Cyberscope

A TAC Security Company

Audit Report

Celestia

November 2025

Repository https://github.com/HEIN-TECH/cobo_public

Commit 21b4fef15fbfb6c1636345c052ea543037d30740

Audited by © cyberscope

Table of Contents

Risk Classification

The criticality of findings in Cyberscope's smart contract audits is determined by evaluating multiple variables. The two primary variables are:

1. **Likelihood of Exploitation**: This considers how easily an attack can be executed, including the economic feasibility for an attacker.
2. **Impact of Exploitation**: This assesses the potential consequences of an attack, particularly in terms of the loss of funds or disruption to the contract's functionality.

Based on these variables, findings are categorized into the following severity levels:

1. **Critical**: Indicates a vulnerability that is both highly likely to be exploited and can result in significant fund loss or severe disruption. Immediate action is required to address these issues.
2. **Medium**: Refers to vulnerabilities that are either less likely to be exploited or would have a moderate impact if exploited. These issues should be addressed in due course to ensure overall contract security.
3. **Minor**: Involves vulnerabilities that are unlikely to be exploited and would have a minor impact. These findings should still be considered for resolution to maintain best practices in security.
4. **Informative**: Points out potential improvements or informational notes that do not pose an immediate risk. Addressing these can enhance the overall quality and robustness of the contract.

Severity	Likelihood / Impact of Exploitation
Critical	Highly Likely / High Impact
Medium	Less Likely / High Impact or Highly Likely/ Lower Impact
Minor / Informative	Unlikely / Low to no Impact

Review

Repository	https://github.com/HEIN-TECH/cobo_public
Commit	21b4fef15fbfb6c1636345c052ea543037d30740

Audit Updates

Initial Audit	03 Nov 2025

Findings Breakdown

Critical	0
Medium	0
Minor / Informative	4

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	0	0	0	0
Minor / Informative	4	0	0	0

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
●	HPKS	Hardcoded Public Key Selection	Unresolved
●	PRAV	Potential Replay Attack Vulnerability	Unresolved
●	PSVM	Potential Signature Verification Misuse	Unresolved
●	SVV	Signature Validation Vulnerability	Unresolved

HPKS - Hardcoded Public Key Selection

Criticality	Minor / Informative
Location	cobo_api_callback_server/app.py#L48
Status	Unresolved

Description

The codebase uses a hard-coded selection of the development (DEV) public key, which increases the risk of misconfiguration in production environments. This approach can lead to accidental deployments that rely on development verification keys, potentially compromising system integrity.

```
pub_keys = {
    "DEV": "a04ea1d5fa8da71f1dcfccf972b9c4eba0a2d8aba1f6da26f49977b08a0d2718",
    "PROD": "8d4a482641adb2a34b726f05827dba9a9653e5857469b8749052bf4458a86729",
}

pubkey = pub_keys["DEV"]
```

Recommendation

Load the target environment and public key from environment variables or a configuration file. In production environments, default to the "PROD" environment to ensure safe and consistent behavior.

PRAV - Potential Replay Attack Vulnerability

Criticality	Minor / Informative
Location	cobo_api_callback_server/app.py#L54
Status	Unresolved

Description

The implementation lacks replay protection, as the `biz_timestamp` included in the signed message is never validated against the current time. This omission allows an attacker to reuse previously signed messages indefinitely, potentially leading to unauthorized or duplicated processing of webhook events.

```
@app.post("/api/webhook")
async def handle_webhook(
    request: Request,
    biz_timestamp: Optional[str] = Header(None),
    biz_resp_signature: Optional[str] = Header(None),
):
    raw_body = await request.body()
    sig_valid = verify_signature(
        pubkey, biz_resp_signature, f"{raw_body.decode('utf8')}|{biz_timestamp}"
    )
    if not sig_valid:
        raise HTTPException(
            status_code=401, detail="Signature verification failed")
    event = WebhookEvent.from_dict(json.loads(raw_body.decode('utf8')))
    logger.info(event)
    logger.info(event.data)
```

Recommendation

To enhance the reliability and integrity of time-sensitive operations, it is recommended to require a `biz_timestamp` parameter in each request and enforce a strict time skew window—such as 5 minutes. This ensures that only requests with timestamps close to the current server time are accepted, helping to prevent replay attacks and mitigate issues caused by clock drift.

PSVM - Potential Signature Verification Misuse

Criticality	Minor / Informative
Location	cobo_api_callback_server/app.py#L121
Status	Unresolved

Description

Potential misuse of Ed25519 verification: `VerifyKey.verify` is called with a pre-hashed double SHA-256 and keyword `smessage`, which is not the standard PyNaCl API for detached signatures. This may fail verification or create false assumptions. Even if detached verify is used, Ed25519 generally signs the message, not a pre-hash, unless using Ed25519ph.

```
def verify_signature(public_key, signature, message):
    vk = VerifyKey(key=bytes.fromhex(public_key))
    sha256_hash = hashlib.sha256(hashlib.sha256(
        message.encode()).digest()).digest()
    try:
        vk.verify(signature=bytes.fromhex(signature), smessage=sha256_hash)
        return True
    except BadSignatureError:
        return False
```

Recommendation

The team is recommended to verify the exact signing scheme required by the upstream. For detached signatures with PyNaCl, use `vk.verify(message_bytes, signature_bytes)`. Only pre-hash if the upstream uses Ed25519ph. Add strict header presence checks and replay protection based on `biz_timestamp`.

SVV - Signature Validation Vulnerability

Criticality	Minor / Informative
Location	cobo_api_callback_server/app.py#L75
Status	Unresolved

Description

The signature is not validated before the request is parsed and logged. The code decodes and parses the request body into a `Transaction` object and logs it prior to verifying the signature. This allows potentially malicious or malformed data to be processed and recorded, leading to unnecessary resource usage and possible log pollution.

```
raw_body = await request.body()
sig_valid = verify_signature(
    pubkey, biz_resp_signature, f"{raw_body.decode('utf8')}|{biz_timestamp}"
)
tx = Transaction.from_dict(json.loads(raw_body.decode('utf8')))
logger.info(tx)
```

Recommendation

The team is advised to:

- Validate request headers and the signature before parsing the JSON body or processing the request.
- Immediately reject requests with missing or malformed headers using appropriate HTTP status codes such as 400 (Bad Request) or 401 (Unauthorized).
- Avoid logging untrusted or unverified payloads to prevent potential exposure of malicious or sensitive data.

Summary

Celestia implements a backend mechanism. This audit investigates security issues, business logic concerns, and potential improvements.

Disclaimer

The information provided in this report does not constitute investment, financial or trading advice and you should not treat any of the document's content as such. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes nor may copies be delivered to any other person other than the Company without Cyberscope's prior written consent. This report is not nor should be considered an "endorsement" or "disapproval" of any particular project or team. This report is not nor should be regarded as an indication of the economics or value of any "product" or "asset" created by any team or project that contracts Cyberscope to perform a security assessment. This document does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors' business, business model or legal compliance. This report should not be used in any way to make decisions around investment or involvement with any particular project. This report represents an extensive assessment process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk Cyberscope's position is that each company and individual are responsible for their own due diligence and continuous security Cyberscope's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies and in no way claims any guarantee of security or functionality of the technology we agree to analyze. The assessment services provided by Cyberscope are subject to dependencies and are under continuing development. You agree that your access and/or use including but not limited to any services reports and materials will be at your sole risk on an as-is where-is and as-available basis Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives false negatives and other unpredictable results. The services may access and depend upon multiple layers of third parties.

About Cyberscope

Cyberscope is a TAC blockchain cybersecurity company that was founded with the vision to make web3.0 a safer place for investors and developers. Since its launch, it has worked with thousands of projects and is estimated to have secured tens of millions of investors' funds.

Cyberscope is one of the leading smart contract audit firms in the crypto space and has built a high-profile network of clients and partners.

A **TAC Security** Company

The Cyberscope team

cyberscope.io