SHERLOCK

Security Review For Zyfai

Collaborative Audit Prepared For: **Zyfai**
Lead Security Expert(s): **defsec**
Date Audited: **November 11 - November 21, 2025**

Introduction

Zyfai is a DeFi agent built to optimize yields using real-time data and automated on-chain actions. Leveraging ERC-7579 Safe modular accounts, for example with session modules, it provides a non-custodial solution for open and personalized financial rebalancing across DeFi.

Zyfai is pushing the boundaries of on-chain capital allocation with a more tailored, efficient, and customizable protocol

Scope

Repository: 0xSulpiride/zyfai-backend

Audited Commit: fb5a6253b5a0ad12a634fdb4324e95e8da1ad2d3

Final Commit: 72dbcecf25cf7809bf8be68d8cabcc2608aaa2b5

Files:

- src/api/admin/admin.controller.ts
- src/api/admin/admin.module.ts
- src/api/admin/admin.service.ts
- src/api/api.module.ts
- src/api/auth/auth.controller.ts
- src/api/auth/auth.module.ts
- src/api/auth/auth.service.ts
- src/api/auth/config/auth.config.ts
- src/api/auth/config/auth-config.type.ts
- src/api/auth/strategies/ethereum-strategy/store/session.ts
- src/api/auth/strategies/ethereum-strategy/strategy.ts
- src/api/auth/strategies/ethereum.strategy.ts
- src/api/auth/types/jwt-payload.type.ts
- src/api/auth/types/jwt-refresh-payload.type.ts
- src/api/customization/customization.controller.ts
- src/api/customization/customization.module.ts
- src/api/customization/customization.service.ts
- src/api/customization/dto/customize-batch.req.dto.ts
- src/api/customization/dto/customize.req.dto.ts

- src/api/customization/entities/customization.entity.ts
- src/api/data/data.controller.ts
- src/api/data/data.module.ts
- src/api/data/data.service.ts
- src/api/erc4626wrapper/erc4626wrapper.controller.ts
- src/api/erc4626wrapper/erc4626wrapper.module.ts
- src/api/erc4626wrapper/erc4626wrapper.service.ts
- src/api/health/health.controller.ts
- src/api/health/health.module.ts
- src/api/history/entities/active-wallet-view.entity.ts
- src/api/history/entities/multi-position-view.entity.ts
- src/api/history/entities/wallet-history-view.entity.ts
- src/api/history/history.controller.ts
- src/api/history/history.module.ts
- src/api/history/history.service.ts
- src/api/price-oracle/constants.ts
- src/api/price-oracle/entities/price-oracle.entity.ts
- src/api/price-oracle/price-oracle.module.ts
- src/api/price-oracle/price-oracle.service.ts
- src/api/price-oracle/sonic.ts
- src/api/protocol/entities/contract.entity.ts
- src/api/protocol/entities/protocol.entity.ts
- src/api/protocol/protocol.controller.ts
- src/api/protocol/protocol.module.ts
- src/api/protocol/protocol.service.ts
- src/api/session-keys/constants.ts
- src/api/session-keys/dto/add-session-key.req.dto.ts
- src/api/session-keys/entities/session-key.entity.ts
- src/api/session-keys/rhinestone.ts
- src/api/session-keys/session-keys.controller.ts
- src/api/session-keys/session-keys.interface.ts

- src/api/session-keys/session-keys.module.ts
- src/api/session-keys/session-keys.repository.ts
- src/api/session-keys/session-keys.service.ts
- src/api/session-keys/session-key-versioning.ts
- src/api/session-keys/utils/index.ts
- src/api/session-keys/utils/toAccount.ts
- src/api/simulate/dto/best-positions.dto.ts
- src/api/simulate/simulate.controller.ts
- src/api/simulate/simulate.module.ts
- src/api/simulate/simulate.service.ts
- src/api/supported-tokens/dto/supported-token.res.dto.ts
- src/api/supported-tokens/entities/supported-token.entity.ts
- src/api/supported-tokens/supported-tokens.controller.ts
- src/api/supported-tokens/supported-tokens.module.ts
- src/api/supported-tokens/supported-tokens.service.ts
- src/api/user/dto/get-position.res.dto.ts
- src/api/user/dto/log-deposit.req.dto.ts
- src/api/user/dto/log-withdraw.req.dto.ts
- src/api/user/dto/partial-withdraw.req.dto.ts
- src/api/user/dto/partial-withdraw.res.dto.ts
- src/api/user/dto/reset-telegram.res.dto.ts
- src/api/user/dto/update-telegram.req.dto.ts
- src/api/user/dto/update-user.req.dto.ts
- src/api/user/dto/user-autoselect.res.dto.ts
- src/api/user/dto/user.res.dto.ts
- src/api/user/entities/chain.entity.ts
- src/api/user/entities/deposits.entity.ts
- src/api/user/entities/protocol-balances.entity.ts
- src/api/user/entities/session.entity.ts
- src/api/user/entities/user.entity.ts
- src/api/user/user.controller.ts

- src/api/user/user.module.ts
- src/api/user/user.repository.ts
- src/api/user/user.service.ts
- src/app.module.ts
- src/background/background.module.ts
- src/background/cron/rebalance/rebalance.service.ts
- src/background/cron/rebalance/rebalance.utils.ts
- src/background/cron/wallet/wallet.service.ts
- src/background/queues/transaction-queue/transaction.processor.ts
- src/background/queues/transaction-queue/transaction-queue.events.ts
- src/background/queues/transaction-queue/transaction-queue.module.ts
- src/background/queues/transaction-queue/transaction-queue.service.ts
- src/common/dto/cursor-pagination/cursor-pagination.dto.ts
- src/common/dto/cursor-pagination/page-options.dto.ts
- src/common/dto/cursor-pagination/paginated.dto.ts
- src/common/dto/error-detail.dto.ts
- src/common/dto/error.dto.ts
- src/common/dto/offset-pagination/offset-pagination.dto.ts
- src/common/dto/offset-pagination/page-options.dto.ts
- src/common/dto/offset-pagination/paginated.dto.ts
- src/common/interfaces/job.interface.ts
- src/common/types/common.type.ts
- src/common/types/types.ts
- src/config/app.config.ts
- src/config/app-config.type.ts
- src/config/config.type.ts
- src/config/rebalance-tiers.config.ts
- src/constants/app.constant.ts
- src/constants/cache.constant.ts
- src/constants/constraint-errors.ts
- src/constants/contract-addresses.constants.ts

- src/constants/entity.enum.ts
- src/constants/error-code.constant.ts
- src/constants/job.constant.ts
- src/constants/networks.constants.ts
- src/constants/rewards.constants.ts
- src/constants/time.constants.ts
- src/constants/transaction.enum.ts
- src/constants/usdc.constants.ts
- src/constants/wallet.constants.ts
- src/core/admin/admin.module.ts
- src/core/admin/admin.service.ts
- src/core/admin/constants.ts
- src/core/admin/utils.ts
- src/core/bridging/bungee/bungee.service.ts
- src/core/bridging/bungee/interface.ts
- src/core/bridging/entities/bridge-log.entity.ts
- src/core/core.module.ts
- src/core/defi/aave/aave.module.ts
- src/core/defi/aave/aave.service.ts
- src/core/defi/aave/v3/vault.ts
- src/core/defi/beets/beets.config.ts
- src/core/defi/beets/beets.module.ts
- src/core/defi/beets/beets.service.ts
- src/core/defi/beets/readme.md
- src/core/defi/beets/strategies/abis.ts
- src/core/defi/beets/strategies/BoostedDollarSuiteNo1.ts
- src/core/defi/beets/strategies/index.ts
- src/core/defi/compoundv3/compoundv3.config.ts
- src/core/defi/compoundv3/compoundv3.module.ts
- src/core/defi/compoundv3/compoundv3.service.ts
- src/core/defi/defi.module.ts

- src/core/defi/defi.service.ts
- src/core/defi/erc20/erc20.service.ts
- src/core/defi/euler/euler.config.ts
- src/core/defi/euler/euler.module.ts
- src/core/defi/euler/euler.service.ts
- src/core/defi/euler/euler.utils.ts
- src/core/defi/euler/strategies/index.ts
- src/core/defi/euler/strategies/partial_withdraw.ts
- src/core/defi/fluid/fluid.config.ts
- src/core/defi/fluid/fluid.module.ts
- src/core/defi/fluid/fluid.service.ts
- src/core/defi/fluid/readme.md
- src/core/defi/harvest-finance/harvest-finance.module.ts
- src/core/defi/harvest-finance/harvest-finance.service.ts
- src/core/defi/harvest-finance/strategies/autopilot_usdc.ts
- src/core/defi/harvest-finance/strategies/index.ts
- src/core/defi/harvest-finance/strategies/interface.ts
- src/core/defi/harvest-finance/strategies/usdc_moonwell.ts
- src/core/defi/harvest-finance/strategies/usdc.ts
- src/core/defi/harvest-finance/strategies/vault.ts
- src/core/defi/index.ts
- src/core/defi/merkl/contracts.ts
- src/core/defi/merkl/merkl.module.ts
- src/core/defi/merkl/merkl.service.ts
- src/core/defi/moonwell/moonwell.module.ts
- src/core/defi/moonwell/moonwell.service.ts
- src/core/defi/moonwell/strategies/index.ts
- src/core/defi/moonwell/strategies/Moonwell_Flagship_USDC.ts
- src/core/defi/moonwell/strategies/partial_withdraw.ts
- src/core/defi/moonwell/strategies/USDC.ts
- src/core/defi/morpho/constants.ts

- src/core/defi/morpho/morpho.config.ts
- src/core/defi/morpho/morpho.module.ts
- src/core/defi/morpho/morpho.service.ts
- src/core/defi/morpho/strategies/interface.ts
- src/core/defi/morpho/strategies/USDCVaults.ts
- src/core/defi/multicall/multicall.service.ts
- src/core/defi/pendle/interface.ts
- src/core/defi/pendle/pendle.config.ts
- src/core/defi/pendle/pendle.module.ts
- src/core/defi/pendle/pendle.service.ts
- src/core/defi/pendle/strategies/index.ts
- src/core/defi/pendle/strategies/partial_withdraw.ts
- src/core/defi/penpie/penpie.module.ts
- src/core/defi/penpie/penpie.service.ts
- src/core/defi/penpie/strategies/AAVE-aUSDC.ts
- src/core/defi/penpie/strategies/abi.ts
- src/core/defi/penpie/strategies/index.ts
- src/core/defi/silo/interfaces.ts
- src/core/defi/silo/silo.config.ts
- src/core/defi/silo/silo.module.ts
- src/core/defi/silo/silo.service.ts
- src/core/defi/silo/strategies/Apostro_USDC.ts
- src/core/defi/silo/strategies/Greenhouse_USDC.ts
- src/core/defi/silo/strategies/index.ts
- src/core/defi/silo/strategies/partial_withdraw.ts
- src/core/defi/silo/strategies/Re7_scUSD.ts
- src/core/defi/silo/strategies/Varlamore_USDC_Growth.ts
- src/core/defi/silo/strategies/vault.ts
- src/core/defi/silo/strategies/x33_USDC_49.ts
- src/core/defi/spark/readme.md
- src/core/defi/spark/spark.config.ts

- src/core/defi/spark/spark.module.ts
- src/core/defi/spark/spark.service.ts
- src/core/defi/spectra/readme.md
- src/core/defi/spectra/strategies/LP-scUSD.ts
- src/core/defi/sushi/sushi.config.ts
- src/core/defi/sushi/sushi.module.ts
- src/core/defi/sushi/sushi.service.ts
- src/core/defi/sushi/utils/index.ts
- src/core/defi/sushi/utils/swapExactAmount.ts
- src/core/defi/swapx/swapx.config.ts
- src/core/defi/swapx/swapx.module.ts
- src/core/defi/swapx/swapx.service.ts
- src/core/defi/wasabi/wasabi.config.ts
- src/core/defi/wasabi/wasabi.module.ts
- src/core/defi/wasabi/wasabi.service.ts
- src/core/defi/yieldfi/yield.config.ts
- src/core/defi/yieldfi/yieldfi.module.ts
- src/core/defi/yieldfi/yieldfi.service.ts
- src/core/eliza/eliza.module.ts
- src/core/eliza/eliza.service.ts
- src/core/eliza/types.ts
- src/core/enso/strategies/euler-PT-aSonUSDC-14AUG2025-scUSD.ts
- src/core/enso/utils/approve.ts
- src/core/enso/utils/route.ts
- src/core/fees/entities/fees.entity.ts
- src/core/fees/fees.constants.ts
- src/core/fees/fees.module.ts
- src/core/fees/fees.service.ts
- src/core/fees/fees.utils.ts
- src/core/lifi/lifi-swap.ts
- src/core/rebalance/cache/rebalance-cache.service.ts

- src/core/rebalance/interface.ts
- src/core/rebalance/rebalance.module.ts
- src/core/rebalance/rebalance.service.ts
- src/core/rebalance/strategies/base.ts
- src/core/rebalance/strategies/crosschain.ts
- src/core/rebalance/strategies/single-sided-stable-opportunity.ts
- src/core/rhinestone/utils.ts
- src/core/signing/eoa-derivation.service.ts
- src/core/signing/signing.module.ts
- src/core/signing/signing.service.ts
- src/core/signing/turnkey.service.ts
- src/core/tenderly/utils.ts
- src/core/transaction-runner/entities/onchain-transaction.entity.ts
- src/core/transaction-runner/entities/useroperation.entity.ts
- src/core/transaction-runner/transaction-runner.module.ts
- src/core/transaction-runner/transaction-runner.service.ts
- src/core/wallet/entities/rebalance-log.entity.ts
- src/core/wallet/entities/wallet-state.entity.ts
- src/core/wallet/entities/wallet-transaction.entity.ts
- src/core/wallet/entities/wallet-transaction-multi-position.entity.ts
- src/core/wallet/entities/wallet-transaction-position.entity.ts
- src/core/wallet/wallet.module.ts
- src/core/wallet/wallet.service.ts
- src/database/repositories/.gitkeep
- src/database/typeorm-config.service.ts
- src/database/utils/bigint-transformer.ts
- src/guards/api-key.guard.ts
- src/guards/auth.guard.ts
- src/guards/data-api-key.guard.ts
- src/guards/erc4626-api-key.guard.ts
- src/main.ts

Repository: PaulDeFi/zyfai-executor-module

Audited Commit: e35f3e41527b7e89345c89360d8257c11c90ab10

Final Commit: 0c1cf9a54a42e23f3581fe83047286791408789b

Files:

- src/module/GuardedExecModuleUpgradeable.sol
- src/registry/TargetRegistry.sol
- test/GuardedExecModuleUpgradeableTest.t.sol
- test/TargetRegistryTest.t.sol

Findings

Each issue has an assigned severity:

- High issues are directly exploitable security vulnerabilities that need to be fixed.
- Medium issues are security vulnerabilities that may not be directly exploitable or may require certain conditions in order to be exploited. All major issues should be addressed.
- Low/Info issues are non-exploitable, informational findings that do not pose a security risk or impact the system's integrity. These issues are typically cosmetic or related to compliance requirements, and are not considered a priority for remediation.

Issues Found

High	Medium	Low/Info
0	8	32

Issues Not Fixed and Not Acknowledged

High	Medium	Low/Info
0	0	0

Issue M-1: [Code Review] Multiple hardcoded secrets and API keys [RESOLVED]

Source: https://github.com/sherlock-audit/2025-11-zyfai-nov-11th/issues/128

Summary

Multiple hardcoded secrets and API keys have been identified throughout the codebase. These credentials are directly embedded in source code rather than being loaded from environment variables or secure configuration systems. This exposes sensitive credentials to version control, making them accessible to anyone with repository access, including past commits, forks, and potential attackers.

Vulnerability Detail

The following hardcoded secrets and API keys have been identified:

1. Turnkey API Credentials

Location: `zyfai-backend/src/core/signing/turnkey.service.ts:15-20`

```
const TURNKEY_API_KEY =
  '****';
const TURNKEY_API_KEY_SECRET =
  '****';
const TURNKEY_API_URL = 'https://api.turnkey.com';
const TURNKEY_ORGANIZATION_ID = '1f965845-2999-40a1-8815-adf5020348a4';
```

Usage: These credentials are used to authenticate with Turnkey's API for signing operations. The API key and secret are used to create an `ApiKeyStamper` for the Turnkey client.

Impact:

- Unauthorized access to Turnkey services
- Potential compromise of signing operations
- Financial risk if used for production transactions
- Organization ID exposure may reveal internal infrastructure details

2. Express Session Secret

Location: `zyfai-backend/src/main.ts:104`

```
app.use(
  session({
    secret: '****=',
    // ...
  }),
);
```

Usage: Used by `express-session` middleware for signing and encrypting session cookies.

Impact:

- Session hijacking if the secret is compromised
- Ability to forge or tamper with session cookies
- Note: The application primarily uses JWT tokens for authentication, but this secret is still a security risk

3. Pimlico API Keys

Location: `zyfai-backend/src/constants/networks.constants.ts:69, 72, 92, 95, 113, 116, 133, 136, 154, 157`

```
pimlicoUrl: 'https://api.pimlico.io/v2/146/rpc?apikey=test',
pimlicoApiKey: '****',
```

Usage: Used for Pimlico bundler and paymaster services across multiple blockchain networks (Sonic, Base, Arbitrum, Linea, Plasma).

Impact:

- Unauthorized usage of Pimlico services
- Potential billing abuse
- Service disruption if keys are revoked
- Affects multiple blockchain networks

4. Gelato API Key

Location: `zyfai-backend/src/constants/networks.constants.ts:71`

```
gelatoApiKey: '****',
```

Usage: Used for Gelato automation services on the Sonic network.

Impact:

- Unauthorized access to Gelato services

- Potential billing abuse
- Service disruption if key is revoked

5. Alchemy API Key

Location:

- `zyfai-backend/src/constants/networks.constants.ts:57, 80, 103, 124, 144` (RPC URLs)
- `zyfai-backend/src/core/rhinestone/utils.ts:111, 155` (Provider configuration)

```
// In RPC URLs
'https://sonic-mainnet.g.alchemy.com/v2/***'
'https://base-mainnet.g.alchemy.com/v2/***'
'https://arb-mainnet.g.alchemy.com/v2/***'
'https://linea-mainnet.g.alchemy.com/v2/***'
'https://plasma-mainnet.g.alchemy.com/v2/***'

// In provider configuration
apiKey: '****',
```

Usage: Used for Alchemy RPC endpoints and provider configuration across multiple blockchain networks.

Impact:

- Unauthorized access to Alchemy RPC services
- Potential billing abuse (rate limits and usage quotas)
- Service disruption if key is revoked
- Affects multiple blockchain networks

Impact

Attackers can use exposed API keys to make unauthorized API calls.

Tool Used

Manual Review

Recommendation

Replace all hardcoded values with environment variables.

Discussion

0xSulpiride

Thanks! It probably makes sense to rotate them and replace them with environment variables now that more people have access to the repo

Issue M-2: [API] Deterministic session key derivation from single master mnemonic [RESOLVED]

Source: https://github.com/sherlock-audit/2025-11-zyfai-nov-11th/issues/133

Summary

The application uses a deterministic method to derive all session key private keys from a single master mnemonic (`SIGNER_MNEMONIC`). The derivation path is calculated directly from wallet addresses, creating a predictable and centralized key management system. This design introduces a single point of failure where compromise of the master mnemonic would enable an attacker to derive all session keys for all users in the system.

Vulnerability Detail

The `SignerService.sessionSigner()` method derives session key private keys deterministically using HD wallet derivation:

```
private sessionSigner(wallet: Address) {
  const [accountIndex, addressIndex, changeIndex] = wallet
    .match(/.{14}/g)
    .map((idx) => (idx.startsWith('0x') ? idx : `0x${idx}`))
    .map((idx) => hexToNumber(idx as Hex))
    .map((idx) => idx % (0x80000000 - 1));
  return mnemonicToAccount(this.mnemonic, {
    accountIndex,
    addressIndex,
    changeIndex,
  });
}
```

Key Characteristics:

- All session keys derive from a single master mnemonic stored in `SIGNER_MNEMONIC` environment variable
- Derivation path is deterministically calculated from the wallet address
- The same wallet address always produces the same session key
- Private keys are derived on-demand but the derivation algorithm is predictable

Storage Model:

- Master mnemonic: Stored in environment variable `SIGNER_MNEMONIC`
- Session key metadata: Stored in database (`session_key` table)
- Private keys: Never stored, only derived when needed

- Derivation algorithm: Public and deterministic

Impact

- Compromise of the `SIGNER_MNEMONIC` environment variable would allow an attacker to derive all session keys for all users.
- No key isolation between users - all keys share the same root.

Code Snippet

File: `zyfai-backend/src/core/signing/signing.service.ts`

File: `zyfai-backend/src/core/signing/signing.service.ts`

Tool Used

Manual Review

Recommendation

Use Hardware Security Modules (HSM) or Secure Enclaves:

Store the master mnemonic in a hardware security module or trusted execution environment:

- Use cloud HSM services (AWS CloudHSM, Azure Dedicated HSM, Google Cloud HSM).
- Implement TEE (Trusted Execution Environment) for key derivation.
- Ensure the mnemonic never leaves the secure environment.

Discussion

Lucas | Sherlock

The issue has been fixed.

Issue M-3: [Code Review] Hardcoded email password [RESOLVED]

Source: https://github.com/sherlock-audit/2025-11-zyfai-nov-11th/issues/134

Summary

A hardcoded email password (Gmail app password) is present in the mail module configuration (`src/mail/mail.module.ts`). This password is directly embedded in source code rather than being loaded from environment variables or a secure configuration system. This exposes the email account credentials to version control, making them accessible to anyone with repository access, including past commits, forks, and potential attackers.

Vulnerability Detail

The email password is hardcoded in the NestJS MailerModule configuration:

Location: `zyfai-backend/src/mail/mail.module.ts:15-18`

```
@Module({
  imports: [
    MailerModule.forRoot({
      transport: {
        host: 'smtp.gmail.com',
        port: 465,
        secure: true,
        logger: true,
        debug: true,
        auth: {
          user: 'contact@zyf.ai',
          pass: '******', //   HARDCODED PASSWORD
        },
        tls: {
          rejectUnauthorized: false, //   ADDITIONAL SECURITY CONCERN
        },
      },
    }),
  ],
  // ...
})
```

Impact

Attackers with access to the repository can:

- Access the `contact@zyf.ai` Gmail account
- Read all emails sent to/from this account
- Send emails impersonating the organization
- Access other services linked to this email account
- Reset passwords for accounts using this email

Code Snippet

```
// zyfai-backend/src/mail/mail.module.ts
import { MailerModule } from '@nestjs-modules/mailer';
import { Global, Module } from '@nestjs/common';
import { MailService } from './mail.service';

@Global()
@Module({
  imports: [
    MailerModule.forRoot({
      transport: {
        host: 'smtp.gmail.com',
        port: 465,
        secure: true,
        logger: true,
        debug: true,
        auth: {
          user: 'contact@zyf.ai',
          pass: '-******', //  HARDCODED PASSWORD
        },
        tls: {
          rejectUnauthorized: false, //  DISABLES CERTIFICATE VALIDATION
        },
      },
    }),
  ],
  providers: [MailService],
  exports: [MailService],
})
export class MailModule {}
```

Tool Used

Manual Review

Recommendation

Remove hardcoded password and replace with environment variable.

Discussion

0xSulpiride

Thank you for highlighting this! We will rotate this api key

0xSulpiride

https://github.com/0xSulpiride/zyfai-backend/commit/793a695d8456d22fa1530163f79c2d5ce807dd06

defsec

Fix is confirmed.

Issue M-4: [Code Review] Weak random number generation for session hashes [RESOLVED]

Source: https://github.com/sherlock-audit/2025-11-zyfai-nov-11th/issues/147

Summary

The application uses `randomStringGenerator()` from NestJS utilities to generate session hashes. This function may not provide cryptographically secure randomness, potentially making session tokens predictable and vulnerable to session hijacking attacks.

Vulnerability Detail

Session hashes are generated using `randomStringGenerator()` which is then hashed with SHA-256. The underlying randomness source may not be cryptographically secure:

```
// src/api/auth/auth.service.ts:88-91
const hash = crypto
  .createHash('sha256')
  .update(randomStringGenerator())
  .digest('hex');
```

The same pattern is used in the refresh token flow:

```
// src/api/auth/auth.service.ts:138-141
const newHash = crypto
  .createHash('sha256')
  .update(randomStringGenerator())
  .digest('hex');
```

Reference : https://github.com/nestjs/nest/issues/14628

Impact

Predictable session hashes allow attackers to guess or brute-force valid session tokens.

Code Snippet

```
// src/api/auth/auth.service.ts:88-91
async signIn(dto: { address: string }): Promise<LoginResDto> {
  // ... user lookup ...

  const hash = crypto
    .createHash('sha256')
```

```
    .update(randomStringGenerator())
    .digest('hex');

  const session = new SessionEntity({
    hash,
    userId: user.id,
  });
  await session.save();
  // ...
}

// src/api/auth/auth.service.ts:138-141
async refreshToken(dto: RefreshReqDto): Promise<RefreshResDto> {
  // ... validation ...

  const newHash = crypto
    .createHash('sha256')
    .update(randomStringGenerator())
    .digest('hex');

  await SessionEntity.update(session.id, { hash: newHash });
  // ...
}
```

Tool Used

Manual Review

Recommendation

Replace with Cryptographically Secure Random Number Generator:

- Use `crypto.randomBytes()` for generating random data.
- Ensure sufficient entropy (at least 32 bytes = 256 bits).
- Hash the random bytes to create session hashes.

Discussion

0xSulpiride

Fixed - https://github.com/0xSulpiride/zyfai-backend/commit/1db92fd5805df66957ecdc9d7f10b913976babbd

defsec

Hi @0xSulpiride , can we get hash of random number instead of using directly random number on our end?

Issue M-5: [Smart Contract] Missing ERC20 approve() and transferFrom() validation [RESOLVED]

Source: https://github.com/sherlock-audit/2025-11-zyfai-nov-11th/issues/154

Summary

The `executeGuardedBatch()` function only validates `transfer()` calls for ERC20 transfer authorization, but does not validate `approve()` or `transferFrom()` calls. This allows session keys to approve unlimited token spending or use `transferFrom()` to drain tokens, bypassing the intended transfer restrictions.

Vulnerability Detail

The executor module implements special validation for ERC20 `transfer()` calls to ensure tokens can only be sent to authorized recipients (wallet itself, wallet owners, or explicitly authorized addresses). However, it does not validate `approve()` or `transferFrom()` calls, which can be used to bypass these restrictions.

Impact

Code Snippet

https://github.com/sherlock-audit/2025-11-zyfai-nov-11th/blob/main/zyfai-executor-module/src/module/GuardedExecModuleUpgradeable.sol#L259

Tool Used

Manual Review

Recommendation

Add validation for `approve()` and `transferFrom()` selectors.

Discussion

sunnyRK

Fixed: https://github.com/PaulDeFi/zyfai-executor-module/commit/fa8adf334f47c7e96ad506b51f913336e1a6ffac

Add ERC20 approve validation and related tests

- Introduced validation for ERC20 approve calls in GuardedExecModuleUpgradeable to ensure only whitelisted spenders can be authorized.
- Updated TargetRegistry to maintain a count of whitelisted selectors and track whitelisted targets.
- Enhanced tests to cover scenarios for both authorized and unauthorized ERC20 approve attempts, ensuring proper functionality and security checks.

Can you verify this @defsec ? Also want to know from you regarding implementation and also as we discussed transferFrom is not needed as we are never going to whitelist it so.

defsec

The issue has been fixed. Thank you!

Issue M-6: [Code Review] Missing response validation in DeFi api integrations [RESOLVED]

Source: https://github.com/sherlock-audit/2025-11-zyfai-nov-11th/issues/157

Summary

The Sushi integration logic does not validate third-party API responses before using them. Malformed or malicious responses can propagate directly into transaction-building logic, potentially causing invalid or harmful on-chain calls.

Vulnerability Detail

Multiple DeFi API calls (e.g., Sushi routing API) directly parse and consume JSON responses without validating structure, types, or required fields. The code assumes the presence of fields such as tx.to, tx.data, and assumedAmountOut, and uses them to construct on-chain transaction calldata.

If the API returns malformed JSON, unexpected field formats, or intentionally malicious payloads, the application may:

- Construct transactions with invalid or malicious to addresses
- Embed malformed calldata that leads to reverts
- Trigger unintended contract interactions

Because these external APIs are not fully trusted, strict validation is required before consuming response data.

Impact

If a compromised or spoofed API returns malicious values, the application may generate harmful transactions (e.g., arbitrary contract calls).

Code Snippet

https://github.com/sherlock-audit/2025-11-zyfai-nov-11th/blob/main/zyfai-backend/src/core/defi/sushi/utils/swapExactAmount.ts#L38

Tool Used

Manual Review

Recommendation

Add a validation on the fields.

Discussion

0xSulpiride

We don't use Sushi, but we use enso and lifi. Added validation for them - https://github.com/0xSulpiride/zyfai-backend/commit/3ccbf8550491c5b8a326059c66eb224f0723d7eb

Issue M-7: [Code Review] BigInt to Number conversion precision loss [RESOLVED]

Source: https://github.com/sherlock-audit/2025-11-zyfai-nov-11th/issues/163

Summary

The fee calculation service converts large BigInt token amounts to JavaScript Number type to calculate ratios. JavaScript Number can only safely represent integers up to 9,007,199,254,740,991. Token amounts with 18 decimals exceeding this limit lose precision when converted, causing incorrect ratio calculations used for fee computation and logging.

Reference : https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Number/MAX_SAFE_INTEGER

Vulnerability Detail

Location: `src/core/fees/fees.service.ts:421-422, 432`

The code converts BigInt token amounts to Number for ratio calculations:

```
// Line 421-422 - Used in debug logging
ratio=${(
  (Number(totalEarnings) / Number(currentTotalPosition)) * 100
).toFixed(4)}%

// Line 432 - Returned as earningsRatio
earningsRatio: Number(totalEarnings) / Number(currentTotalPosition),
```

The Problem:

JavaScript's Number type uses 64-bit floating point with 53-bit precision for integers. This means it can only safely represent integers up to `Number.MAX_SAFE_INTEGER` (9,007,199,254,740,991).

For tokens with 18 decimals:

- 1 token = 1,000,000,000,000,000,000 wei
- 9,007 tokens = 9,007,000,000,000,000,000,000 wei
- This exceeds MAX_SAFE_INTEGER, causing precision loss

When BigInt values exceed this limit, converting to Number rounds to the nearest representable value, losing precision. This affects:

1. The `earningsRatio` calculation returned to callers
2. Debug logging that shows incorrect ratios

3. Any downstream code using the `earningsRatio` value

Impact

The `earningsRatio` value may be slightly inaccurate for large positions.

Code Snippet

https://github.com/sherlock-audit/2025-11-zyfai-nov-11th/blob/main/zyfai-backend/src/core/fees/fees.service.ts#L432

Tool Used

Manual Review

Recommendation

Keep calculations in BigInt and only convert the final result.

Reference : https://github.com/MetaMask/metamask-extension/issues/25755

Discussion

0xSulpiride

This is fine until we add assets with 18 decimals. For now we're just using USDC, but yes, in future we should switch to using only BigInt. We can close this issue.

Issue M-8: [API] Exposed Pimlico API Key in client-side requests [RESOLVED]

Source: https://github.com/sherlock-audit/2025-11-zyfai-nov-11th/issues/182

Summary

The Pimlico API key (`pim_xxxx`) is exposed in client-side network requests because the frontend makes direct calls to `api.pimlico.io` instead of using the backend proxy endpoint.

Vulnerability Detail

The backend has a proxy implementation at `/bundler/:chainId` in `main.ts` that is designed to hide the Pimlico API key from clients. However, the frontend is making direct requests to `https://api.pimlico.io/v2/9745/rpc?apikey=pim_Pj8F4e4yjMiBej7UmJjgcC` instead of using the proxy.

Current Flow (Vulnerable):

```
Frontend → Direct Request → api.pimlico.io?apikey=xxxx
                          @audit API key visible in browser DevTools
```

Intended Flow (Secure):

```
Frontend → Backend Proxy (/bundler/:chainId) → api.pimlico.io?apikey=HIDDEN
                                              @audit API key never exposed to client
```

Evidence:

- Network request shows: `POST /v2/9745/rpc?apikey=xxxx`
- Request goes directly to `api.pimlico.io` (not through backend)
- API key is visible in URL query parameters
- Same API key is hardcoded in `src/constants/networks.constants.ts` for all chains

Impact

Sponsorship wallet can be manipulated with the api key.

Code Snippet

Vulnerable Request (Client-Side - Visible in Browser):

```
POST /v2/9745/rpc?apikey=xxxx HTTP/2
Host: api.pimlico.io
Origin: https://www.zyf.ai
```

Tool Used

Manual Review

Recommendation

Update the frontend to use the backend proxy endpoint instead of direct Pimlico calls.

Discussion

0xSulpiride

we will limit api keys to specific IPs and domains in Pimlico's dashboard

defsec

Hi @0xSulpiride , In that case still the user is able to see API key but are you going to white-list only backend?

0xSulpiride

@defsec yes, do you think it makes sense to hide it and create a proxy? Since the proxy server just redirects to the api, we would still need to whitelists ips and domains

defsec

I thought still end user needs to call pimlico endpoint, according to that I thought Its not possible to white-list IPs, I'm not sure If I'm missing anything

0xSulpiride

yep, i can do this in pimlico's dashboard

I will do this after devconnect though when I will be able to monitor everything 24/7

defsec

Got it, thank you!

Issue L-1: [API] Servers not behind cloudflare [RESOLVED]

Source: https://github.com/sherlock-audit/2025-11-zyfai-nov-11th/issues/129

Summary

The API servers appears to be exposed directly to the internet without protection from Cloudflare or another reverse proxy.

Vulnerability Detail

The API Servers appears to be exposed directly to the internet without protection from Cloudflare or another reverse proxy.

Impact

- Increased risk of **DDoS attacks**, **IP-based enumeration**, or **direct scanning**
- Lack of **WAF** (Web Application Firewall) and **rate limiting**

Code Snippet

```
https://zyf.ai/
https://defiapi.zyf.ai/
https://staging-defiapi.zyf.ai/
https://api.zyf.ai/
https://staging-api.zyf.ai/
```

Tool Used

Manual Review

Recommendation

Route all traffic through a security-focused reverse proxy like **Cloudflare**, **AWS CloudFront** to harden edge-layer protection and obfuscate infrastructure details.

Discussion

0xSulpiride

These were already turned on in cloudflare:

- HTTP DDoS attack protection
- Network-layer DDoS attack protection
- SSL/TLS DDoS attack protection

Not sure if I need to do anything else here.

defsec

Hi @0xSulpiride , This looks great! Thank you so much! We can mark as fixed according to these controls however, If there is any possibility to enable WAF also? Thank you!

Issue L-2: [API] Express server information disclosure via X-Powered-By header [RESOLVED]

Source: https://github.com/sherlock-audit/2025-11-zyfai-nov-11th/issues/130

Summary

The API exposes the `X-Powered-By: Express` HTTP response header, which reveals information about the backend technology stack (Express.js framework). This unnecessary information disclosure can aid attackers in identifying potential vulnerabilities and targeting framework-specific exploits.

Vulnerability Detail

Observed in Production:

```
HTTP/1.1 200 OK
Server: nginx/1.26.0 (Ubuntu)
X-Powered-By: Express
...
```

Current Code Configuration: The application uses Helmet.js for security headers, but the `X-Powered-By` header is not explicitly disabled. In `src/main.ts`, Helmet is configured but doesn't disable this header:

```
app.use(
  helmet({
    crossOriginEmbedderPolicy: false,
    contentSecurityPolicy: false,
    xXssProtection: true,
  }),
);
```

Impact

Reveals the web framework being used, allowing attackers to:

- Research known vulnerabilities specific to Express.js
- Target framework-specific attack vectors
- Identify potential security misconfigurations

Code Snippet

HTTP Request:

```
GET /api/earnings/wallet-earnings-summary/0x4F0813A7A06b57a56D986A344f580c0148EE18E
↪ F HTTP/1.1
Host: defiapi.zyf.ai
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:144.0) Gecko/20100101
↪ Firefox/144.0
Accept: application/json, text/plain, */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Referer: https://www.zyf.ai/
Origin: https://www.zyf.ai
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-site
Te: trailers
Connection: keep-alive
```

HTTP Response Headers Observed:

```
HTTP/1.1 200 OK
Server: nginx/1.26.0 (Ubuntu)
Date: Tue, 11 Nov 2025 19:49:38 GMT
Content-Type: application/json; charset=utf-8
X-Powered-By: Express  ← VULNERABILITY
Access-Control-Allow-Origin: https://www.zyf.ai
X-Content-Type-Options: nosniff
X-Frame-Options: DENY
X-XSS-Protection: 1; mode=block
...
```

Tool Used

Manual Review

Recommendation

Disable X-Powered-By Header.

Discussion

defsec

The issue has been fixed.

Issue L-3: [API] No rate limiting on endpoints [RESOLVED]

Source: https://github.com/sherlock-audit/2025-11-zyfai-nov-11th/issues/132

Summary

No evidence of rate limiting on the endpoints, enabling brute-force attacks.

Vulnerability Detail

On the APIs, there is no rate-limiting observed during the testing.

Impact

- Brute-force attacks on wallet addresses
- DoS through repeated authentication attempts
- Session exhaustion attacks

Tool Used

Manual Review

Recommendation

Implement rate limiting:

```
@UseGuards(ThrottlerGuard)
@Throttle({ default: { limit: 5, ttl: 60000 } })
async signIn(@Body() dto: LoginReqDto) { ... }
```

Issue L-4: [Code Review] Information disclosure in login error responses [RESOLVED]

Source: https://github.com/sherlock-audit/2025-11-zyfai-nov-11th/issues/136

Summary

The global exception filter returns detailed error information including stack traces to clients when debug mode is enabled. This affects authentication endpoints, potentially exposing internal application structure, file paths, and database schema information to attackers during failed login attempts.

Vulnerability Detail

The `GlobalExceptionFilter` includes stack traces and full error objects in API responses when `app.debug` is enabled:

```
// src/filters/global-exception.filter.ts:55-60
if (this.debug) {
  error.stack = exception.stack;
  error.trace = exception;

  this.logger.debug(error);
}

response.status(error.statusCode).json(error);
```

When authentication fails (e.g., invalid JWT token, expired session, or database errors during login), the error response may include:

- Full stack traces revealing file paths and internal structure
- Database error messages exposing schema information
- Internal exception details useful for reconnaissance

Impact

Stack traces reveal application file structure and internal implementation.

Code Snippet

```
// src/filters/global-exception.filter.ts:31-63
catch(exception: any, host: ArgumentsHost): void {
  // ... error handling logic ...
```

```
  if (this.debug) {
    error.stack = exception.stack;  // @audit Stack trace exposed
    error.trace = exception;          // @audit Full error object exposed
    this.logger.debug(error);
  }

  response.status(error.statusCode).json(error);  // @audit Sent to client
}
```

Tool Used

Manual Review

Recommendation

1. **Never return stack traces to clients in production:**
    - Remove stack trace and trace fields from client responses
    - Ensure `app.debug` is `false` in production environment
2. **Use generic error messages:**
    - Return generic messages like "Authentication failed" for login errors
    - Avoid exposing specific failure reasons (invalid token vs expired session)

Discussion

0xSulpiride

app.debug is set in .env. I will set it to false in prod

defsec

Marking as a fixed.

Issue L-5: [Code Review] API Key timing attack vulnerability [RESOLVED]

Source: https://github.com/sherlock-audit/2025-11-zyfai-nov-11th/issues/137

Summary

API key authentication guards use simple string comparison (`!==`) which is vulnerable to timing attacks. Attackers can measure response times to gradually discover API keys character-by-character, enabling unauthorized access to protected endpoints.

Vulnerability Detail

Three API key guards perform non-constant-time string comparisons when validating API keys:

1. **ApiKeyAuthGuard** - Validates `API_KEY` environment variable
2. **DataApiKeyAuthGuard** - Validates `DATA_API_KEY` environment variable
3. **Erc4626ApiKeyAuthGuard** - Validates `ERC4626_API_KEY` environment variable

All guards use JavaScript's `!==` operator which returns immediately upon finding the first differing character. This creates a timing difference that attackers can measure:

- Correct characters at the start = longer comparison time
- Incorrect characters at the start = shorter comparison time
- Attackers can systematically discover the API key by measuring response times

Impact

Attackers can gradually discover API keys through timing analysis.

Code Snippet

```
// src/guards/api-key.guard.ts:21
if (apiKey !== process.env.API_KEY) {  // @audit Timing attack vulnerable
  throw new UnauthorizedException('Invalid API key.');
}

// src/guards/data-api-key.guard.ts:21
if (apiKey !== process.env.DATA_API_KEY) {  // @audit Timing attack vulnerable
  throw new UnauthorizedException('Invalid API key.');
}
```

```
// src/guards/erc4626-api-key.guard.ts:21
if (apiKey !== process.env.ERC4626_API_KEY) {  // @audit Timing attack vulnerable
  throw new UnauthorizedException('Invalid API key.');
}
```

Tool Used

Manual Review

Recommendation

Use constant-time comparison:

- Replace string comparison with `crypto.timingSafeEqual()`.
- Ensure both strings are converted to Buffers of equal length.

Discussion

0xSulpiride

Fix - https://github.com/0xSulpiride/zyfai-backend/commit/5cfe358e7c0eae1abd51acde9ccad1cad0128c79

defsec

Fix is confirmed.

Issue L-6: [API] Missing input validation on pagination parameters [RESOLVED]

Source: https://github.com/sherlock-audit/2025-11-zyfai-nov-11th/issues/139

Summary

Pagination parameters lack validation, allowing potential DoS through large limit values, negative numbers, and invalid input handling.

Vulnerability Detail

```
// src/api/user/user.controller.ts:135-140
async getUsersAutoselect(
  @Query('page') page?: string,
  @Query('limit') limit?: string,
): Promise<UserAutoselectResDto> {
  const pageNum = page ? parseInt(page, 10) : 1;  // @audit No validation
  const limitNum = limit ? parseInt(limit, 10) : 100;  // @audit No validation
  return await this.userService.getUsersAutoselect(pageNum, limitNum);
}
```

Legitimate Request (Expected Behavior)

```
GET /api/v1/users/autoselect?page=1&limit=100 HTTP/1.1
Host: api.zyf.ai
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:144.0) Gecko/20100101
↪  Firefox/144.0
Accept: application/json, text/plain, */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Referer: https://www.zyf.ai/
Origin: https://www.zyf.ai
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-site
Connection: keep-alive
```

Response: Returns up to 100 users (reasonable limit)

DoS Attack - Extremely Large Limit

This request attempts to retrieve an extremely large number of records, potentially causing database overload and memory exhaustion:

```
GET /api/v1/users/autoselect?page=1&limit=999999999 HTTP/1.1
Host: api.zyf.ai
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:144.0) Gecko/20100101
↪  Firefox/144.0
Accept: application/json, text/plain, */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Referer: https://www.zyf.ai/
Origin: https://www.zyf.ai
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-site
Connection: keep-alive
```

Impact

Database query may timeout or consume excessive resources.

Code Snippet

```
// src/api/user/user.controller.ts:135-140
async getUsersAutoselect(
  @Query('page') page?: string,
  @Query('limit') limit?: string,
): Promise<UserAutoselectResDto> {
  const pageNum = page ? parseInt(page, 10) : 1;  // @audit No validation
  const limitNum = limit ? parseInt(limit, 10) : 100;  // @audit No validation
  return await this.userService.getUsersAutoselect(pageNum, limitNum);
}
```

Tool Used

Manual Review

Recommendation

Enforce maximum limit on the code.

Discussion

0xSulpiride

Fixed - https://github.com/0xSulpiride/zyfai-backend/commit/451bb452f923603f70fab04b42e226be4483f63e

defsec

Fix is confirmed.

Issue L-7: [Code Review] Missing request body size limits [RESOLVED]

Source: https://github.com/sherlock-audit/2025-11-zyfai-nov-11th/issues/140

Summary

The application does not configure explicit request body size limits for Express/NestJS, allowing potential Denial of Service attacks through oversized request payloads. Default limits may be too large or undefined, enabling attackers to exhaust server resources.

Vulnerability Detail

No request body size limits are configured in the application bootstrap process. Express/NestJS default limits (typically 100kb for JSON) may be insufficient or undefined, allowing:

- Unlimited request body sizes
- Memory exhaustion from processing large payloads
- Resource consumption attacks
- Potential application crashes from oversized requests

The application uses NestJS with Express, but does not configure body parser limits in `main.ts` or application configuration.

Impact

Attackers can send extremely large request bodies to exhaust server memory.

Code Snippet

```
// src/main.ts - No body size limits configured
async function bootstrap() {
  const app = await NestFactory.create<NestExpressApplication>(AppModule, {
    bufferLogs: true,
  });

  // @audit No express.json() or express.urlencoded() with limits configured
  // Default limits may be undefined or too large

  app.useGlobalPipes(
    new ValidationPipe({
      transform: true,
```

```
      whitelist: true,
      // ... validation config
    }),
  );
}
```

Tool Used

Manual Review

Recommendation

Configure body parser limits:

- Set explicit limits for JSON and URL-encoded bodies.
- Use reasonable limits (e.g., 1-10MB depending on endpoint needs).
- Configure limits in application bootstrap.

Discussion

0xSulpiride

Fixed - https://github.com/0xSulpiride/zyfai-backend/commit/76960e91ec3eec8796f59a5e85ed40f74d23c90e

defsec

Fix is confirmed.

Issue L-8: [Code Review] Missing request timeout on external API calls [RESOLVED]

Source: https://github.com/sherlock-audit/2025-11-zyfai-nov-11th/issues/141

Summary

External API calls using `fetch()` lack timeout configuration, allowing requests to hang indefinitely. This can lead to resource exhaustion, connection pool depletion, and potential Denial of Service if external services are slow or unresponsive.

Vulnerability Detail

Multiple external API calls throughout the codebase use `fetch()` without timeout configuration:

1. **ElizaService** - Fetches opportunities data without timeout
2. **Other external API calls** - Various services make HTTP requests without timeout limits

When external services are slow, unresponsive, or experiencing issues, these requests can hang indefinitely, consuming server resources and connection pools.

Impact

Hanging connections consume memory and CPU.

Code Snippet

```
// src/core/eliza/eliza.service.ts:187-194
async getOpportunities(
  chainId: SupportedChain,
  token: SupportedToken,
  strategy: RebalanceStrategy,
): Promise<SingleSidedStableOpportunity[]> {
  const baseUrl = process.env.DEFAI_API_URL;
  const path = strategy === RebalanceStrategy.DEGEN_STRATEGY
    ? Config[chainId].degen
    : Config[chainId].safe;
  const apiUrl = `${baseUrl}${path}`;
  const response = await fetch(apiUrl);  // @audit No timeout configured
  const rawData = await response.json();
  if (rawData.status == 'success') {
    await this.cacheManager.set(cacheKey, rawData.data, 30 * 1000);
```

```
    return rawData.data;
  } else {
    return [];
  }
}
```

Tool Used

Manual Review

Recommendation

Add timeout to all fetch requests:

- Use AbortController with setTimeout for timeout control.
- Set reasonable timeout values (10-30 seconds depending on endpoint).
- Ensure timeout cleanup in error handlers.

Issue L-9: [Code Review] Disabled security headers [RESOLVED]

Source: https://github.com/sherlock-audit/2025-11-zyfai-nov-11th/issues/143

Summary

Helmet security headers are partially disabled, specifically Content Security Policy (CSP) and Cross-Origin Embedder Policy (COEP), leaving the application vulnerable to XSS attacks and missing critical security protections. The application relies on minimal security headers, reducing defense-in-depth against common web vulnerabilities.

Vulnerability Detail

The Helmet middleware is configured with critical security headers disabled:

```
app.use(
  helmet({
    crossOriginEmbedderPolicy: false,  //  DISABLED
    contentSecurityPolicy: false,       //  DISABLED
    xXssProtection: true,               // Only basic XSS protection enabled
  }),
);
```

Impact

Missing protection against cross-origin attacks.

Code Snippet

```
// src/main.ts:86-92
app.use(
  helmet({
    crossOriginEmbedderPolicy: false,  //  DISABLED - Missing COEP protection
    contentSecurityPolicy: false,       //  DISABLED - No CSP
    xXssProtection: true,               // Only basic XSS protection
  }),
);
```

Tool Used

Manual Review

Recommendation

Consider enabling disabled security headers.

Discussion

0xSulpiride

Fixed - https://github.com/0xSulpiride/zyfai-backend/commit/da2d3d1151fa307d831b71482d9c37e15e1111c3

defsec

Fix is confirmed.

Issue L-10: [API] Database error message exposure [RESOLVED]

Source: https://github.com/sherlock-audit/2025-11-zyfai-nov-11th/issues/145

Summary

The `/api/earnings/initialize` endpoint (and potentially other endpoints) exposes raw database error messages to clients, revealing sensitive information about database schema, table names, constraint names, and internal structure. This information disclosure aids attackers in understanding the database architecture and crafting targeted attacks.

Vulnerability Detail

When database errors occur (such as unique constraint violations), the raw database error message is returned to the client:

```
{
  "status": "error",
  "message": "Error creating wallet entry: duplicate key value violates unique
  ↪  constraint \"wallet_earnings_pkey\""
}
```

The error message exposes:

- Table name: `wallet_earnings`
- Constraint name: `wallet_earnings_pkey` (primary key constraint)
- Database structure information
- Internal error details

Impact

Database schema, table names, and constraint names are exposed.

Request

```
POST /api/earnings/initialize HTTP/1.1
Host: defiapi.zyf.ai
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:144.0) Gecko/20100101
↪  Firefox/144.0
Accept: application/json, text/plain, */*
```

```
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Referer: https://www.zyf.ai/
Content-Type: application/json
Content-Length: 62
Origin: https://www.zyf.ai
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-site
Te: trailers
Connection: keep-alive

{"walletAddress":"0x34AB6Cf22ad3b556159eE5f68Bd20dc7e27B9B34"}
```

Response

```
HTTP/1.1 400 Bad Request
Server: nginx/1.26.0 (Ubuntu)
Date: Wed, 12 Nov 2025 08:10:18 GMT
Content-Type: application/json; charset=utf-8
Content-Length: 131
Connection: keep-alive
X-Powered-By: Express
Access-Control-Allow-Origin: https://www.zyf.ai
Vary: Origin
Access-Control-Allow-Credentials: true
X-Content-Type-Options: nosniff
X-Frame-Options: DENY
X-XSS-Protection: 1; mode=block
Referrer-Policy: strict-origin-when-cross-origin
Strict-Transport-Security: max-age=31536000; includeSubDomains
ETag: W/"83-EXD8YgT17RWd/0X3aOUBGBGScR8"

{"status":"error","message":"Error creating wallet entry: duplicate key value
↪  violates unique constraint \"wallet_earnings_pkey\""}
```

Tool Used

Manual Review

Recommendation

- Never return raw database error messages to clients
- Map database error codes to user-friendly messages

- Strip sensitive information (table names, constraint names, SQL fragments)

Discussion

defsec

The issue has been fixed.

Issue L-11: [API] Nginx version disclosure in server header [RESOLVED]

Source: https://github.com/sherlock-audit/2025-11-zyfai-nov-11th/issues/146

Summary

The HTTP `Server` header exposes the Nginx version (`nginx/1.26.0 (Ubuntu)`) to clients, providing attackers with information about the web server software and version. This information disclosure aids in reconnaissance and can help attackers identify known vulnerabilities specific to the server version.

Vulnerability Detail

The HTTP response includes a `Server` header that reveals the Nginx version and operating system:

```
HTTP/1.1 400 Bad Request
Server: nginx/1.26.0 (Ubuntu)
```

Impact

Attackers can identify the web server version and OS

Request

```
POST /api/earnings/initialize HTTP/1.1
Host: defiapi.zyf.ai
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:144.0) Gecko/20100101
↪  Firefox/144.0
Accept: application/json, text/plain, */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Referer: https://www.zyf.ai/
Content-Type: application/json
Content-Length: 62
Origin: https://www.zyf.ai
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-site
Te: trailers
Connection: keep-alive
```

```
{"walletAddress":"0x34AB6Cf22ad3b556159eE5f68Bd20dc7e27B9B34"}
```

Response

```
HTTP/1.1 400 Bad Request
Server: nginx/1.26.0 (Ubuntu)
Date: Wed, 12 Nov 2025 08:10:18 GMT
Content-Type: application/json; charset=utf-8
Content-Length: 131
Connection: keep-alive
X-Powered-By: Express
Access-Control-Allow-Origin: https://www.zyf.ai
Vary: Origin
Access-Control-Allow-Credentials: true
X-Content-Type-Options: nosniff
X-Frame-Options: DENY
X-XSS-Protection: 1; mode=block
Referrer-Policy: strict-origin-when-cross-origin
Strict-Transport-Security: max-age=31536000; includeSubDomains
ETag: W/"83-EXD8YgT17RWd/0X3a0UBGBGScR8"

{"status":"error","message":"Error creating wallet entry: duplicate key value
↪  violates unique constraint \"wallet_earnings_pkey\""}
```

Tool Used

Manual Review

Recommendation

Use custom server header instead of nginx.

Discussion

defsec

The issue has been fixed.

Issue L-12: [Code Review] CSV filename injection [RESOLVED]

Source: https://github.com/sherlock-audit/2025-11-zyfai-nov-11th/issues/148

Summary

The CSV download endpoint constructs the `Content-Disposition` header filename from user-controlled input (`walletAddress` and `days`) without proper sanitization or encoding. This allows HTTP header injection attacks, enabling attackers to inject arbitrary HTTP headers or perform path traversal attacks through the filename parameter.

Vulnerability Detail

The `Content-Disposition` header is set using string interpolation with user-controlled values:

```
// src/api/data/data.controller.ts:107-125
@Get('csv-history')
@Public()
async getCsvHistory(
  @Query('walletAddress') walletAddress: string,
  @Query('days') days: number | string,
  @Res() res: Response,
) {
  // ... processing ...

  res.setHeader('Content-Type', 'text/csv');
  res.setHeader(
    'Content-Disposition',
    `attachment; filename=${walletAddress}-${days}-days.csv`,  //  User-controlled,
    ↪  no sanitization
  );

  // ... CSV generation ...
}
```

1. `walletAddress` and `days` are directly interpolated into the HTTP header without sanitization
2. No validation or encoding of special characters (newlines, quotes, semicolons, etc.)
3. HTTP header injection is possible through newline characters (`\r\n` or `%0d%0a`)
4. Path traversal possible through `../` sequences in filename
5. No length limits on filename, allowing header manipulation

Impact

Filenames with `../` can attempt to write files outside intended directories.

Code Snippet

```
// src/api/data/data.controller.ts:107-125
@Get('csv-history')
@Public()
async getCsvHistory(
  @Query('walletAddress') walletAddress: string,
  @Query('days') days: number | string,
  @Res() res: Response,
) {
  if (days === 'all-time') {
    const firstTopup = await this.dataService.getFirstTopup(walletAddress);
    days = Math.floor(
      (Date.now() - new Date(firstTopup.date).getTime()) / ONE_DAY_MS,
    );
  }
  try {
    res.setHeader('Content-Type', 'text/csv');
    res.setHeader(
      'Content-Disposition',
      `attachment; filename=${walletAddress}-${days}-days.csv`,
    );

    const csvStream = fastcsv.format({ headers: true, writeHeaders: true });
    csvStream.pipe(res);
    await this.dataService.getCsvHistory(
      walletAddress,
      days as number,
      csvStream,
    );
  } catch (error) {
    this.logger.error(
      `Failed to generate CSV history for ${walletAddress}`,
      error,
    );
    return {
      error: 'Failed to generate CSV history',
    };
  }
}
```

Tool Used

Manual Review

Recommendation

Sanitize Filename Input:

- Remove or replace dangerous characters (newlines, quotes, semicolons, path separators)
- Use whitelist approach - only allow safe characters
- Limit filename length to prevent header manipulation

Discussion

0xSulpiride

Fixed https://github.com/0xSulpiride/zyfai-backend/commit/37d09861c862007cb8ed6d6806e5953daa64ea05

defsec

Fix is confirmed.

Issue L-13: [Code Review] Withdrawal validation by-pass via rounding tolerance [ACKNOWLEDGED]

Source: https://github.com/sherlock-audit/2025-11-zyfai-nov-11th/issues/149

This issue has been acknowledged by the team but won't be fixed at this time.

Summary

The withdrawal validation logic in `getSuperPartialWithdrawCall` allows users to request withdrawal amounts that exceed their available balance by up to 10 wei due to a rounding tolerance buffer. This tolerance is intended to account for protocol calculation rounding errors, but it can be exploited to bypass balance validation checks, potentially leading to transaction failures.

Vulnerability Detail

The withdrawal validation includes a `ROUNDING_TOLERANCE` of 10 wei that is added to the available funds check, allowing withdrawal requests that exceed the actual available balance:

```
// src/core/rebalance/strategies/single-sided-stable-opportunity.ts:536-542
const totalProtocolBalance = currentPosition.balance;
const totalAvailableFunds = activeBalance + totalProtocolBalance;

// Add a small tolerance buffer (10 wei) to account for rounding errors in protocol
↪  calculations
// This prevents failures due to tiny precision differences between estimated and
↪  actual withdrawal amounts
const ROUNDING_TOLERANCE = 10n;

if (withdrawAmount > totalAvailableFunds + ROUNDING_TOLERANCE) {
  throw new Error('Insufficient funds for withdrawal');
}
```

The Problem:

1. The tolerance is applied to the validation check, not to the actual withdrawal amount
2. Users can request withdrawals up to `totalAvailableFunds + 10 wei`
3. If a user has exactly 1000 USDC available, they can request 1000 USDC + 10 wei
4. The validation passes, but the actual withdrawal will fail on-chain.

Impact

Withdrawals that pass validation may fail on-chain.

Code Snippet

```
// src/core/rebalance/strategies/single-sided-stable-opportunity.ts:524-542
// Get current protocol position balance
const currentPosition = await this.getWithdrawCall(
  token.chainId,
  protocol,
  pool,
  token,
  user.smartWallet as Address,
  user.address as Address,
);
const totalProtocolBalance = currentPosition.balance;
const totalAvailableFunds = activeBalance + totalProtocolBalance;

// Add a small tolerance buffer (10 wei) to account for rounding errors in protocol
↪  calculations
// This prevents failures due to tiny precision differences between estimated and
↪  actual withdrawal amounts
const ROUNDING_TOLERANCE = 10n;

if (withdrawAmount > totalAvailableFunds + ROUNDING_TOLERANCE) {
  throw new Error('Insufficient funds for withdrawal');
}
```

Tool Used

Manual Review

Recommendation

Apply the tolerance only to protocol-calculated withdrawal amounts, not to user-requested amounts.

Discussion

0xSulpiride

this is fine. user can only withdraw as much as we have onchain and what he has on-chain 100% belongs to him.

Issue L-14: [Smart Contract] Ownable2StepUpgradeable should be used Instead of OwnableUpgradeable [RESOLVED]

Source: https://github.com/sherlock-audit/2025-11-zyfai-nov-11th/issues/150

Summary

The executor smart contracts use `OwnableUpgradeable` which allows instant ownership transfers. This creates a risk where a compromised owner key or accidental misconfiguration can immediately transfer ownership to a malicious address, with no recovery mechanism. The contracts should use `Ownable2StepUpgradeable` instead, which implements a two-step ownership transfer pattern that requires the new owner to explicitly accept the transfer.

Vulnerability Detail

Both `GuardedExecModuleUpgradeable` and `TargetRegistry` use `OwnableUpgradeable`, which allows instant ownership transfers:

Current Implementation:

```
// src/module/GuardedExecModuleUpgradeable.sol:4-5
import "@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol";
// ...
contract GuardedExecModuleUpgradeable is
    ERC7579ExecutorBase,
    OwnableUpgradeable,  //  Single-step ownership transfer
    PausableUpgradeable,
    UUPSUpgradeable
{
    // ...
}

// src/registry/TargetRegistry.sol:5
import "@openzeppelin/contracts/access/Ownable.sol";
// ...
contract TargetRegistry is Ownable, Pausable {  //  Single-step ownership transfer
    // ...
}
```

Impact

`OwnableUpgradeable.transferOwnership()` immediately transfers ownership.

Code Snippet

https://github.com/sherlock-audit/2025-11-zyfai-nov-11th/blob/main/zyfai-executor-module/src/module/GuardedExecModuleUpgradeable.sol#L23

Tool Used

Manual Review

Recommendation

Replace `OwnableUpgradeable` with `Ownable2StepUpgradeable` in both contracts.

- Ownable2StepUpgradeable Documentation

Discussion

sunnyRK

Fixed: https://github.com/PaulDeFi/zyfai-executor-module/commit/9b35c2fd96dccb1810598c2e456801997725c5d4

can you confirm @defsec for Ownable2StepUpgradeable and Ownable2Step? I have implmented in both contracts like GuardedExecModuleUpgradeable and TargetRegistry.

defsec

Thank you so much, the fix looks good to me!

Issue L-15: [Smart Contract] Missing whenNotPaused modifier on executeOperation function [RESOLVED]

Source: https://github.com/sherlock-audit/2025-11-zyfai-nov-11th/issues/151

Summary

The `executeOperation()` function in `TargetRegistry.sol` is missing the `whenNotPaused` modifier, allowing scheduled whitelist operations to be executed even when the contract is paused. This defeats the purpose of the pause functionality, as critical whitelist changes can proceed during emergency situations when the contract should be fully paused.

Vulnerability Detail

The `executeOperation()` function allows anyone to execute scheduled whitelist operations after the timelock delay expires, but it does not check if the contract is paused:

```
// src/registry/TargetRegistry.sol:328-339
function executeOperation(address[] calldata targets, bytes4[] calldata selectors)
↪  external {
    uint256 length = targets.length;
    if (length == 0) revert EmptyBatch();
    if (length != selectors.length) revert LengthMismatch();

    for (uint256 i = 0; i < length;) {
        _executeOperation(targets[i], selectors[i]);
        unchecked {
            ++i;
        }
    }
}
```

The `scheduleAdd()` and `scheduleRemove()` functions correctly include the `whenNotPaused` modifier:

```
// src/registry/TargetRegistry.sol:269-276
function scheduleAdd(
    address[] calldata targets,
    bytes4[] calldata selectors
)
    external
    onlyOwner
    whenNotPaused
```

```
    returns (bytes32[] memory operationIds)
```

```
// src/registry/TargetRegistry.sol:299-306
function scheduleRemove(
    address[] calldata targets,
    bytes4[] calldata selectors
)
    external
    onlyOwner
    whenNotPaused
    returns (bytes32[] memory operationIds)
```

Impact

Scheduling is paused, but execution is not.

Code Snippet

https://github.com/sherlock-audit/2025-11-zyfai-nov-11th/blob/main/zyfai-executor-module/src/registry/TargetRegistry.sol#L328

Tool Used

Manual Review

Recommendation

Add the `whenNotPaused` modifier to the `executeOperation()` function.

Discussion

sunnyRK

Fixed: https://github.com/PaulDeFi/zyfai-executor-module/commit/f082647453627f7b4cd557fb12aedfcabff51504 Can you confirm this `whenNotPaused` modifier to the `executeOperation()` function @defsec ?

defsec

The fix is confirmed, thank you!

Issue L-16: [Code Review] Weak password requirements [RESOLVED]

Source: https://github.com/sherlock-audit/2025-11-zyfai-nov-11th/issues/153

Summary

The application enforces minimal password requirements, allowing users to create weak passwords that are vulnerable to brute-force attacks. The current validation only checks for a minimum length of 6 characters and a limited character set, without enforcing complexity requirements (uppercase, lowercase, numbers, special characters), maximum length limits, or entropy-based validation.

Vulnerability Detail

The password validation is implemented in two places:

1. Character Set Validation (`IsPassword` decorator):

```
// src/decorators/validators/is-password.decorator.ts:14-15
validate(value: string) {
  return /^[\d!#$%&*@A-Z^a-z]*$/.test(value);
}
```

2. Length Validation (`PasswordField` decorator):

```
// src/decorators/field.decorators.ts:190
const decorators = [StringField({ ...options, minLength: 6 }), IsPassword()];
```

- Password "123456" (6 digits) would be accepted
- Password "password" (8 lowercase letters) would be accepted

Impact

Weak passwords are easily cracked through brute-force or dictionary attacks.

Code Snippet

```
// src/decorators/field.decorators.ts:186-199
export function PasswordField(
  options: Omit<ApiPropertyOptions, 'type' | 'minLength'> &
    IStringFieldOptions = {},
): PropertyDecorator {
```

```
  const decorators = [StringField({ ...options, minLength: 6 }), IsPassword()];  //
  ↪  @audit Only 6 characters minimum

  if (options.nullable) {
    decorators.push(IsNullable());
  } else {
    decorators.push(NotEquals(null));
  }

  return applyDecorators(...decorators);
}
```

Tool Used

Manual Review

Recommendation

Implement comprehensive password validation.

Discussion

0xSulpiride

this is from a boilerplate and we don't use this password decorator. but I will keep this in mind if we decide to use it.

Issue L-17: [Code Review] Missing error handling in DeFi protocol API calls [RESOLVED]

Source: https://github.com/sherlock-audit/2025-11-zyfai-nov-11th/issues/158

Summary

DeFi protocol API calls throughout the codebase lack proper error handling, timeout configuration, and response validation. External API calls to services like Pendle, Sushi, and other DeFi protocols can hang indefinitely, fail silently, or proceed with invalid data, leading to service unavailability, incorrect transaction parameters, and potential financial losses.

Vulnerability Detail

Multiple DeFi service implementations make external API calls without proper error handling, timeout configuration, or response validation. The vulnerable code patterns include:

1. **No Timeout Configuration**: API calls can hang indefinitely, exhausting server resources
2. **No Error Handling**: Network errors, API errors, and invalid responses are not caught or handled
3. **No Response Validation**: Response structure, data types, and required fields are not validated before use
4. **Silent Failures**: Errors may not be logged, leading to undetected issues

The vulnerability affects multiple DeFi integrations:

- **Pendle API calls** (`src/core/defi/pendle/pendle.config.ts`): Axios requests without timeout or error handling
- **Sushi API calls** (`src/core/defi/sushi/sushi.service.ts`, `src/core/defi/sushi/utils/swapExactAmount.ts`): Fetch requests without timeout, error handling, or response validation
- **Other DeFi protocol integrations**: Similar patterns exist across multiple services

Impact

Hanging requests can exhaust server resources (memory, connection pools), leading to denial of service.

Code Snippet

```
// src/core/defi/pendle/pendle.config.ts:96-114
const slippage = Pendle.getSlippage(balance) / 10000;
const response = await axios.get(  //  No timeout, no error handling
  `${Pendle.baseUrl}/${chainId}/markets/${market.market}/add-liquidity`,
  {
    headers: {
      'Content-Type': 'application/json',
      Authorization: `Bearer ${apiKey}`,
    },
    params: {
      receiver: wallet,
      amountIn: balance.toString(),
      tokenIn: market.token,
      slippage: slippage.toString(),
      enableAggregator: market.enableAggregator,
      aggregators: market.aggregators,
    },
  },
);

const data: PendleAddLiquidityResponse = response.data;  //  No validation of
↪  response.status
//  No validation of response.data structure
//  No validation of required fields
```

Tool Used

Manual Review

Recommendation

Configure timeouts for all external API calls to prevent indefinite hanging and handle errors on the APIs.

Discussion

0xSulpiride

Fixed - https://github.com/0xSulpiride/zyfai-backend/commit/dd8f7670aeed37fa4156b31ea8c83988424c88b2

Issue L-18: [Code Review] Implement mTLS for TEE Integration [ACKNOWLEDGED]

Source: https://github.com/sherlock-audit/2025-11-zyfai-nov-11th/issues/160

This issue has been acknowledged by the team but won't be fixed at this time.

Summary

When implementing a Trusted Execution Environment (TEE) solution to address the deterministic session key derivation vulnerability (Issue #133), the application should also implement mutual TLS (mTLS) authentication to secure communication between the application and the TEE service. This ensures that only authenticated and authorized services can communicate with the TEE, preventing man-in-the-middle attacks and unauthorized access to the secure key derivation environment.

Vulnerability Detail

*Related Issue:** [API] Deterministic session key derivation from single master mnemonic #133

Current State:

- The application uses a deterministic method to derive session keys from a single master mnemonic
- Communication between application services and external TEE/HSM services is not secured with mutual authentication
- Standard TLS (one-way) only authenticates the server, not the client
- Without mTLS, an attacker who compromises network access could potentially intercept or impersonate communication with the TEE service

Impact

Without mTLS, any service that can reach the TEE endpoint could potentially access key derivation functions.

Tool Used

Manual Review

Recommendation

Implement mutual TLS (mTLS) authentication when deploying the TEE solution to ensure secure, mutually authenticated communication between the application and the TEE service.

Discussion

0xSulpiride

Thanks! For now we don't have TEEs but we will keep this in mind when adding it

Issue L-19: zyf.ai – missing email authentication records (DMARC, SPF, DKIM) [RESOLVED]

Source: https://github.com/sherlock-audit/2025-11-zyfai-nov-11th/issues/161

Summary

We were unable to identify any DMARC, SPF, or DKIM DNS records for the domain **zyf.ai**. Without these authentication mechanisms in place, the domain is not protected against spoofing or abuse and is unlikely to meet the new sender requirements enforced by Google, Yahoo, and other major mailbox providers.

Vulnerability Detail

DMARC No DMARC record was found. This exposes the domain to phishing and spoofing attacks, and prevents receiving mail servers from verifying legitimate messages.

SPF No SPF record was detected. SPF is required to specify which mail servers are authorized to send email on behalf of the domain.

DKIM No DKIM records were identified. DKIM cryptographically signs outgoing mail and is required for domain authenticity. If a selector is known, a targeted lookup can be performed.

Impact

Increased risk of domain impersonation by attackers.

Tool Used

Manual Review

Recommendation

1. **Add a DMARC record** Example (monitoring mode):

```
_dmarc.zyf.ai TXT "v=DMARC1; p=none; rua=mailto:postmaster@zyf.ai"
```

2. **Publish an SPF record** Example (only allows your mail server–adjust as needed):

```
zyf.ai TXT "v=spf1 include:your-mail-provider -all"
```

3. **Configure DKIM**

- Enable DKIM signing on your mail provider
- Publish the provider-generated DKIM public key in DNS

4. **After verification**, consider strengthening DMARC to `quarantine` or `reject`.

Discussion

defsec

The issue has been fixed, however, we can convert DMARC records to ”p=reject;”

Issue L-20: [Code Review] Hardcoded 6 decimal assumptions in financial calculations [RESOLVED]

Source: https://github.com/sherlock-audit/2025-11-zyfai-nov-11th/issues/162

Summary

Multiple parts of the code assume all tokens use 6 decimals (USDC standard). This causes wrong calculations when handling tokens with different decimal places like ETH (18 decimals) or WBTC (8 decimals). The system divides amounts by 1,000,000 or uses formatUnits with 6 decimals everywhere, leading to incorrect balance displays and calculation errors.

Vulnerability Detail

The codebase has hardcoded assumptions that all token amounts use 6 decimal places. This works for USDC but breaks for other tokens. The code divides by `1_000_000n` or calls `formatUnits(amount, 6)` in multiple places without checking the actual token decimals.

Affected Locations:

1. **Balance calculations** - `src/api/data/data.service.ts:404, 408, 423, 436`
   - Divides balances by `1_000_000n` assuming 6 decimals
   - Used in `getActivePositions()` and `getTVL()`
2. **Volume display** - `src/api/data/data.controller.ts:174`
   - Uses `formatUnits(volume, 6)` hardcoded
3. **Transaction history** - `src/api/history/history.service.ts:264`
   - Uses `formatUnits(position?.amount || 0, 6)` for USD amounts
4. **Rebalance cache** - `src/core/rebalance/cache/rebalance-cache.service.ts:275`
   - Uses `formatUnits(..., 6)` for pending amounts
5. **Rebalance service** - `src/background/cron/rebalance/rebalance.service.ts:197`
   - Divides by `1_000_000n` for USD value calculations

Impact

Tokens with different decimals show incorrect amounts.

Code Snippet

```
// src/api/data/data.service.ts:404
balance: Number(BigInt(balance.balance) / 1_000_000n),  // Assumes 6 decimals

// src/api/data/data.controller.ts:174
volumeInUSD: formatUnits(volume, 6),  // Hardcoded 6 decimals

// src/api/history/history.service.ts:264
amountInUSD: formatUnits(position?.amount || 0, 6),  // Hardcoded 6
```

Tool Used

Manual Review

Recommendation

Replace all hardcoded 6-decimal assumptions with dynamic token decimal values.

Discussion

0xSulpiride

This is fine since we support only USDC for now.

Issue L-21: [Code Review] Dependency supply chain exposure due to unlocked versions and missing registry/script restrictions [RESOLVED]

Source: https://github.com/sherlock-audit/2025-11-zyfai-nov-11th/issues/165

Summary

The project currently installs dependencies directly from the public npm registry without version pinning, registry restrictions, or script-execution hardening. This exposes the codebase to supply chain attacks, such as dependency hijacking, malicious package updates, compromised maintainers, and arbitrary post-install script execution.

Vulnerability Detail

The `package.json` lists dozens of dependencies using loose semantic version ranges (e.g.,^, ~), allowing npm to pull newer, unvetted versions on every install. Combined with unrestricted access to external registries and enabled lifecycle scripts, this significantly increases the attack surface.

Modern supply-chain attacks often involve compromised maintainers, malicious package updates, DNS hijacking of package namespaces, and dependency confusion attacks. Because the repository does not enforce registry restrictions or disable scripts, any transitive package can execute code during installation.

This risk is amplified in JavaScript ecosystems where dependency graphs are deep and complex.

Impact

A malicious or compromised dependency could execute arbitrary code during installation, steal secrets, modify build artifacts, inject backdoors, or compromise CI pipelines. The attack can occur without any change to the project's source code, simply by publishing a malicious patch/minor release under an existing package.

This creates a realistic threat of:

- CI/CD compromise
- Token and environment variable theft
- Deployment of malicious builds
- Persistent infrastructure breaches
- Full takeover of the application environment

Code Snippet

Example from `package.json` where dependency versions are not locked:

```
"dependencies": {
  "@nestjs/common": "^10.4.7",
  "axios": "^1.7.7",
  "lodash": "^4.17.21",
  "typeorm": "^0.3.20",
  ...
}
```

The ^ prefix allows automatic installation of newer, unverified versions.

Tool Used

Manual Review

Recommendation

To mitigate supply-chain risks, implement the following:

1. Restrict Package Sources

Create `.npmrc`:

```
registry=https://registry.npmjs.org/
```

This blocks installations from untrusted registries and prevents dependency confusion attacks.

---

2. Block Lifecycle Scripts

Either enforce the CLI flag in CI:

```
npm install --ignore-scripts
```

Or add to `.npmrc`:

```
ignore-scripts=true
```

3. Enforce Lockfile Integrity

Use:

```
npm ci
```

This ensures installations strictly follow `package-lock.json`.

4. Delay Newly Published Package Adoption

Since npm lacks Yarn's `minimumReleaseAge`:

- Only update dependencies manually after they have aged in the ecosystem
- Avoid automatic upgrades
- Consider using a malware scanner such as: https://github.com/ValkyriSecurity/npm-malware-scanner

5. (Optional) Consider migration to Yarn Berry in the future

Yarn provides:

- `networkSettings` registry lockdown
- `enableScripts: false`

This would allow stronger supply-chain controls.

Discussion

0xSulpiride

Fixed - https://github.com/0xSulpiride/zyfai-backend/commit/6ebef13ae1a2efda08fe283fb0489c24a3ae8b43

defsec

Fix is verified. Dependencies are now locked.

Issue L-22: [Code Review] Missing smart wallet ownership validation [RESOLVED]

Source: https://github.com/sherlock-audit/2025-11-zyfai-nov-11th/issues/166

Summary

The user update endpoint allows authenticated users to set any smart wallet address without cryptographic proof of ownership. The system only checks if the address is already claimed by another user, but does not verify that the requesting user actually controls the wallet. This enables attackers to claim unclaimed wallet addresses, preventing legitimate owners from associating their wallets and potentially accessing data associated with those wallets.

Vulnerability Detail

Location:

- Endpoint: `PATCH /api/v1/users/me`
- Service: `src/api/user/user.service.ts:89-101`
- Controller: `src/api/user/user.controller.ts:80-92`

The `update` method allows authenticated users to set a `smartWallet` address if their account doesn't already have one. The validation logic only checks if the wallet address is already associated with another user account, it does not require cryptographic proof that the user controls the wallet address.

Endpoint Details:

- **Path:** `/api/v1/users/me`
- **Method:** `PATCH`
- **Authentication:** Required (JWT Bearer token via global `AuthGuard`)
- **Authorization:** None - any authenticated user can set any unclaimed wallet address

Root Cause:

The system assumes that if a wallet address is not already claimed, the user setting it must own it. This is a trust assumption without cryptographic verification. The code performs a simple database lookup to check for existing associations, but never requires the user to prove ownership through a signature.

Impact

Attackers can claim wallet addresses before legitimate owners register.

Code Snippet

```
// src/api/user/user.service.ts:89-101
async update(id: Uuid, updateUserDto: UpdateUserReqDto) {
  const user = await this.userRepository.findOneByOrFail({ id });

  if (updateUserDto.smartWallet && !user.smartWallet) {
    //  Only checks if wallet is already taken, not if user owns it
    if (
      await this.userRepository.findOne({
        where: { smartWallet: getAddress(updateUserDto.smartWallet) },
      })
    ) {
      throw new BadRequestException('Smart wallet already exists');
    }
    //  No cryptographic proof required - user can claim any unclaimed wallet
    user.smartWallet = getAddress(updateUserDto.smartWallet);
  }
  // ... rest of update logic
}
```

Tool Used

Manual Review

Recommendation

Add signature verification when setting smart wallet address.

Discussion

0xSulpiride

yes, the problem is that the wallet is not deployed yet at the time of sign up, but I think I should be able to calculate deterministic address of the wallet in the backend, instead of relying on the frontend

Issue L-23: [Code Review] Session key signature replay vulnerability [RESOLVED]

Source: https://github.com/sherlock-audit/2025-11-zyfai-nov-11th/issues/168

Summary

Session key creation accepts signatures without checking if they have been previously used. An attacker who intercepts or obtains a valid signature can replay it to create additional session keys.

Vulnerability Detail

```
async addSessionKey(userId: Uuid, body: AddSessionKeyReqDto) {
  // ... user validation ...

  const { permissionId, permissionEnableHash, customHash } =
    await this.getSessionHash(user, body.nonces.map((nonce) => BigInt(nonce)));
  const { hash, nonces } = body;

  // Signature verification
  let isValidSignature = false;
  if (user.walletType === WalletType.SafeL2) {
    isValidSignature = await publicClient.verifyMessage({
      message: { raw: permissionEnableHash as `0x${string}` },
      signature: hash as `0x${string}`,
      address: user.smartWallet as Address,
    });
  }

  if (!isValidSignature) {
    throw new BadRequestException('Invalid signature');
  }

  // @audit No check if this signature was already used
  // Attacker can replay the same signature multiple times

  const newSessionKey = new SessionKeyEntity({
    expiresAt: new Date(Date.now() + 1000 * 60 * 60 * 24 * 365),
    hash,  // Signature stored but not checked for uniqueness
    // ...
  });

  await this.sessionKeyRepository.save(newSessionKey);
}
```

Impact

Compromised signatures remain valid indefinitely.

Tool Used

Manual Review

Recommendation

Consider checking If the signature is already used.

Discussion

0xSulpiride

Fixed - https://github.com/0xSulpiride/zyfai-backend/commit/f08186ed9df98ba9e7ac511babd0933714cb3ece

Issue L-24: [Code Review] Session key expiration not enforced [RESOLVED]

Source: https://github.com/sherlock-audit/2025-11-zyfai-nov-11th/issues/169

Summary

Session keys have an expiration date set during creation, but the expiration is never validated when executing transactions. Expired session keys can continue to execute transactions indefinitely, violating the intended security policy and allowing stale permissions to remain active.

Vulnerability Detail

Location:

- `src/api/session-keys/session-keys.service.ts:232-424` - `estimateAndSenderUserOperation` method
- `src/api/session-keys/session-keys.service.ts:577-596` - `getActiveSessionKey` method

Root Cause:

The system sets an expiration date (`expiresAt`) when creating session keys, but this expiration is never checked:

1. `getActiveSessionKey` only filters by `isActive: true` and `customHash`, ignoring `expiresAt`
2. `estimateAndSenderUserOperation` accepts a session key parameter but never validates its expiration date
3. No validation occurs anywhere in the transaction execution path.

Impact

Session keys that have passed their expiration date can still execute transactions.

Code Snippet

```
// src/api/session-keys/session-keys.service.ts:577-596
async getActiveSessionKey(user: UserEntity) {
  const sessions = await this.getConfig(user.id);
  const customHash = this.hashSessionConfig(sessions);
  const sessionKey = await this.sessionKeyRepository.findOneBy({
```

```
    customHash,
    isActive: true,  //  Only checks isActive, not expiration
    //  No check for expiresAt > new Date()
  });
  return sessionKey;
}
```

```
// src/api/session-keys/session-keys.service.ts:232-424
async estimateAndSenderUserOperation(
  user: UserEntity,
  chain: SupportedChain,
  sessionKey: SessionKeyEntity,  //  No expiration validation
  calls: unknown[],
  walletTransaction: WalletTransactionMultiPositionEntity,
) {
  // ... transaction preparation and execution ...
  //  No check: if (sessionKey.expiresAt < new Date()) throw error
}
```

Session Key Creation:

```
// src/api/session-keys/session-keys.service.ts:213-223
const newSessionKey = new SessionKeyEntity({
  expiresAt: new Date(Date.now() + 1000 * 60 * 60 * 24 * 365), // 1 year
  hash,
  user,
  signer: signerAddress,
  // ... other fields
});
//  Expiration is set but never enforced
```

Tool Used

Manual Review

Recommendation

Add expiration check in `getActiveSessionKey`.

Discussion

0xSulpiride

Fixed - https://github.com/0xSulpiride/zyfai-backend/commit/43d2aa363a2656b6153f79963aa7401c75f68177

Issue L-25: [Code Review] SQL query logging in production [RESOLVED]

Source: https://github.com/sherlock-audit/2025-11-zyfai-nov-11th/issues/171

Summary

SQL queries and their parameters are logged in production, potentially exposing sensitive data such as user IDs, wallet addresses, API keys, and other confidential information. This violates security best practices and can lead to data breaches if log files are compromised or accessed by unauthorized personnel.

Vulnerability Detail

Location: `src/utils/typeorm-custom-logger.ts:25-36`

The TypeORM custom logger logs all SQL queries along with their parameters to the application logs. In production environments, this can expose sensitive information that should not be logged, including:

- User identifiers (UUIDs, wallet addresses)
- API keys or tokens passed as query parameters
- Personal information or sensitive business data
- Internal system identifiers

The logger uses the `log` level for all queries, meaning they are logged regardless of environment. There is no sanitization or filtering of sensitive parameters before logging.

Vulnerable Code:

```
logQuery(query: string, parameters?: any[], queryRunner?: QueryRunner) {
  const sql =
    query +
    (parameters && parameters.length
      ? ' -- PARAMETERS: ' + this.stringifyParams(parameters)
      : '');
  this.logger.log(`query: ${sql}`);  // @audit Sensitive data logged
}
```

Impact

Sensitive data such as user IDs, wallet addresses, and API keys may be exposed in log files.

Code Snippet

```
// src/utils/typeorm-custom-logger.ts:25-36
logQuery(query: string, parameters?: any[], queryRunner?: QueryRunner) {
  const sql =
    query +
    (parameters && parameters.length
      ? ' -- PARAMETERS: ' + this.stringifyParams(parameters)
      : '');
  this.logger.log(`query: ${sql}`);  // @audit Sensitive data in logs
}
```

Tool Used

Manual Review

Recommendation

Disable SQL query logging in production environments and implement parameter sanitization for development environments.

Discussion

0xSulpiride

we will need this for debugging prod incidents though. What would be the best way forward on this? Also, I think we don't have any sensitive data that would make sense to hide from the logs

defsec

Hi @0xSulpiride , can we make it only debugging log? Instead of by default logging, making it managed by "debug" flag can be much better.

0xSulpiride

Fixed - https://github.com/0xSulpiride/zyfai-backend/commit/6deee63452f6c3a6159f6e2a5bb27d27095ebdd3

Issue L-26: [Code Review] Out-of-date dependencies [RESOLVED]

Source: https://github.com/sherlock-audit/2025-11-zyfai-nov-11th/issues/172

Summary

npm audit identified 96 vulnerabilities in project dependencies: 5 low severity, 57 moderate severity, and 34 high severity. These vulnerabilities affect multiple packages including @merkl/api, vuepress, @nestjs-modules/mailer, @nestjs/swagger, nodemailer, and @nestjs/cli. The vulnerabilities include uncontrolled resource consumption, prototype pollution, regular expression denial of service (REDoS), and arbitrary file write capabilities.

Vulnerability Detail

Location: `package.json` and dependency tree

Audit Results:

- **Total Vulnerabilities:** 96
- **High Severity:** 34
- **Moderate Severity:** 57
- **Low Severity:** 5

Critical Vulnerabilities:

1. braces <3.0.3 (High Severity)

Affected Package: `@merkl/api` (via `tscpaths` → `globby` → `fast-glob` → `micromatch` → `braces`)

Issue: Uncontrolled resource consumption vulnerability in braces package **Advisory:** https://github.com/advisories/GHSA-grv7-fg5c-xmjg

Impact:

- DoS through resource exhaustion
- Affects `@merkl/api@0.9.15 - 0.21.47`
- Fix requires breaking change to `@merkl/api@1.6.36`

2. esbuild <=0.24.2 (Moderate Severity)

Affected Package: `vuepress` and related plugins (via `@vuepress/cli` → `esbuild`)

Issue: esbuild enables any website to send requests to the development server and read the response **Advisory:** https://github.com/advisories/GHSA-67mh-4wv8-2f99

Impact:

- Development server security vulnerability
- Affects all vuepress packages (2.0.0-alpha.1+)
- Fix requires breaking change to `vuepress-plugin-md-enhance@1.30.0`

3. html-minifier (High Severity)

Affected Package: `@nestjs-modules/mailer` (via `mjml` → `html-minifier`)

Issue: kangax html-minifier REDoS (Regular Expression Denial of Service) vulnerability **Advisory:** https://github.com/advisories/GHSA-pfq8-rq6v-vf5m

Impact:

- DoS through regular expression processing
- Affects `@nestjs-modules/mailer >=1.4.0`
- Fix requires breaking change to `@nestjs-modules/mailer@2.0.1`

4. js-yaml <4.1.1 (Moderate Severity)

Affected Packages: `@nestjs/swagger`, `jest`, `gray-matter` (via vuepress)

Issue: js-yaml has prototype pollution in merge («) operator **Advisory:** https://github.com/advisories/GHSA-mh29-5h37-fv8m

Impact:

- Prototype pollution vulnerability
- Affects `@nestjs/swagger@5.3.0-next.1 - 11.2.1`
- Affects jest testing framework
- Fix requires breaking change to `@nestjs/swagger@11.2.2`

5. nodemailer <7.0.7 (Moderate Severity)

Affected Package: `nodemailer` (via `@nestjs-modules/mailer` → `preview-email`)

Issue: Email to an unintended domain can occur due to Interpretation Conflict **Advisory:** https://github.com/advisories/GHSA-mm7p-fcc7-pg87

Impact:

- Email delivery to unintended domains
- Security risk in email functionality
- Fix requires breaking change to `nodemailer@7.0.10`

6. tmp <=0.2.3 (High Severity)

Affected Package: `@nestjs/cli` (via `@angular-devkit/schematics-cli` → `inquirer` → `exter nal-editor` → `tmp`)

Issue: tmp allows arbitrary temporary file / directory write via symbolic link `dir` parameter **Advisory:** https://github.com/advisories/GHSA-52f5-9888-hmc6

Impact:

- Arbitrary file write vulnerability
- Affects `@nestjs/cli@2.0.0-rc.1 - 10.4.9`
- Fix requires breaking change to `@nestjs/cli@11.0.10`

Impact

Multiple vulnerabilities can lead to denial of service.

Code Snippet

Vulnerable Dependencies:

```
{
  "dependencies": {
    "@merkl/api": "^0.21.47",  //  Vulnerable to braces DoS
    "@nestjs-modules/mailer": "^2.0.2",  //  Vulnerable to html-minifier REDoS,
    ↪  nodemailer
    "@nestjs/swagger": "^8.0.5",  //  Vulnerable to js-yaml prototype pollution
    "nodemailer": "^6.9.16",  //  Email domain interpretation conflict
    // ...
  },
  "devDependencies": {
    "@nestjs/cli": "^10.4.5",  //  Vulnerable to tmp arbitrary file write
    "jest": "30.0.0-alpha.6",  //  Vulnerable to js-yaml prototype pollution
    "vuepress": "2.0.0-rc.17",  //  Vulnerable to esbuild dev server issue
    // ...
  }
}
```

Audit Output:

```
96 vulnerabilities (5 low, 57 moderate, 34 high)

Critical vulnerabilities:
- braces <3.0.3 (High) - Uncontrolled resource consumption
- esbuild <=0.24.2 (Moderate) - Development server exposure
- html-minifier (High) - REDoS vulnerability
- js-yaml <4.1.1 (Moderate) - Prototype pollution
- nodemailer <7.0.7 (Moderate) - Email domain interpretation
- tmp <=0.2.3 (High) - Arbitrary file write
```

Tool Used

Manual Review

Recommendation

Consider upgrading the dependencies.

Discussion

0xSulpiride

This one is going to take some time. I `pnpm audit fix` (we're using pnpm btw) but it breaks the backend.

Issue L-27: [Smart Contract] ERC-7201 Namespaced Storage [RESOLVED]

Source: https://github.com/sherlock-audit/2025-11-zyfai-nov-11th/issues/173

Summary

The contract uses the legacy `__gap` storage pattern instead of the modern ERC-7201 namespaced storage pattern introduced in OpenZeppelin Contracts 5.0. While the current implementation works, it's harder to maintain during upgrades.

Vulnerability Detail

Location: `src/module/GuardedExecModuleUpgradeable.sol:61-65`

Issue: The contract uses the legacy storage gap pattern (`uint256[50] private __gapGuardedExecModule`) instead of the modern ERC-7201 namespaced storage pattern that was introduced in OpenZeppelin Contracts 5.0. The project is already using OpenZeppelin Contracts 5.4.0, which supports the new pattern.

Current Implementation (Legacy Pattern):

```
/**
 * @notice Storage gap for future variables in upgrades
 * @dev Prevents storage layout collisions in future upgrades. Reserved 50 storage
 ↪  slots.
 */
uint256[50] private __gapGuardedExecModule;
```

Why This Is a Problem:

According to OpenZeppelin Contracts 5.0 announcement, the legacy `__gap` pattern has several issues:

1. **Storage Layout Collisions**: The old pattern requires careful manual tracking of storage slots across all inherited contracts. If any parent contract adds variables, the gap must be manually adjusted.
2. **Error-Prone**: Developers must manually calculate and maintain gaps for each inherited contract, making it easy to miss collisions.
3. **Inflexible**: Adding new variables requires reducing the gap size, which can be forgotten or miscalculated.
4. **No Tooling Support**: The old pattern doesn't integrate well with automated storage layout validation tools.

The New ERC-7201 Pattern Benefits:

1. **Namespaced Storage**: Each contract gets its own storage namespace, preventing collisions automatically.
2. **Tooling Integration**: OpenZeppelin Defender and other tools can automatically validate storage layouts.
3. **Safer Upgrades**: Variables can be added without worrying about parent contract storage changes.
4. **Standardized**: ERC-7201 is a standardized approach that the ecosystem is adopting.

Impact

Developers must manually track and adjust gaps across all inherited contracts.

Code Snippet

```
// src/module/GuardedExecModuleUpgradeable.sol:61-65
/**
 * @notice Storage gap for future variables in upgrades
 * @dev Prevents storage layout collisions in future upgrades. Reserved 50 storage
  ↪  slots.
 */
uint256[50] private __gapGuardedExecModule;
```

Tool Used

Manual Review

Recommendation

Migrate to ERC-7201 Namespaced Storage Pattern.

According to the OpenZeppelin Contracts 5.0 announcement:

> "Previously, upgradeable contracts used to have a `__gap` variable in storage between inherited contracts in order to reserve space for supporting new variables added between upgrades. But, in this new version storage locations are namespaced using ERC-7201 so that variables can be added without compromising previously stored variables.
> By scoping variables by namespace, each inherited contract will keep its own storage pointer that can easily hold more values after a subsequent upgrade."

Discussion

sunnyRK

Fixed: https://github.com/PaulDeFi/zyfai-executor-module/commit/0452e9ef9d0281e09cbd8fea6edd4ea60644416c Can you check once @defsec ?

defsec

The fix looks good to me. Thank you!

defsec

Storage layout comparison is also passed.

Issue L-28: [API] Whitelist Bypass via HTTP Response Manipulation [RESOLVED]

Source: https://github.com/sherlock-audit/2025-11-zyfai-nov-11th/issues/175

Summary

The backend whitelist validation endpoint returns a JSON field `isWhitelisted` that directly determines access decisions on the client side. Because the result is fully controlled by the HTTP response from the server and the application appears to rely on this field on the *frontend* for authorization, an attacker can bypass whitelist checks by intercepting and modifying the HTTP response before it reaches the browser.

Vulnerability Detail

The endpoint:

```
GET /api/v2/whitelist/check?walletAddress=<address>
```

returns:

```
{
  "status": "success",
  "data": {
    "walletAddress": "...",
    "isWhitelisted": false
  }
}
```

If the frontend simply checks this value to enable access (e.g., by checking `isWhitelisted === true`), then an attacker can use a proxy (Burp Suite, MitMProxy, TamperDev, etc.) to rewrite the server's JSON response to:

```
{"status":"success","data":{"walletAddress":"0x...","isWhitelisted":true}}
```

The application will then treat the attacker as whitelisted even though the backend never authorized them. This indicates the whitelist logic is **not enforced server-side**, making the control ineffective and easily bypassable.

Impact

An attacker can gain access to whitelist-protected features.

Code Snippet

Example manipulated response by attacker:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "status": "success",
  "data": {
    "walletAddress": "0x9eF93EfD2e389aBf2Fee728084CE29bf60258f36",
    "isWhitelisted": true
  }
}
```

Since the frontend trusts this value, the attacker becomes “whitelisted”.

Tool Used

Manual Review

Recommendation

- **Move whitelist enforcement to backend logic.** Backend must enforce authorization checks independently of any client-side flag.
- **Do not trust `isWhitelisted` from frontend-editable sources.**
- **Ensure sensitive actions (claim, mint, buy, stake, etc.) require server-side verification of the user’s whitelist status.**
- Optionally, **sign whitelist status with a server-generated non-forgeable signature**, e.g.:

```
isWhitelisted: true
signature: <serverSigned(validUntil, walletAddress)>
```

and validate signature on the backend before any privileged action.

Discussion

0xSulpiride

This was a temporary check when we had a beta test of a new feature. As of now, it's disabled both in backend and frontend. But we will keep this in my mind when something like this again in future

Issue L-29: [Smart Contract] Missing state validation in whitelist operations [RESOLVED]

Source: https://github.com/sherlock-audit/2025-11-zyfai-nov-11th/issues/176

Summary

The `_addToWhitelist` and `_removeFromWhitelist` functions in TargetRegistry do not validate the current whitelist state before modifying it. While these functions are external (prefixed with _ by naming convention) and can only be called by the TimelockController, they allow redundant operations that waste gas and emit duplicate events.

Vulnerability Detail

Location: `src/registry/TargetRegistry.sol:553-570`

The `_addToWhitelist` and `_removeFromWhitelist` functions are external functions (despite the _ prefix naming convention) that can only be called by the TimelockController. However, they do not check the current state before modifying the whitelist:

1. `_addToWhitelist` sets `whitelist[target][selector] = true` without checking if it's already `true`
2. `_removeFromWhitelist` sets `whitelist[target][selector] = false` without checking if it's already `false`

This allows:

- Adding already-whitelisted entries (idempotent but wasteful)
- Removing already-removed entries (idempotent but wasteful)
- No state consistency validation

Note: These functions are external (not internal) but use the _ prefix naming convention. They are protected by `require(msg.sender == address(timelock))` and can only be called by the TimelockController after the timelock delay expires.

Impact

Unnecessary state changes when the target+selector is already in the desired state.

Code Snippet

```
// src/registry/TargetRegistry.sol:553-570
/**
```

```
 * @notice Add to whitelist (called by TimelockController after delay)
 * @dev Only callable by TimelockController. This is the callback function executed
  ↪  after
 * timelock expires.
 * @param target The contract address to add to whitelist
 * @param selector The function selector to add to whitelist
 */
function _addToWhitelist(address target, bytes4 selector) external {
    require(msg.sender == address(timelock), "Only timelock");
    whitelist[target][selector] = true;  //  No check if already true
    emit TargetSelectorAdded(target, selector);
}

/**
 * @notice Remove from whitelist (called by TimelockController after delay)
 * @dev Only callable by TimelockController. This is the callback function executed
  ↪  after
 * timelock expires.
 * @param target The contract address to remove from whitelist
 * @param selector The function selector to remove from whitelist
 */
function _removeFromWhitelist(address target, bytes4 selector) external {
    require(msg.sender == address(timelock), "Only timelock");
    whitelist[target][selector] = false;  //  No check if already false
    emit TargetSelectorRemoved(target, selector);
}
```

Tool Used

Manual Review

Recommendation

Add state validation before modifying the whitelist to prevent redundant operations.

Discussion

sunnyRK

can you confirm this one @defsec ? https://github.com/PaulDeFi/zyfai-executor-module/commit/98ae98acd8e56079df881bcd43782608bf88cd4c

defsec

The commit looks good.

Issue L-30: [Smart Contract] Incorrect function naming convention for external functions [RESOLVED]

Source: https://github.com/sherlock-audit/2025-11-zyfai-nov-11th/issues/178

Summary

The `_addToWhitelist` and `_removeFromWhitelist` functions in TargetRegistry are declared as `external` but use the _ prefix naming convention typically reserved for `internal` or `private` functions. This violates Solidity naming conventions and can cause confusion about function visibility and access control. These functions should be renamed to `addToWhitelist` and `removeFromWhitelist` to reflect their external visibility.

Vulnerability Detail

Location: `src/registry/TargetRegistry.sol:553-570`

The functions `_addToWhitelist` and `_removeFromWhitelist` are declared as `external` functions but use the underscore prefix (_) which is a Solidity naming convention typically used for:

- `internal` functions
- `private` functions
- Internal helper functions

Impact

The _ prefix is reserved for internal/private functions in Solidity best practices.

Code Snippet

```
// src/registry/TargetRegistry.sol:553-570
/**
 * @notice Add to whitelist (called by TimelockController after delay)
 * @dev Only callable by TimelockController. This is the callback function executed
 ↪  after
 * timelock expires.
 * @param target The contract address to add to whitelist
 * @param selector The function selector to add to whitelist
 */
function _addToWhitelist(address target, bytes4 selector) external {  // External
 ↪  but uses _ prefix
    require(msg.sender == address(timelock), "Only timelock");
```

```
    whitelist[target][selector] = true;
    emit TargetSelectorAdded(target, selector);
}

/**
 * @notice Remove from whitelist (called by TimelockController after delay)
 * @dev Only callable by TimelockController. This is the callback function executed
  ↪  after
 * timelock expires.
 * @param target The contract address to remove from whitelist
 * @param selector The function selector to remove from whitelist
 */
function _removeFromWhitelist(address target, bytes4 selector) external {  //
 ↪  External but uses _ prefix
    require(msg.sender == address(timelock), "Only timelock");
    whitelist[target][selector] = false;
    emit TargetSelectorRemoved(target, selector);
}
```

Tool Used

Manual Review

Recommendation

Rename the external functions to remove the underscore prefix.

Discussion

sunnyRK

can you confirm this @defsec ? https://github.com/PaulDeFi/zyfai-executor-module/commit/4350a408b4b45c338ac49e2c2c62dc3b45c0b001

defsec

The fix is confirmed.

Issue L-31: [API] GraphQL Parser Error Disclosure via Unhandled Malformed Query (https://utility-indexer.zyf.ai/) [RESOLVED]

Source: https://github.com/sherlock-audit/2025-11-zyfai-nov-11th/issues/183

Summary

The GraphQL endpoint leaks detailed parser error messages when malformed or intentionally corrupted queries are submitted. This behavior reveals internal error-handling mechanisms and may expose implementation details that can assist attackers during reconnaissance or exploitation phases.

Vulnerability Detail

During testing, submitting a malformed GraphQL query containing invalid syntax triggered a verbose error message from the backend. Specifically, the server returned a **GraphQL_PARSE_FAILED** exception along with precise location data (`line` and `column`) and parsing expectations.

Example response observed:

- The server disclosed internal parsing logic
- Returned the exact failure location
- Revealed the parsing engine's behavior regarding quoting rules
- Confirmed the underlying GraphQL engine in use

These detailed error messages assist attackers in systematically crafting payloads for:

- Denial-of-service attempts (high-volume malformed queries)
- Query complexity attacks
- Schema enumeration
- Injection payload tuning

Properly hardened GraphQL endpoints should standardize or suppress internal error details to minimize information disclosure.

Impact

An attacker can use this information disclosure to:

- Enumerate the schema more effectively

- Craft targeted malicious queries
- Identify parsing weaknesses or exploitable patterns
- Generate automated payloads tuned to bypass filters
- Accelerate the discovery of deeper logical or injection vulnerabilities

Although this issue alone does not constitute direct compromise, it significantly aids reconnaissance and reduces the effort required to escalate to more impactful attacks.

Code Snippet

URL : https://utility-indexer.zyf.ai/

Request:

```
{'query': 'query { query_4f4722ea: test_table_aggregated { max {id id id id id id
↪  id id id id} } }'}
```

Response (excerpt):

```
{
  "errors": [
    {
      "message": "Syntax Error: Unexpected single quote character ('), did you mean
      ↪  to use a double quote (\")?",
      "locations": [
        {
          "line": 32,
          "column": 2
        }
      ],
      "extensions": {
        "code": "GRAPHQL_PARSE_FAILED"
      }
    }
  ]
}
```

Tool Used

Manual Review

Recommendation

- Implement **generic error messages** for GraphQL parsing failures (e.g., “Invalid query format”).

- Avoid returning detailed parse context (line/column numbers, parser expectations, internal exception types).
- Ensure the server is configured to **suppress GraphQL engine-specific error traces**.
- Add an **API gateway or WAF rule** to block malformed or excessively nested/complex queries.
- Enable **query schema whitelisting** or **operation name validation** where possible.

Discussion

0xSulpiride

this is an indexer created by https://ponder.sh/ - doesn't look like I can configure graphql here

Issue L-32: [API] GraphQL Introspection Enabled Leading to Full Schema Disclosure

Source: https://github.com/sherlock-audit/2025-11-zyfai-nov-11th/issues/184

Summary

The GraphQL endpoint at `utility-indexer.zyf.ai` has **GraphQL introspection fully enabled**, allowing unauthenticated clients to retrieve the complete schema, type definitions, object structures, scalars, directives, and relationships. This significantly increases the attack surface by revealing internal API structure and logic.

Vulnerability Detail

During manual testing, a standard GraphQL `__schema` introspection query successfully returned the full schema. The response included:

- All type definitions
- Custom scalars (`JSON`, `BigInt`, etc.)
- Object types, interfaces, enums, and input types
- Field names, arguments, return types
- Deprecation metadata
- Directives
- Query entry points and structure

This level of visibility enables an attacker to conduct highly targeted attacks, such as:

- Enumerating every available query and mutation
- Identifying sensitive fields and parameters
- Detecting undocumented or internal API functionality
- Crafting injection payloads tailored to the schema
- Mapping relationships for excessive data exposure attempts

Exposing the entire API specification without authentication is an industry-recognized weakness.

Impact

An attacker can gain complete knowledge of the internal GraphQL schema, enabling:

- Rapid reconnaissance for deeper vulnerabilities

- Crafting precise and complex malicious queries
- Identification of sensitive internal data models
- Enumeration of hidden fields or administrative routes
- Targeting of nested queries for DoS, mass data extraction, or injection attacks

Although schema disclosure is not a direct exploit, it drastically reduces attacker effort and increases likelihood of further compromise.

Code Snippet

Request:

```
query IntrospectionQuery {
  __schema {
    queryType { name }
    mutationType { name }
    types {
      name
      kind
      description
      fields { name }
    }
  }
}
```

Response excerpt:

```
{
  "data": {
    "__schema": {
      "queryType": { "name": "Query" },
      "mutationType": null,
      "types": [
        {
          "kind": "SCALAR",
          "name": "JSON",
          "description": "The `JSON` scalar type represents JSON values..."
        },
        {
          "kind": "OBJECT",
          "name": "PageInfo",
          "fields": [
            { "name": "hasNextPage" },
            ...
          ]
        }
      ]
```

```
    }
  }
}
```

This confirms that full introspection is exposed.

Tool Used

Manual Review (GraphQL client / browser-based GraphiQL)

Recommendation

- **Disable GraphQL introspection** in production environments.
  - For Apollo Server: `introspection: false`
  - For Yoga / Mercurius / GraphQL-JS: disable introspection based on environment variables.
- Implement **authentication and authorization** before serving any GraphQL operations.

Discussion

0xSulpiride

same as here - https://github.com/sherlock-audit/2025-11-zyfai-nov-11th/issues/183 allowbreak #issuecomment-3562696991 - ponder doesn't allow configurating graphql

defsec

Hi @0xSulpiride thank you for the answer, you are definitely right but I'm trying to find a way to block these. Thank you!

Disclaimers

Sherlock does not provide guarantees nor warranties relating to the security of the project.

Usage of all smart contract software is at the respective users' sole risk and is the users' responsibility.