The assessment's findings should not be considered an exhaustive list of all potential security issues. This report does not cover out-of-scope components that may interact with the analyzed system, nor does it assess the operational security of the organization that developed and deployed the system.

This report does not imply ongoing security monitoring or guaranteeing the current security status of the assessed system. Due to the dynamic nature of information security threats, new vulnerabilities may emerge after the assessment period.

This report should not be considered an endorsement or disapproval of any project or team. It does not provide investment advice and should not be used to make investment decisions.

© Coinspect 2024 18 / 18