// Security Assessment

06.19.2025 - 06.27.2025

BITSX API
Coredao

HALBORN

BITSX API - Coredao

Prepared by: HALBORN

Last Updated 06/30/2025

Date of Engagement: June 19th, 2025 - June 27th, 2025

Summary

15% OF ALL REPORTED FINDINGS HAVE BEEN ADDRESSED

ALL FINDINGS	CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
13	0	1	7	4	1

TABLE OF CONTENTS

1. Summary

2. Introduction

`CoreDAO` engaged Halborn to conduct a security assessment of the repository of BITS API beginning on *2025-06-19* and ending on *2025-06-27.* The security assessment was scoped to the assets provided to the Halborn team.

3. Assessment Summary

The team at Halborn was provided one week for the engagement and assigned a full-time security engineer to verify the security of the source code of the repository. The security engineer is a penetration testing expert with advanced knowledge in web, recon, discovery & infrastructure penetration testing and blockchain protocols.

The goals for this assessment are:

- Improve the security of the API by testing it as white-box approach

- Identify potential security issues that could be affecting the BITS API

The audited scope included the following repositories:

- https://github.com/Shapeshifter-Technologies/bits-api/tree/8a8e56c81d8f679f6e3aa6a1b76bff8917e4b0f7

Additionally, the following live staging environment was provided in order to support the security audit:

- https://staging.bits.financial (which calls the staging BITS API https://api-bits-testnet.devv.bitsapi.info)

However, due to the registration process not being available at the time of the audit, many of the authenticated functionality could not be tested in depth in the live staging environment. Source code review however was performed over all the repository.

In summary, Halborn identified `one high`, `six medium`, `four low` and `one informational` security issues.

Several possible race condition vulnerabilities were found, where concurrent operations may lead to unpredictable or unintended behavior. However for the most critical one the team indicates that the code is no longer being used.

Several lack of checks on sensitive environment variables was found. The source code was found using outdated and vulnerable dependencies, potentially exposing it to known public exploits. Some dependencies were found to be unpinned as well. Additional issues include possible denial-of-service through resource exhaustion or blocking synchronous calls, exposure of secrets in Git history, unauthenticated access to sensitive API endpoints, and missing key security headers.

Other issues such as TLS misconfigurations, floating point precision errors, and insufficient backend-side validation of business logic were also identified. Additionally, the exposure of Swagger/OpenAPI documentation could provide attackers with insight into the API structure.

It is recommended to resolve all the security issues listed in the document to improve the security health of the application and its underlying infrastructure.

4. Test Approach And Methodology

Halborn performed a combination of manual and automated security testing to balance efficiency, timeliness, practicality, and accuracy in regard to the scope of this assessment. While manual testing is recommended to uncover flaws in logic, process and implementation; automated testing techniques help enhance coverage of the code and can quickly identify items that do not follow security best practices.

Several tests were carried out during the assessment; including, but not limited to:

- Vulnerable or outdated software or dependencies

- Access Handling

- Source code audit

- Sensitive information disclosure

- Application Logic Flaws

- Identify other potential vulnerabilities that may pose a risk to `CoreDAO`

5. RISK METHODOLOGY

Every vulnerability and issue observed by Halborn is ranked based on **two sets** of **Metrics** and a **Severity Coefficient**. This system is inspired by the industry standard Common Vulnerability Scoring System.

The two **Metric sets** are: **Exploitability** and **Impact**. **Exploitability** captures the ease and technical means by which vulnerabilities can be exploited and **Impact** describes the consequences of a successful exploit.

The **Severity Coefficients** is designed to further refine the accuracy of the ranking with two factors: **Reversibility** and **Scope**. These capture the impact of the vulnerability on the environment as well as the number of users and smart contracts affected.

The final score is a value between 0-10 rounded up to 1 decimal place and 10 corresponding to the highest security risk. This provides an objective and accurate rating of the severity of security vulnerabilities in smart contracts.

The system is designed to assist in identifying and prioritizing vulnerabilities based on their level of risk to address the most critical issues in a timely manner.

5.1 EXPLOITABILITY

ATTACK ORIGIN (AO):

Captures whether the attack requires compromising a specific account.

ATTACK COST (AC):

Captures the cost of exploiting the vulnerability incurred by the attacker relative to sending a single transaction on the relevant blockchain. Includes but is not limited to financial and computational cost.

ATTACK COMPLEXITY (AX):

Describes the conditions beyond the attacker's control that must exist in order to exploit the vulnerability. Includes but is not limited to macro situation, available third-party liquidity and regulatory challenges.

METRICS:

EXPLOITABILITY METRIC (M_E)	METRIC VALUE	NUMERICAL VALUE
Attack Origin (AO)	Arbitrary (AO:A) Specific (AO:S)	1 0.2

EXPLOITABILITY METRIC (M_E)	METRIC VALUE	NUMERICAL VALUE
Attack Cost (AC)	Low (AC:L) Medium (AC:M) High (AC:H)	1 0.67 0.33
Attack Complexity (AX)	Low (AX:L) Medium (AX:M) High (AX:H)	1 0.67 0.33

Exploitability E is calculated using the following formula:

$$E = \prod m_e$$

5.2 IMPACT

CONFIDENTIALITY (C):

Measures the impact to the confidentiality of the information resources managed by the contract due to a successfully exploited vulnerability. Confidentiality refers to limiting access to authorized users only.

INTEGRITY (I):

Measures the impact to integrity of a successfully exploited vulnerability. Integrity refers to the trustworthiness and veracity of data stored and/or processed on-chain. Integrity impact directly affecting Deposit or Yield records is excluded.

AVAILABILITY (A):

Measures the impact to the availability of the impacted component resulting from a successfully exploited vulnerability. This metric refers to smart contract features and functionality, not state. Availability impact directly affecting Deposit or Yield is excluded.

DEPOSIT (D):

Measures the impact to the deposits made to the contract by either users or owners.

YIELD (Y):

Measures the impact to the yield generated by the contract for either users or owners.

METRICS:

IMPACT METRIC (M_I)	METRIC VALUE	NUMERICAL VALUE
Confidentiality (C)	None (I:N) Low (I:L) Medium (I:M) High (I:H) Critical (I:C)	0 0.25 0.5 0.75 1
Integrity (I)	None (I:N) Low (I:L) Medium (I:M) High (I:H) Critical (I:C)	0 0.25 0.5 0.75 1
Availability (A)	None (A:N) Low (A:L) Medium (A:M) High (A:H) Critical (A:C)	0 0.25 0.5 0.75 1
Deposit (D)	None (D:N) Low (D:L) Medium (D:M) High (D:H) Critical (D:C)	0 0.25 0.5 0.75 1
Yield (Y)	None (Y:N) Low (Y:L) Medium (Y:M) High (Y:H) Critical (Y:C)	0 0.25 0.5 0.75 1

Impact I is calculated using the following formula:

$$I = max(m_I) + \frac{\sum m_I - max(m_I)}{4}$$

5.3 SEVERITY COEFFICIENT

REVERSIBILITY (R):

Describes the share of the exploited vulnerability effects that can be reversed. For upgradeable contracts, assume the contract private key is available.

SCOPE (S):

Captures whether a vulnerability in one vulnerable contract impacts resources in other contracts.

METRICS:

SEVERITY COEFFICIENT (C)	COEFFICIENT VALUE	NUMERICAL VALUE
Reversibility (r)	None (R:N) Partial (R:P) Full (R:F)	1 0.5 0.25
Scope (s)	Changed (S:C) Unchanged (S:U)	1.25 1

Severity Coefficient C is obtained by the following product:

$$C = rs$$

The Vulnerability Severity Score S is obtained by:

$$S = min(10, EIC * 10)$$

The score is rounded up to 1 decimal places.

SEVERITY	SCORE VALUE RANGE
Critical	9 - 10
High	7 - 8.9
Medium	4.5 - 6.9
Low	2 - 4.4

SEVERITY	SCORE VALUE RANGE
Informational	0 - 1.9

6. SCOPE

Out-of-Scope: New features/implementations after the remediation commit IDs.

7. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	1	7	4	1

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
HAL-01 - POSSIBLE RACE CONDITION	HIGH	PARTIALLY SOLVED - 06/27/2025
HAL-02 - NO CHECKS ON SENSITIVE ENVIRONMENT VARIABLES	MEDIUM	PENDING
HAL-03 - OUTDATED AND VULNERABLE DEPENDENCIES	MEDIUM	PENDING
HAL-04 - POSSIBLE DENIAL-OF-SERVICE DUE TO RESOURCE EXHAUSTION	MEDIUM	PENDING
HAL-05 - POSSIBLE DENIAL-OF-SERVICE DUE TO BLOCKING SYNCHRONOUS CALL	MEDIUM	PENDING
HAL-06 - EXPOSURE OF SECRETS IN GIT HISTORY	MEDIUM	PENDING

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
HAL-07 - UNAUTHENTICATED SENSITIVE API ENDPOINTS	MEDIUM	SOLVED - 06/27/2025
HAL-08 - MISSING IMPORTANT SECURITY HEADERS	MEDIUM	PENDING
HAL-09 - DEPENDENCIES SHOULD BE PINNED TO EXACT VERSIONS	LOW	PENDING
HAL-10 - TLS MISCONFIGURATION	LOW	PENDING
HAL-11 - FLOATING POINT PRECISION ERRORS	LOW	PENDING
HAL-12 - INSUFFICIENT BACKEND-SIDE BUSINESS LOGIC VALIDATION	LOW	PENDING
HAL-13 - EXPOSED SWAGGER / OPENAPI	INFORMATIONAL	PENDING

8. FINDINGS & TECH DETAILS

8.1 (HAL-01) POSSIBLE RACE CONDITION

// HIGH

Description

When multiple processes or users access and modify shared data concurrently without proper synchronization this can lead to unpredictable or inconsistent outcomes. A race condition can be exploited to bypass business rules, perform duplicate actions (like double spending), or corrupt data.

Proof of Concept

Check `app\services\core_service.py`:

```
async def process_redemption(request_id: int):
    """
    Processes a redemption request.

    Args:
        request_id (int): The ID of the redemption request.
    """
    try:
        async for db in get_db():
            logger.info(f"Processing redemption for request ID: {request_id}")

            # Fetch the redemption request
            result = await db.execute(
                select(RedemptionRequestDBModel).where(RedemptionRequestDBModel.id == request_id)
            )
            redemption_request = result.scalars().first()
            if not redemption_request:
                raise ValueError(f"Redemption request {request_id} not found.")

            if redemption_request.status != "initiated": # TODO ENUM
                logger.info(f"Skipping request {request_id}; status is not 'initiated'.")
                return

            # Fetch the BTC/BITS rate
            ## TODO: FIX RATE FUNCTION
            # rate_data = await get_rate_data(
            #     vault_id=VAULT_ID, db=db
            # )
            rate_data =  {
                "btcReserve": 1,
                "circulatingSupply": 1,
                "bitsToBtc":  1,
                "btcToBits": 1,
                }
            logger.info(f"Rate data: {rate_data}")

            # Burn equivalent BITS tokens
            logger.info(f"Burning equivalent BITS tokens for request {request_id}.")
            burn_tx = await burn_bits_tokens(redemption_request, rate_data)
            logger.info(f"Burn transaction receipt: {burn_tx}")

            # Update the redemption request status

            request =await update_redemption_request(
                request_id=request_id,
                status=RedemptionStatus.COMPLETED,
                transaction_hash=burn_tx,
                db=db,
            )

            logger.info(f"Redemption request {request_id} completed successfully.")

            # TODO DOUBLE CHECK CIRCUIT AGAIN WHEN HAVE BTC, CURRENTLY PLACEHOLDER CODE

...
```

If an attacker can concurrently call the GET endpoint `/claim-redemption` defined in `app\api\user\user.py`, so that two or more workers observe the same redemption request in the 'initiated' state, each one could proceed to burn BITS tokens and mark the request as completed, possibly resulting in over-burning of tokens.

Check `app\services\bits_deposit_orchestrator.py`:

```
        async def _handle_deposit_event(
            self,
            db: AsyncSession,
            user: str,
            amount: int,
            deposit_id: int,
            deposit_block: int,
            transaction_hash: Optional[str] = None
         ) -> None:
            """
            Handle BITS Deposited event by recording transaction.

            Args:
                db: Database session
                user: User address
                amount: Deposit amount in BITS (smallest unit)
                deposit_id: Unique deposit identifier
                deposit_block: Block number when the deposit was made
                transaction_hash: Hash of the transaction
            """
            try:
                logger.info("BITS Deposit Event Detected:")
                logger.info(f"  User: {user}")
                logger.info(f"  Amount: {amount}")
                logger.info(f"  Deposit ID: {deposit_id}")
                logger.info(f"  Deposit Block: {deposit_block}")
                logger.info(f"  Transaction Hash: {transaction_hash}")
                # Check if this transaction has already been processed
                if await BaseOrchestrator.check_transaction_exists(db, transaction_hash):
                    logger.info(f"Transaction {transaction_hash} already processed, skipping")
                    return
                # Add deposit transaction to the transactions table
                stmt = insert(TransactionDBModel).values(
                    wallet_address=Web3.to_checksum_address(user),  # Ensure checksummed address
                    transaction_type=TransactionType.DEPOSIT.value,
                    asset_type=AssetType.BITS.value,
                    amount=amount,  # Already in smallest unit (wei)
                    decimals=18,  # BITS has 18 decimals
                    status="completed",
                    transaction_hash=transaction_hash,
                    created_at=datetime.utcnow(),
                    updated_at=datetime.utcnow(),
                ).returning(TransactionDBModel.id)
                # Execute the statement and fetch the inserted transaction ID
                result = await db.execute(stmt)
                transaction_id = result.scalar_one()
                logger.info(f"BITS deposit transaction successfully recorded with ID: {transaction_id}")
                await db.commit()

    ...
```

The deposit orchestrator first performs `check_transaction_exists` to verify whether a given transaction hash has already been processed and, in a separate query, inserts the new row. Because the SELECT and INSERT are not wrapped in the same database transaction, two concurrent orchestrator instances could both observe 'transaction does not exist' and then simultaneously insert a row. Both rows will subsequently be committed.

Check `app\services\bits_deposit_orchestrator.py`:

```
    async def _handle_minting_event(self, amount: int) -> None:
        """
        Handle BITS minting by updating the GlobalConfig totals.

        Args:
            amount: Amount minted in wei
        """
        try:
            db = await self.get_db_session()

            # Get or create the GlobalConfig singleton
            global_config = await GlobalConfigDBModel.get_instance(db)

            # Update both total minted and minted since last sweep
            new_total = int(global_config.minted_bits_amount) + amount
            new_since_sweep = int(global_config.minted_bits_amount_since_last_sweep) + amount

            await global_config.update_minted_bits(db, new_total)
            await global_config.update_minted_bits_since_last_sweep(db, new_since_sweep)


...

    async def _handle_burning_event(self, amount: int) -> None:
        """
        Handle BITS burning by updating the GlobalConfig totals.

        Args:
            amount: Amount burned in wei
        """
        try:
            db = await self.get_db_session()

            # Get or create the GlobalConfig singleton
            global_config = await GlobalConfigDBModel.get_instance(db)

            # Subtract from both total minted and minted since last sweep
            new_total = max(0, int(global_config.minted_bits_amount) - amount)
            new_since_sweep = max(0, int(global_config.minted_bits_amount_since_last_sweep) - amount)

            await global_config.update_minted_bits(db, new_total)
            await global_config.update_minted_bits_since_last_sweep(db, new_since_sweep)

...
```

These event handlers can be called concurrently and then the global state of the counters of minted BITS could be updated incorrectly.

Score

CVSS:3.1/AV:N/AC:H/PR:N/UI:N/S:U/C:H/I:H/A:N (7.4)

Recommendation

Implement proper synchronization mechanisms, such as locks, atomic operations, or transactional controls, to ensure that shared resources are accessed and modified safely in concurrent environments.
Introduce an atomic state transition using a DB transaction + `SELECT ... FOR UPDATE`, or update the row to an 'in-progress' status in the same database transaction before calling any external side-effects.

Remediation Comment

PARTIALLY SOLVED: The team indicate that the function `process_redemption()` is disabled (controlled by the `settings.V0_API` variable).

8.2 (HAL-02) NO CHECKS ON SENSITIVE ENVIRONMENT VARIABLES

// MEDIUM

Description

This vulnerability occurs when an application assumes critical environment variables are set without verifying their validity. If these variables are missing or improperly configured, it can lead to unexpected behavior or security bypasses. Proper checks should be implemented to ensure required environment variables are set and correctly formatted.

Proof of Concept

Check `app\config.py`:

```
# Security and Authentication
    JWT_SECRET_KEY = os.getenv("JWT_SECRET_KEY", os.getenv("JTW_SECRET_KEY"))  # Support both for backwa
    USER_AUTH_TOKEN_EXPIRATION_TIME = 30  # In mins
    ALGORITHM = "HS256"
    OAUTH2_SECRET_KEY = os.getenv("OAUTH2_SECRET_KEY")
```

The value of JWT_SECRET_KEY is taken directly from two environment variables without any checks or fallback. If both JWT_SECRET_KEY and JTW_SECRET_KEY are unset in production, the result is `None`. This could allow attackers to sign valid JWT using empty secrets.

Check `app\config.py`:

```
    PLATFORM_ADMIN_ADDRESS = os.getenv("PLATFORM_ADMIN_ADDRESS")
    PLATFORM_ADMIN_PK = os.getenv("PLATFORM_ADMIN_PK")
    BITS_ADMINS = os.getenv("BITS_ADMINS", "").split(",") if os.getenv("BITS_ADMINS") else []
```

The value of PLATFORM_ADMIN_PK is taken directly from two environment variables without any checks or fallback.

Check `app\config.py`:

```
    # API Configuration
    GENERAL_API_RATE_LIMIT = os.getenv("GENERAL_API_RATE_LIMIT")
```

The value of GENERAL_API_RATE_LIMIT is taken directly from two environment variables without any checks or fallback.

Score

CVSS:3.1/AV:L/AC:H/PR:L/UI:N/S:U/C:H/I:H/A:N (6.3)

Recommendation

Implement strict checks at application startup to verify that all required sensitive environment variables are set and contain valid values. Fail if any are missing or malformed.

8.3 (HAL-03) OUTDATED AND VULNERABLE DEPENDENCIES // MEDIUM

Description

The scoped repository uses multiple third-party dependencies. Using vulnerable third-party libraries can result in security vulnerabilities in the project that can be exploited by attackers. This can result in data breaches, theft of sensitive information, and other security issues. However, some of them were affected by public-known vulnerabilities that may pose a risk to the global application security level.

Proof of Concept

Observe the public vulnerabilities of the third-party dependencies of the packages in the repository **bits-api**. Some of the packages presented third-party dependencies with public vulnerabilities disclosed, for example:

- https://github.com/Shapeshifter-Technologies/bits-api/blob/8a8e56c81d8f679f6e3aa6a1b76bff8917e4b0f7/requirements.txt

```
Tested 114 dependencies for known issues, found 10 issues, 42 vulnerable paths.


Issues to fix by upgrading dependencies:

  Upgrade cryptography@43.0.3 to cryptography@44.0.1 to fix
  ✗ Missing Report of Error Condition [Medium Severity][https://security.snyk.io/vuln/SNYK-PYTHON-CRYPTO
    introduced by cryptography@43.0.3 and 2 other path(s)

  Upgrade h11@0.14.0 to h11@0.16.0 to fix
  ✗ HTTP Request Smuggling (new) [Critical Severity][https://security.snyk.io/vuln/SNYK-PYTHON-H11-10293
    introduced by h11@0.14.0 and 2 other path(s)

  Upgrade requests@2.32.3 to requests@2.32.4 to fix
  ✗ Insertion of Sensitive Information Into Sent Data (new) [Medium Severity][https://security.snyk.io/v
    introduced by requests@2.32.3 and 3 other path(s)

  Upgrade tornado@6.4.1 to tornado@6.5 to fix
  ✗ Regular Expression Denial of Service (ReDoS) [Medium Severity][https://security.snyk.io/vuln/SNYK-PY
    introduced by tornado@6.4.1 and 1 other path(s)
  ✗ Allocation of Resources Without Limits or Throttling [High Severity][https://security.snyk.io/vuln/S
    introduced by tornado@6.4.1 and 1 other path(s)

  Upgrade urllib3@2.0.7 to urllib3@2.5.0 to fix
  ✗ Open Redirect (new) [Medium Severity][https://security.snyk.io/vuln/SNYK-PYTHON-URLLIB3-10390193] in
    introduced by urllib3@2.0.7 and 7 other path(s)
  ✗ Open Redirect (new) [Medium Severity][https://security.snyk.io/vuln/SNYK-PYTHON-URLLIB3-10390194] in
    introduced by urllib3@2.0.7 and 7 other path(s)
  ✗ Improper Removal of Sensitive Information Before Storage or Transfer [Medium Severity][https://secur
    introduced by urllib3@2.0.7 and 7 other path(s)


Issues with no direct upgrade or patch:
  ✗ Timing Attack [High Severity][https://security.snyk.io/vuln/SNYK-PYTHON-ECDSA-6184115] in ecdsa@0.19
    introduced by ecdsa@0.19.1 and 1 other path(s)
  No upgrade or patch available
  ✗ Missing Encryption of Sensitive Data [High Severity][https://security.snyk.io/vuln/SNYK-PYTHON-ECDSA
    introduced by ecdsa@0.19.1 and 1 other path(s)
  No upgrade or patch available
```

Score

CVSS:3.1/AV:N/AC:H/PR:N/UI:R/S:C/C:L/I:L/A:L (5.8)

Recommendation

Update all affected packages to its latest version.
It is strongly recommended to perform an automated analysis of the dependencies from the birth of the project and if they contain any security issues. Developers should be aware of this and apply any necessary mitigation measures to protect the affected application.

8.4 (HAL-04) POSSIBLE DENIAL-OF-SERVICE DUE TO RESOURCE EXHAUSTION

// MEDIUM

Description

If size limits are not enforced on lists and other parameters, it is possible to insert a huge number of elements in those items possibly provoking a Denial-of-Service in the backend side.

Proof of Concept

Check `app\services\core_service.py` :

```
@router.get("/earn-options", response_model=list[EarnOption | Strategy])
async def get_earn_options(
    evm_wallet: Optional[str] = Query(
        None, description="EVM wallet address to filter options"
    ),
    db: AsyncSession = Depends(get_db),
):
    """
    Get available earn options for a user.

    If evm_wallet is provided, it will include deposit transactions for that wallet.
    """
    try:
        # Create a list to store all earn options
        earn_options = []

        # Get current BITS to BTC rate for BITS deposits
        rate_data = await get_rate_data(db)
        bits_to_btc_rate = rate_data["bitsToBtc"]

        if evm_wallet:

            pending_redemptions = await get_list_of_pending_redemptions(evm_wallet, db)
            logger.info(f"Found {len(pending_redemptions)} pending redemptions for {evm_wallet}")
            # Add pending redemption options based on transactions
            for tx in pending_redemptions:
                earn_options.append(prepare_strategy(tx))

            completed_bits_deposits = await get_list_of_bits_complete_deposits(evm_wallet, db)
            logger.info(f"Found {len(completed_bits_deposits)} completed bits deposit transactions for {
            # Add BITS deposit options based on transactions
            for tx in completed_bits_deposits:
                earn_options.append(prepare_strategy(tx))


...
```

The GET `/earn-options` endpoint queries all matching pending or completed transactions without limits or pagination. An attacker controlling a wallet with a large number of transactions could force the API to load huge result sets into memory and iterate over them, exhausting CPU and memory.

Check the usage of the `requests` library in:

- `app\services\exchange_rate_service.py`
- `app\utils\kyc_aml.py`

- `app\utils\notify.py`

```
    @staticmethod
    async def fetch_price_from_coingecko(ticker_name: str) -> Optional[float]:
        """
        Fetch the current price for a ticker from CoinGecko.

        Args:
            ticker_name: The ticker symbol (e.g., "bitcoin-usd")

        Returns:
            The current price as a float, or None if fetch fails
        """
        try:
            # Parse the ticker name (e.g., "bitcoin-usd" -> ids=bitcoin, vs_currencies=usd)
            parts = ticker_name.lower().split('-')
            if len(parts) != 2:
                logger.error(f"Invalid ticker format: {ticker_name}. Expected format: 'coin-currency' (e
                return None

            coin_id, vs_currency = parts

            # Construct the CoinGecko API URL
            url = f"https://api.coingecko.com/api/v3/simple/price?ids={coin_id}&vs_currencies={vs_curren

            # Run the blocking requests call in a thread pool
            loop = asyncio.get_event_loop()
            response = await loop.run_in_executor(None, requests.get, url)

            response.raise_for_status()
            data = response.json()


...
```

The function performs a HTTP request using `requests` without specifying any timeout. These calls can hang indefinitely, exhausting executor threads and possibly leading to a denial-of-service.

Score

CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:L (5.3)

Recommendation

To mitigate this vulnerability, a multifaceted approach is advised:
- Impose strict size limits on all incoming parameters and payloads, including string lengths and array elements.
- Implement load balancing and robust DoS protection measures like Cloudflare to help distribute traffic evenly and defend against attack vectors.

8.5 (HAL-05) POSSIBLE DENIAL-OF-SERVICE DUE TO BLOCKING SYNCHRONOUS CALL

// MEDIUM

Description

Blocking synchronous calls within asynchronous functions can block the event loop and prevent other tasks from executing. Under high load, this can lead to denial-of-service.

Proof of Concept

During the security audit, many instances of synchronous calls of `Web3.HTTPProvider` within an async function were found. Some examples are listed here:

- `app\utils\blockchain_utils.py`

```
async def execute_blockchain_transaction(
    rpc_url: str,
    contract_address: str,
    contract_abi: list,
    function_name: str,
    function_args: list,
    from_address: str,
    private_key: str,
    gas_limit: Optional[int] = None,
    gas_price: Optional[int] = None,
    value: int = 0,
    network_name: str = "blockchain"
) -> str:

...

    try:
        # Connect to the network
        web3 = Web3(Web3.HTTPProvider(rpc_url))

        # Verify connection
        if not web3.is_connected():
            raise BlockchainConnectionError(f"Failed to connect to {network_name} network")

        logger.info(f"Connected to {network_name} network")

...
```

- `app\api\endpoints\earn_options.py`

```
async def get_bits_balance_for_wallet(wallet_address: str) -> tuple[int, int]:
    """
    Get BITS token balance for a wallet on Core network.
    Returns tuple of (balance_in_wei, decimals)
    """
    decimals = get_asset_decimals(AssetType.BITS.value)
    try:
        # Connect to Core network
        web3 = Web3(Web3.HTTPProvider(settings.CORE_RPC_URL))
        if not web3.is_connected():
            logger.error("Failed to connect to Core network")
            return (0, decimals)  # Return 0 balance if connection fails
```

```
...
```

Score

CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:L (5.3)

Recommendation

Replace blocking synchronous calls within asynchronous functions with their non-blocking asynchronous equivalents to ensure the event loop remains responsive.
Use an executor blocking web3 calls or migrate to Web3.AsyncHTTPProvider.

8.6 (HAL-06) EXPOSURE OF SECRETS IN GIT HISTORY

// MEDIUM

Description

Sharing code publicly or even privately can unintentionally expose sensitive information if secrets such as API keys, private keys, or environment-specific details are embedded in the codebase. This issue arises because repositories, when cloned or forked, preserve their entire commit history. As a result, any hardcoded secrets present in the repository's history will also be available in all derived repositories. If these secrets are stored in plaintext, they could be exploited by unauthorized individuals, leading to potential misuse of credentials or keys.
During the security assessment, it was identified that the application's Git commit history contained hardcoded API keys, private keys, and other sensitive environment-related details. The presence of these hardcoded secrets poses a significant risk, as any unauthorized access to the repository could lead to the compromise of these secrets. This could further result in unauthorized access to critical infrastructure or data, compromising the security and integrity of the application and its associated systems.

Proof of Concept

- Observe the PostgreSQL password in https://github.com/Shapeshifter-Technologies/bits-api/blob/8a8e56c81d8f679f6e3aa6a1b76bff8917e4b0f7/docker-compose.yml:

```
services:
  web:
    build: .
    container_name: bits_api
    ports:
      - "4000:4000"
    env_file: .env
    environment:
      - DATABASE_URL=postgresql+asyncpg://postgres:pa*****d@db:5432/bits_db
    depends_on:
      - db
      - redis
    volumes:
      - .:/app
      - ./logs:/app/logs
    networks:
      - app_network

  orchestrator:
    build: .
    container_name: bits_orchestrator
    env_file: .env
    environment:
      - DATABASE_URL=postgresql+asyncpg://postgres:pa*****d@db:5432/bits_db
      - PYTHONPATH=/app
    depends_on:
      - db
      - redis
    volumes:
      - .:/app
      - ./logs:/app/logs
    networks:
      - app_network
    command: ["python", "app/orchestrator.py"]
  db:
    image: postgres:13
    container_name: postgres_db
    environment:
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: pa*****d
      POSTGRES_DB: bits_db

...
```

- Observe the private key here: https://github.com/Shapeshifter-Technologies/bits-api/blob/9dde4a8493dae7914d2f7761e8e8af620f5647e1/helm/devv-testnet.values.yaml#L9
- Observe the database credentials and service API key here: https://github.com/Shapeshifter-Technologies/bits-api/blob/e86532a812efc90d21a083f9e26e914a9dd1913c/helm/devv-testnet.values.yaml#L6
- Observe the private key here: https://github.com/Shapeshifter-Technologies/bits-api/blob/84ca9f52139c4ac140760f165b1c9ed11e4a8cf3/fireblocks_secret.key#L1

Note: *An internal team review is required to assess the criticality of the identified secrets and determine if they are used for testing or production purposes. This review will help determine the appropriate response to mitigate potential risks.*

Score

CVSS:3.1/AV:N/AC:H/PR:H/UI:N/S:U/C:H/I:L/A:N (5.0)

Recommendation

To address the identified risks, the following measures would be appropriate:

Remove Hardcoded Secrets: Immediately remove any hardcoded API keys, private keys, or sensitive information from the source code and commit history.

Use Environment Variables: Store sensitive information such as API keys and credentials in environment variables or secure storage solutions like GitHub Secrets and AWS Secrets Manager.

Rotate Compromised Keys: Immediately rotate any exposed or compromised keys to mitigate potential misuse.

8.7 (HAL-07) UNAUTHENTICATED SENSITIVE API ENDPOINTS // MEDIUM

Description

Sensitive API endpoints are accessible without requiring proper authentication. Attackers can exploit these endpoints to access or manipulate protected data or functionality without valid credentials. This undermines access control.

Proof of Concept

Check `app\api\user\user.py`:

```
if settings.V0_API:
    @router.get("/check-redemptions")
    async def check_redemptions(db: AsyncSession = Depends(get_db)):
        """
        Endpoint to manually check and process eligible redemption requests.
        """
        try:
            # Get eligible redemption requests
            eligible_requests = await get_eligible_redemptions(
                db=db, lock_time_seconds=settings.REDEMPTION_LOCK_TIME
            )

            if not eligible_requests:
                return {"message": "No eligible redemption requests found."}

            # Process each eligible request
            for request in eligible_requests:
                logger.info(f"Processing redemption request ID: {request.id}")
                await process_redemption(request.id)

            return {"message": f"Processed {len(eligible_requests)} redemption request(s)."}

        except Exception as e:
            logger.exception("Error checking and processing redemptions.")
            raise HTTPException(status_code=500, detail=f"Error: {str(e)}")

if settings.V0_API:
    @router.get("/claim-redemption")
    async def claim_redemption(redemption_id: int, db: AsyncSession = Depends(get_db)):
        """
        Endpoint to manually check and process eligible redemption requests.
        """
        try:
            await process_redemption(redemption_id)
            return {"message": f"Redemption request {redemption_id} marked as completed."}
        except Exception as e:
            raise HTTPException(status_code=500, detail=f"Error: {str(e)}")
```

If these endpoints are exposed, any external caller can invoke these endpoints and force the backend to execute on-chain redemption transactions. No authentication is required. There are other unauthenticated endpoints in `app\api\user\user.py` but they do not seem to have security implications. However, it is important to always re-check unauthenticated endpoints for possible security implications.

Score

CVSS:3.1/AV:N/AC:H/PR:N/UI:N/S:U/C:L/I:L/A:N (4.8)

Recommendation

Completely remove public exposure or wrap these routes with strict authentication/authorization. Consider moving maintenance tasks to internal scheduled jobs instead of HTTP endpoints.

Remediation Comment

SOLVED: the team indicate that these endpoints are disabled (controlled by the `settings.V0_API` variable).

8.8 (HAL-08) MISSING IMPORTANT SECURITY HEADERS

// MEDIUM

Description

The application lacks several important HTTP security headers, which are instrumental in enhancing the security posture of web applications by instructing browsers on how to behave when handling the site's content. The absence of these headers leaves the application vulnerable to various attack vectors, including Man-in-the-Middle (MITM) attacks and Cross-Site Scripting (XSS).

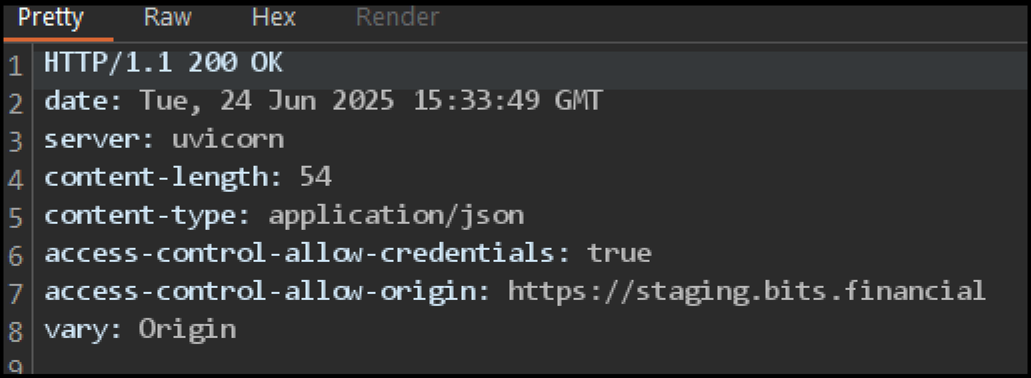
Missing Security Headers:

- **Strict-Transport-Security**: the lack of the HTTP Strict Transport Security (HSTS) header exposes the application to MitM attacks by allowing the use of insecure HTTP connections.
- **Content-Security-Policy (CSP)**: without a CSP header, the application is more susceptible to XSS attacks as it fails to restrict the sources of content that the browser can load.

Proof of Concept

In the images below, it is possible to observe the security headers returned by the servers in the response:

- **https://api-bits-testnet.devv.bitsapi.info:** the server does not return the Strict-Transport-Security not the Content-Security-Policy security headers:



```
HTTP/1.1 200 OK
date: Tue, 24 Jun 2025 15:33:49 GMT
server: uvicorn
content-length: 54
content-type: application/json
access-control-allow-credentials: true
access-control-allow-origin: https://staging.bits.financial
vary: Origin
```

Score

CVSS:3.1/AV:N/AC:H/PR:N/UI:R/S:U/C:L/I:L/A:N (4.2)

Recommendation

It is recommended to implement the missing HTTP security headers with appropriate policies. Implementing these security headers will significantly enhance the security of the application by preventing various web-based attacks and ensuring safer interactions with the user's browser.

References

https://owasp.org/www-project-secure-headers/

8.9 (HAL-09) DEPENDENCIES SHOULD BE PINNED TO EXACT VERSIONS

// LOW

Description

The application contained multiple dependencies that were not pinned to an exact version, but they were set to a supported version. This could potentially allow dependency attacks.

Proof of Concept

Observe the versioning of the third-party dependencies of the packages in the repository **bits-api**. For example:

- https://github.com/Shapeshifter-Technologies/bits-api/blob/8a8e56c81d8f679f6e3aa6a1b76bff8917e4b0f7/requirements.txt

While most dependencies are pinned to exact versions, there are dependecies not pinned:

```
...

websockets==14.0
wrapt==1.16.0
yarl==1.18.3
itsdangerous
sendgrid
base58
bech32
pytest
aiosqlite
sqlalchemy
pytest-asyncio
ecdsa
```

Score

CVSS:3.1/AV:N/AC:H/PR:N/UI:N/S:U/C:N/I:L/A:N (3.7)

Recommendation

The repository dependencies in the `requirements.txt` files should be pinned to exact versions to prevent dependency attacks.

8.10 (HAL-10) TLS MISCONFIGURATION

// LOW

Description

When information is sent between the client and the server, it must be encrypted and protected in order to prevent an attacker from being able to read or modify it. Over the years there have been a large number of cryptographic weaknesses identified in the SSL and TLS protocols, as well as in the ciphers that they use. Additionally, many of the implementations of these protocols have also had serious vulnerabilities.
It should be emphasized that while many of these attacks have been demonstrated in a lab environment, they are not generally considered practical to exploit in the real world, as they require a (usually active) MitM attack, and significant resources.

Proof of Concept

Observe the TLS protocols enabled in the affected servers:

- **https://api-bits-testnet.devv.bitsapi.info:**

```
Testing SSL server api-bits-testnet.devv.bitsapi.info on port 443 using SNI name api-bits-testnet.devv.bitsapi.info

  SSL/TLS Protocols:
SSLv2     disabled
SSLv3     disabled
TLSv1.0   enabled
TLSv1.1   enabled
```

Additionally, observe `app\services\websocket_manager.py` :

```
    async def connect(self) -> AsyncWeb3:
        """
        Establish WebSocket connection with retry logic.

        Returns:
            Configured AsyncWeb3 instance
        """
        retry_count = 0
        backoff = self.initial_backoff

        # Log network details for debugging
        logger.info(f"WebSocketManager attempting to connect to: {self.ws_url}")
        logger.info(f"Max retries: {self.max_retries}, Initial backoff: {self.initial_backoff}s, Max bac

        while self.max_retries == -1 or retry_count < self.max_retries:
            try:
                retry_count += 1
                logger.info(f"Connection attempt {retry_count}/{self.max_retries if self.max_retries !=

                # Ensure the WebSocket URL is properly formatted
                if not self.ws_url.startswith(('ws://', 'wss://')):
                    logger.error(f"Invalid WebSocket URL format: {self.ws_url}. Must start with ws:// or
                    raise ValueError(f"Invalid WebSocket URL format: {self.ws_url}. Must start with ws://
```

The `connect()` method only validates that the URL begins with either "ws://" or "wss://". It does not enforce TLS. Enforce TLS by requiring the URL to begin with "wss://".

Score

CVSS:3.1/AV:A/AC:H/PR:N/UI:R/S:U/C:L/I:L/A:N (3.7)

Recommendation

Disable TLSv1.0 and TLSv1.1 TLS protocols in the affected servers.

References

https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/09-Testing_for_Weak_Cryptography/01-Testing_for_Weak_Transport_Layer_Security

8.11 (HAL-11) FLOATING POINT PRECISION ERRORS // LOW

Description

The application mixes Python floats with Decimal for sensitive financial calculations. Converting large or high-precision Decimal values to float before division and subsequent computations can lead to precision loss.

Proof of Concept

Check `app\utils\pricing.py`:

```
async def get_rate_data(db: AsyncSession) -> dict:
    """
    Fetch BTC/BITS rate based on the vault reserves and total circulating supply.
    """
    total_deposit_amount = await get_total_deposit_amount(db)
    total_redeemed_btc_amount = await get_total_redeemed_btc_amount(db)
    btc_reserve = total_deposit_amount - total_redeemed_btc_amount

    # Connect to the Core chain
    web3_instance = Web3(Web3.HTTPProvider(settings.CORE_RPC_URL))
    contract = web3_instance.eth.contract(
        address=settings.BITS_TOKEN_CONTRACT_ADDRESS, abi=token_abi
    )

    total_supply = contract.functions.totalSupply().call()
    decimals = contract.functions.decimals().call()
    total_supply_decimals = 10 ** decimals

    bits_circulating_supply = float(Decimal(total_supply) / Decimal(total_supply_decimals))
    minted_bits_since_last_sweep = await get_minted_bits_since_last_sweep(db)
    effective_bits = bits_circulating_supply - minted_bits_since_last_sweep


...
```

The conversions between float and Decimal throughout all the file can lose precision.

Score

CVSS:3.1/AV:N/AC:H/PR:N/UI:N/S:U/C:N/I:L/A:N (3.7)

Recommendation

Perform all sensitive financial calculations using Decimal throughout and avoid converting to float. Use Decimal.quantize() with a fixed precision for final outputs. Ensure amounts, rates, and calculations remain in Decimal until serialization for the API response.

8.12 (HAL-12) INSUFFICIENT BACKEND-SIDE BUSINESS LOGIC VALIDATION

// LOW

Description

Missing or inadequate validation of business logic on the backend, particularly for critical operations like transactions, may allow attackers to bypass rules. Robust backend-side validation is essential to enforce business rules and prevent unauthorized actions.

Proof of Concept

Check `app\api\transactions\transactions.py`:

```
@router.post("/redeem-bits", response_model=PostRedeemBTCResponse)
@limiter.limit(settings.GENERAL_API_RATE_LIMIT)
async def redeem_bits(
    request: Request,
    body: RedeemBitsRequest,
    db: AsyncSession = Depends(get_db),
    current_user: UserDBModel = Depends(check_kyc_status),
):
    try:
        # Validate that the provided EVM public key belongs to the authenticated user
        result = await db.execute(
            select(EvmWalletDBModel).where(
                EvmWalletDBModel.evm_wallet_address == body.evm_public_key
            )
        )
        evm_wallet = result.scalar()
        if not evm_wallet:
            raise HTTPException(
                status_code=403,
                detail="The provided EVM public key does not belong to the authenticated user."
            )

        # Check wallet risk if AML is enabled
        if is_aml_enabled():
            await check_wallet_risk(db, body.evm_public_key, "evm")

        # Validate the signature
        message = f"{body.timestamp}:{body.btc_public_key}:{body.transaction_hash}"
        logger.info(f"Message to validate: {message}")

        encoded_message = encode_defunct(text=message)
        recovered_address = Account.recover_message(
            encoded_message,
            signature=body.signature
        )
        logger.info(f"Recovered public key: {recovered_address}")

        if recovered_address.lower() != body.evm_public_key.lower():
            raise HTTPException(
                status_code=400,
                detail="Recovered EVM public key does not match the provided EVM public key. Validat
            )

        # Validate the transaction on the blockchain
        await validate_bits_deposit_transaction_on_chain(
            transaction_hash=body.transaction_hash,
            user_address=body.evm_public_key,
            vault_address=settings.BITS_VAULT_CONTRACT_ADDRESS,
            token_address=settings.BITS_TOKEN_CONTRACT_ADDRESS,
                expected_amount=body.amount,
                provider_url=settings.CORE_RPC_URL,
                vault_abi=vault_abi
            )

            # Add redemption request to DB
            bits_decimals = AmountConverter.get_decimals(AssetType.BITS.value)
            new_request = RedemptionRequestDBModel(
                evm_public_key=body.evm_public_key,
                btc_public_key=body.btc_public_key,
                amount=body.amount,
                decimals=bits_decimals,
                transaction_hash=body.transaction_hash,
                status="pending",
                created_at=datetime.utcnow(),
                updated_at=datetime.utcnow()
            )
            db.add(new_request)
            await db.commit()
            await db.refresh(new_request)
            logger.info("Added request to DB")

...
```

The POST `/redeem-bits` endpoint allows a user supplied "transaction_hash" to be requested to be redeemed without checking whether that hash has already been used in a previous redemption request. Even if this is enforced in the smart contract, it is a good defense-in-depth mechanism to include some previous checks in the offchain logic.
Additionally, the signed message includes a timestamp, however, the code does not enforce freshness of the timestamp or prevent reuse of a valid signature.

Check `app\api\user\user.py`:

```
    @router.post("/associate/evm-wallet")
    async def associate_evm_wallet(
        user_id: str,
        payload: AssociateEVMWalletRequest,
        db: AsyncSession = Depends(get_db),
    ):

        ...

            # Validate wallet uniqueness
            if not await validate_unique_wallet(db, payload.address):
                raise HTTPException(
                    status_code=400,
                    detail="Wallet address is already associated",
                )

            if not validate_one_evm_wallet_per_user(db, user_id):
                raise HTTPException(
                    status_code=400,
                    detail="User already has an EVM wallet associated",
                )
...
```

Due to a mistake, the call to `validate_one_evm_wallet_per_user(db, user_id)` is missing an `await`, so the coroutine is not executed, possibly allowing multiple EVM wallets to be associated per user.

Score

CVSS:3.1/AV:N/AC:H/PR:L/UI:N/S:U/C:N/I:L/A:N (3.1)

Recommendation

Ensure that all essential validations are performed server-side before processing any transaction or sensitive action.

8.13 (HAL-13) EXPOSED SWAGGER / OPENAPI

// INFORMATIONAL

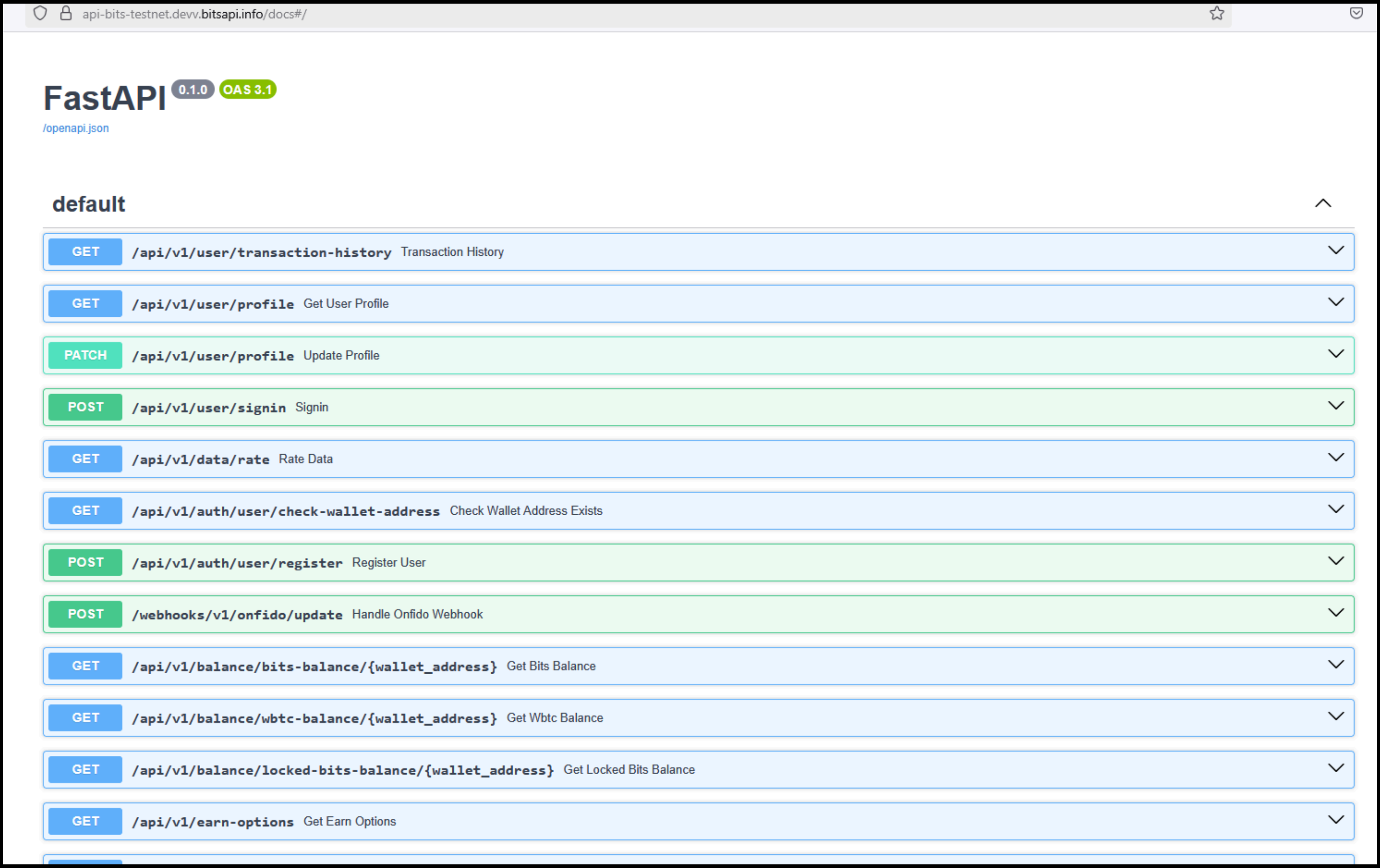
Description

Swagger exposes detailed documentation about your API endpoints, request/response structures, and data models. This information can help an attacker to mount specific attacks against API functionalities described in Swagger.

Proof of Concept

Observe the following links:

- https://api-bits-testnet.devv.bitsapi.info/openapi.json
- https://api-bits-testnet.devv.bitsapi.info/docs


api-bits-testnet.devv.bitsapi.info/docs#/

FastAPI 0.1.0 OAS 3.1

/openapi.json

default

```
GET    /api/v1/user/transaction-history  Transaction History
GET    /api/v1/user/profile  Get User Profile
PATCH  /api/v1/user/profile  Update Profile
POST   /api/v1/user/signin  Signin
GET    /api/v1/data/rate  Rate Data
GET    /api/v1/auth/user/check-wallet-address  Check Wallet Address Exists
POST   /api/v1/auth/user/register  Register User
POST   /webhooks/v1/onfido/update  Handle Onfido Webhook
GET    /api/v1/balance/bits-balance/{wallet_address}  Get Bits Balance
GET    /api/v1/balance/wbtc-balance/{wallet_address}  Get Wbtc Balance
GET    /api/v1/balance/locked-bits-balance/{wallet_address}  Get Locked Bits Balance
GET    /api/v1/earn-options  Get Earn Options
```

Score

CVSS:3.1/AV:A/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:N (0.0)

Recommendation

It is recommended not to expose the Swagger with the API information in production.

Halborn strongly recommends conducting a follow-up assessment of the project either within six months or immediately following any material changes to the codebase, whichever comes first. This approach is crucial for maintaining the project's integrity and addressing potential vulnerabilities introduced by code modifications.