Smart Contract Security Audit

For

Pencil Finance

Audit Date: July 5, 2025

Audit Version: v1.0

Contents

1 Executive Summary

This report presents a comprehensive security audit of the Pencil Finance smart contract system. The audit identified multiple security vulnerabilities, including 4 high-risk vulnerabilities, 1 critical vulnerability, 1 medium-risk vulnerability, and 2 low-risk vulnerabilities. The main issues are concentrated in unchecked ERC20 transfer return values, improper access control, and missing boundary checks.

The codebase exhibits solid overall quality, clear business logic, and a high level of security.

1.1 Audit Scope

This audit covers the following smart contracts:

- Treasury.sol - Treasury contract
- SeniorPool.sol - Senior pool contract
- JuniorInterestPool.sol - Junior interest pool contract
- FirstLossPool.sol - First loss pool contract
- Factory.sol - Factory contract
- AssetPool.sol - Asset pool contract
- Funding.sol - Funding contract

1.2 Project Summary

Type	Load
Arch	Eth
CodeBase	Github Link
CommitID	b0c6a9d
FixCommit	Null
Loan	Ethereum Chain
Audit Method	Cross-manual review; Static Analysis

1.3 Risk Statistics

Risk Level	Count	Color	Status
Critical	1	Critical	Fixed
High	4	High	3 Fixed 1Ack
Medium	1	Medium	Acknowledge
Low	2	Low	1 Fixed 1 Ack
Total	**5 Vulnerabilities Fixed And 3 Acknowledged**		

Table 1: Vulnerability risk level distribution

1.4 Audit Approach Summary

This report has been prepared Pencil Financial to discover issues and vulnerabilities in the source code of the Pencil Financial Repo as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Cross Manual Review and Static Analysis techniques The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders
- Thorough line-by-line manual review of the entire codebase by industry expert
- Cross-audit mode of the current code by more than three security engineers.

The security assessment resulted in findings that ranged from critical to informational. We recommend addressing these findings to ensure a high level of security standards and industry practices. We suggest recommendations that could better serve the project from the security perspective

- Testing the smart contracts against both common and uncommon attack vectors;
- Enhance general coding practices for better structures of source codes;
- Add enough unit tests to cover the possible use cases;
- Provide more comments per each function for readability, especially contracts that are verified in public;
- Provide more transparency on privileged activities once the protocol is live

2 Detailed Audit Findings

2.1 Critical Risk Vulnerabilities

Critical Risk

Vulnerability ID: PENCIL-001
Title: Factory contract can control owner and systemConfig
Risk Level: Critical
Status: Fixed
Affected Contract: Factory.sol

Description: The *initialize* function in the Factory contract has no protection mechanisms, allowing the owner to be arbitrarily modified. An attacker can call the *initialize* function to reset the owner, then use owner privileges to modify the $systemConfigaddress$.

```
/**
 * @dev 初始化工厂合约
 * @param _owner 合约所有者地址
 * 设置合约的初始所有者
 */
function initialize(address _owner) external override {
    _transferOwnership(_owner);
}
```

Figure 1: Factory initialize function without protection

After modification, owner privileges can be used to modify the $systemConfig$ address:

```
/**
 * @dev 设置系统配置合约地址
 * @param newSystemConfig 新的系统配置合约地址
 * 注意：该方法保留 onlyOwner 修饰符，因为系统配置地址是权限管理的基础，
 * 如果交由 SystemConfig 合约中的管理员控制可能导致循环依赖问题
 */
function setSystemConfig(
    address newSystemConfig
) external override onlyOwner {
    _setSystemConfig(newSystemConfig);
}
```

Figure 2: Using owner privileges to modify systemConfig address

Use the following code for verification:

```
// forge test --match-test testFactoryInitialize -vvv
function testFactoryInitialize() public {
    // 测试Factory合约的initialize函数
    address owner = address(this);
    factory.initialize(owner);
    address systemConfigAddress = factory.systemConfig();
    console.log("Factory System Config Address: %s", systemConfigAddress);
    factory.setSystemConfig(address(0xabcdef));
    address newSystemConfigAddress = factory.systemConfig();
    console.log("New Factory System Config Address: %s", newSystemConfigAddress);
}
```

Figure 3: Verification code for the vulnerability

After running, systemConfig was successfully modified.

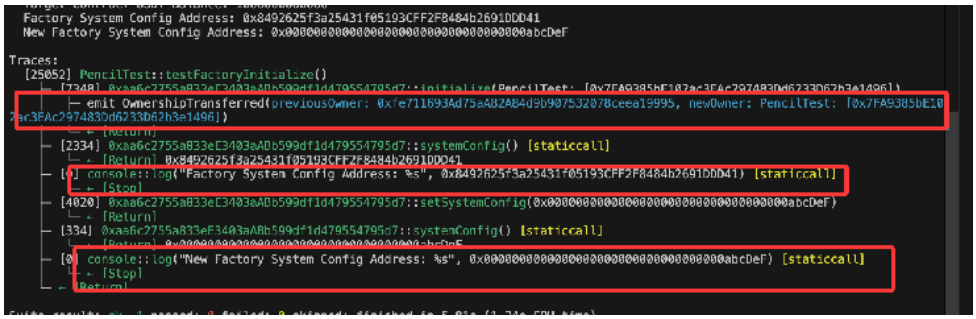


Figure 4: The attack code schedule result to modify systemConfig address

Technical Details:

- The initialize function lacks access control
- Can be called repeatedly leading to malicious owner modification
- After modifying the owner, systemConfig can be further controlled

Impact Assessment:

- Attackers can completely control the Factory contract
- May lead to malicious control of the entire system
- User funds face serious risks

Remediation Recommendations:

- Initialize owner in the constructor
- Use onlyOwner modifier for initialize function, restricting access to owner only
- Add initialization state checks to prevent repeated initialization

2.2 High Risk Vulnerabilities

High Risk

Vulnerability ID: PENCIL-002
Title: Treasury contract does not validate ERC20 transfer return values
Risk Level: High
Status: Fixed
Affected Contract: Treasury.sol

Description: The withdrawFee function does not validate transfer return values. Some ERC20 contract tokens return false instead of reverting when transfers fail. If return values are not checked, transfers may fail while the entire transaction succeeds.

```
    */
    function withdrawFee(address asset, address to, uint256 amount) external override onlyTreasuryAdmin nonReentrant whe
        require(to != address(0), "Invalid receiver address");
        require(amount > 0, "Amount must be greater than 0");
        IERC20 assetTokenContract = IERC20(asset);
        assetTokenContract.transfer(to, amount);
        emit PlatformFeeWithdrawn(asset, to, amount);
    }
```

Figure 5: Treasury withdrawFee function without transfer validation

Using Foundry to simulate calling this function, creating a token that returns false when transfer fails:

```
contract AttackToken is ERC20 {
    uint8 private constant DECIMALS = 6;

    constructor(string memory name, string memory symbol) ERC20(name, symbol) {
        _mint(msg.sender, 1_000_000 * 10**DECIMALS);
    }

    function mint(address to, uint256 amount) external {
        _mint(to, amount);
    }

    // 在转移token失败的时候，会返回false而不是revert
    function transfer(address to, uint256 value) public virtual override returns (bool) {
        if (to == address(0)) return false;
        if (value > balanceOf(msg.sender)) return false;

        _transfer(msg.sender, to, value);
        return true;
    }
}
```

Figure 6: Foundry simulation of failed transfer scenario

When testing with the following code, a call to the *withdrawFee* function in the vault contract still succeeds even if the vault's balance is only 1000 and the input for *withdrawFee* is 1001.

```
// forge test --match-test testAttackTreasury -vvv
function testAttackTreasury() public {
    // Treasury.sol的withdrawFee函数
    attackToken.transfer(address(treasury), 1000 * 10**6); // 转移1000个ATK到Treasury
    uint balanceTreasuryBefore = attackToken.balanceOf(address(treasury));
    console.log("Treasury ATK Balance Before: %s", balanceTreasuryBefore);
    address treasuryAdmin = systemConfig.treasuryAdmin();
    console.log("Treasury Admin: %s", treasuryAdmin);

    // 模拟treasuryAdmin调用
    vm.startPrank(treasuryAdmin);
    treasury.withdrawFee(address(attackToken), address(this), 1001 * 10**6); // 提取1000个ATK到当前合约
    vm.stopPrank();

    uint balanceTreasuryAfter = attackToken.balanceOf(address(treasury));
    console.log("Treasury ATK Balance After: %s", balanceTreasuryAfter);

}
}
```

Figure 7: The attack code

```
Traces:
  [81301] PencilTest::testAttackTreasury()
    ├─ [29897] AttackToken::transfer(ERC1967Proxy: [0x2e234DAe75C793f67A35089C9d99245E1C58470b], 1000000000 [1e9])
    │   ├─ emit Transfer(from: PencilTest: [0x7FA9385bE102ac3EAc297483Dd6233D62b3e1496], to: ERC1967Proxy: [0x2e234DAe75C793f67A35089C9d99245E1C58
b], value: 1000000000 [1e9])
    │   └─ ← [Return] true
    ├─ [552] AttackToken::balanceOf(ERC1967Proxy: [0x2e234DAe75C793f67A35089C9d99245E1C58470b]) [staticcall]
    │   └─ ← [Return] 1000000000 [1e9]
    ├─ [0] console::log("Treasury ATK Balance Before: %s", 1000000000 [1e9]) [staticcall]
    │   └─ ← [Stop]
    ├─ [2961] 0x8492625f3a25431f05193CFF2F8484b2691DDD41::treasuryAdmin() [staticcall]
    │   └─ ← [Return] 0xfe711693Ad75aA82A84d9b907532078ceea19995
    ├─ [0] console::log("Treasury Admin: %s", 0xfe711693Ad75aA82A84d9b907532078ceea19995) [staticcall]
    │   └─ ← [Stop]
    ├─ [0] VM::startPrank(0xfe711693Ad75aA82A84d9b907532078ceea19995)
    │   └─ ← [Return]
    ├─ [22616] ERC1967Proxy::fallback(AttackToken: [0xF62849F9A0B5Bf2913b396098F7c7019b51A820a], PencilTest: [0x7FA9385bE102ac3EAc297483Dd6233D62b
496], 1001000000 [1.001e9])
    │   ├─ [17800] Treasury::withdrawFee(AttackToken: [0xF62849F9A0B5Bf2913b396098F7c7019b51A820a], PencilTest: [0x7FA9385bE102ac3EAc297483Dd6233D
3e1496], 1001000000 [1.001e9]) [delegatecall]
    │   │   ├─ [251] 0x8492625f3a25431f05193CFF2F8484b2691DDD41::TREASURY_ADMIN_ROLE() [staticcall]
    │   │   │   └─ ← [Return] 0x4ba1c0b393f1850d2175b0f2d7c2d42c3f898b0037de3434dbcee26a67c6df66
    │   │   ├─ [3000] 0x8492625f3a25431f05193CFF2F8484b2691DDD41::hasRole(0x4ba1c0b393f1850d2175b0f2d7c2d42c3f898b0037de3434dbcee26a67c6df66, 0xfe
693Ad75aA82A84d9b907532078ceea19995) [staticcall]
    │   │   │   └─ ← [Return] true
    │   │   ├─ [2688] 0x8492625f3a25431f05193CFF2F8484b2691DDD41::paused() [staticcall]
    │   │   │   └─ ← [Return] false
    │   │   ├─ [689] AttackToken::transfer(PencilTest: [0x7FA9385bE102ac3EAc297483Dd6233D62b3e1496], 1001000000 [1.001e9])
    │   │   │   └─ ← [Return] false
    │   │   ├─ emit PlatformFeeWithdrawn(asset: AttackToken: [0xF62849F9A0B5Bf2913b396098F7c7019b51A820a], to: PencilTest: [0x7FA9385bE102ac3EAc29
3Dd6233D62b3e1496], amount: 1001000000 [1.001e9])
    │   │   └─ ← [Stop]
    │   └─ ← [Return]
    ├─ [0] VM::stopPrank()
    │   └─ ← [Return]
    ├─ [552] AttackToken::balanceOf(ERC1967Proxy: [0x2e234DAe75C793f67A35089C9d99245E1C58470b]) [staticcall]
    │   └─ ← [Return] 1000000000 [1e9]
    ├─ [0] console::log("Treasury ATK Balance After: %s", 1000000000 [1e9]) [staticcall]
    │   └─ ← [Stop]
    └─ ← [Return]
```

Figure 8: The simulated schedule result

Technical Details:

- withdrawFee function does not check ERC20 transfer return values
- Some ERC20 tokens return false instead of reverting on transfer failure
- May lead to false successful transfer events

Remediation Recommendations: Use require to validate transfer return values:

```
require(token.transfer(to, amount), "Transfer failed");
```

High Risk

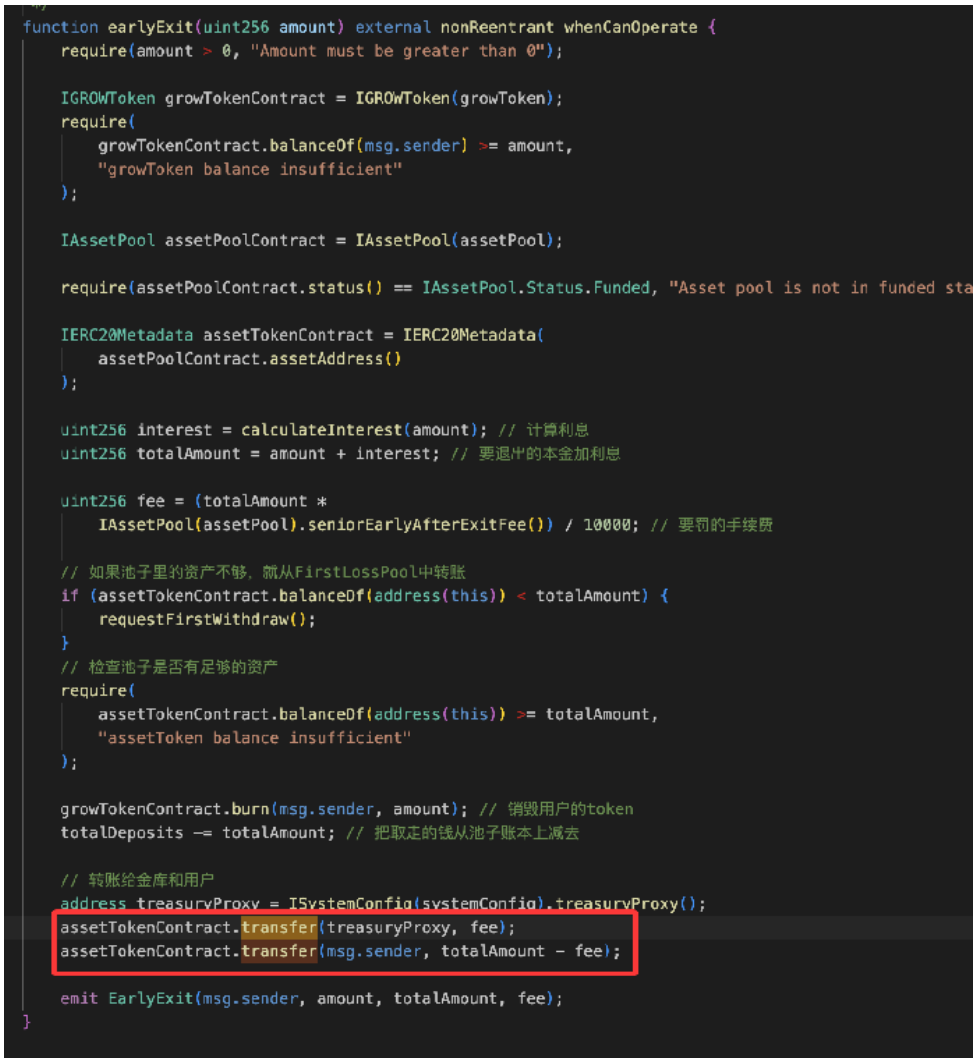
Vulnerability ID: PENCIL-003
Title: SeniorPool contract does not validate ERC20 transfer return values in multiple places
Risk Level: High
Status: Fixed
Affected Contract: SeniorPool.sol

Description: Functions *withdraw* and *earlyExit* do not validate ERC20 *transfer* return values. Some ERC20 contract tokens return false instead of reverting when transfers fail. If return values are not checked, transfers may fail while the entire transaction succeeds.

```
function withdraw(uint256 amount) external nonReentrant whenCanOperate {
    require(amount > 0, "Amount must be greater than 0");
    IAssetPool assetPoolContract = IAssetPool(assetPool);
    IGROWToken growTokenContract = IGROWToken(growToken);
    require(
        assetPoolContract.status() == IAssetPool.Status.Ended,
        "Asset pool is not in ended status"
    );
    require(
        growTokenContract.balanceOf(msg.sender) >= amount,
        "Insufficient balance"
    );

    uint256 totalSupply = growTokenContract.totalSupply(); // 总共还有这么多用户份额
    require(totalSupply > 0, "No total supply available");

    IERC20Metadata assetTokenContract = IERC20Metadata(
        assetPoolContract.assetAddress()
    );

    // 用户应该得到的钱：按当前池子中的总资产，算出每个份额对应的资产数量，再按比例分给用户
    uint256 assetAmount = assetTokenContract.balanceOf(address(this)) * amount / totalSupply;

    // 不管池子里有多少钱，都按比较分给用户，并销毁份额
    growTokenContract.burn(msg.sender, amount);
    totalDeposits -= assetAmount;
    assetTokenContract.transfer(msg.sender, assetAmount);

    emit Withdrawn(msg.sender, amount, assetAmount);
}
```

Figure 9: SeniorPool withdraw function vulnerability

This can cause the *totalDeposits* value to decrease while the actual value remains unchanged:



```
function earlyExit(uint256 amount) external nonReentrant whenCanOperate {
    require(amount > 0, "Amount must be greater than 0");

    IGROWToken growTokenContract = IGROWToken(growToken);
    require(
        growTokenContract.balanceOf(msg.sender) >= amount,
        "growToken balance insufficient"
    );

    IAssetPool assetPoolContract = IAssetPool(assetPool);

    require(assetPoolContract.status() == IAssetPool.Status.Funded, "Asset pool is not in funded sta

    IERC20Metadata assetTokenContract = IERC20Metadata(
        assetPoolContract.assetAddress()
    );

    uint256 interest = calculateInterest(amount); // 计算利息
    uint256 totalAmount = amount + interest; // 要退出的本金加利息

    uint256 fee = (totalAmount *
        IAssetPool(assetPool).seniorEarlyAfterExitFee()) / 10000; // 要罚的手续费

    // 如果池子里的资产不够，就从FirstLossPool中转账
    if (assetTokenContract.balanceOf(address(this)) < totalAmount) {
        requestFirstWithdraw();
    }
    // 检查池子是否有足够的资产
    require(
        assetTokenContract.balanceOf(address(this)) >= totalAmount,
        "assetToken balance insufficient"
    );

    growTokenContract.burn(msg.sender, amount); // 销毁用户的token
    totalDeposits -= totalAmount; // 把取走的钱从池子账本上减去

    // 转账给金库和用户
    address treasuryProxy = ISystemConfig(systemConfig).treasuryProxy();
    assetTokenContract.transfer(treasuryProxy, fee);
    assetTokenContract.transfer(msg.sender, totalAmount - fee);

    emit EarlyExit(msg.sender, amount, totalAmount, fee);
}
```

Figure 10: Impact on totalDeposits calculation

Remediation Recommendations: Use *require* to validate *transfer* return values.

High Risk

Vulnerability ID: PENCIL-004
Title: JuniorInterestPool contract does not validate ERC20 transfer return values
Risk Level: High
Status: Fixed
Affected Contract: JuniorInterestPool.sol

Description: The *claimInterest* function does not validate ERC20 $transfer$ return values. Some ERC20 contract tokens do not revert when a transfer fails; instead, they return false. If the return value is not checked, it could lead to a failed transfer while the overall transaction still succeeds.

```
function claimInterest(uint256 tokenId) external nonReentrant whenCanOperate {
    IJuniorNFT juniorNFTContract = IJuniorNFT(juniorNFT);
    require(juniorNFTContract.ownerOf(tokenId) == msg.sender || msg.sender == assetPool, "Not token owner or asset

    uint256 totalAmount = juniorNFTContract.totalAmount(); // 所有NFT的总金额
    require(totalAmount > 0, "No total amount available");

    uint256 currentAmount = juniorNFTContract.getTokenAmount(tokenId); // 当前NFT的金额
    uint256 interest = (currentAmount * totalDeposits) / totalAmount; // 当前NFT的总利息
    uint256 interestToClaim = interest - tokenInterest[tokenId]; // 当前NFT待领取的利息
    require(interestToClaim > 0, "No interest to claim");

    IAssetPool assetPoolContract = IAssetPool(assetPool);

    IERC20Metadata assetTokenContract = IERC20Metadata(
        assetPoolContract.assetAddress()
    );
    require(assetTokenContract.balanceOf(address(this)) >= interestToClaim, "Insufficient pool balance");

    tokenInterest[tokenId] = interest;
    assetTokenContract.transfer(msg.sender, interestToClaim);

    emit InterestClaimed(msg.sender, tokenId, interestToClaim);
}
```

Figure 11: JuniorInterestPool claimInterest function vulnerability

Remediation Recommendations: Use require to validate $transfer$ return values.

High Risk

Vulnerability ID: PENCIL-005
Title: FirstLossPool contract does not validate ERC20 transfer return values in multiple places
Risk Level: High
Status: Fixed
Affected Contract: FirstLossPool.sol

Description: Functions *repayToSeniorPool* and *withdrawPrincipal* do not validate ERC20 *transfer* return values.

```
*/
function withdrawPrincipal(
    uint256 tokenId
) external nonReentrant whenCanOperate {
    IJuniorNFT juniorNFTContract = IJuniorNFT(juniorNFT);
    require(juniorNFTContract.ownerOf(tokenId) == msg.sender || msg.sender == assetPool, "Not tok
    require(IAssetPool(assetPool).status() == IAssetPool.Status.Ended, "Not repaid");

    IERC20 assetTokenContract = IERC20(
        IAssetPool(assetPool).assetAddress()
    );

    uint256 totalSupply = juniorNFTContract.totalAmount(); // 总共还有这么多用户份额
    require(totalSupply > 0, "No total supply available");

    uint256 amount = juniorNFTContract.getTokenAmount(tokenId); // 用户持有的份额

    // 用户应该得到的钱：按当前池子中的总资产，算出每个份额对应的资产数量，再按比例分给用户
    uint256 assetAmount = assetTokenContract.balanceOf(address(this)) * amount / totalSupply;

    // 不管池子里有多少钱，都按比较分给用户，并销毁份额
    juniorNFTContract.burn(tokenId);
    totalDeposits -= assetAmount;
    assetTokenContract.transfer(msg.sender, assetAmount);

    emit PrincipalWithdrawn(msg.sender, tokenId, amount);
}
```

Figure 12: FirstLossPool repayToSeniorPool function vulnerability

```
*/
function repayToSeniorPool(uint256 amount) external nonReentrant whenCanOperate {
    require(msg.sender == IAssetPool(assetPool).seniorPool(), "Only senior pool can repay");
    require(amount > 0, "Amount must be greater than 0");
    require(totalDeposits >= amount, "Insufficient balance");
    totalDeposits -= amount;
    IERC20 assetTokenContract = IERC20(
        IAssetPool(assetPool).assetAddress()
    );
    assetTokenContract.transfer(IAssetPool(assetPool).seniorPool(), amount);
    emit RepaidToSeniorPool(assetPool, amount);
}

/**
```

Figure 13: FirstLossPool withdrawPrincipal function vulnerability

Remediation Recommendations: Use require to validate transfer return values.

High Risk

Vulnerability ID: PENCIL-006
Title: Unrestricted loop length potential resource exhaustion DoS
Risk Level: High
Status: Acknowledged
Affected Contract: Funding.sol

Description: In *funding.sol*, the *refundSenior* function uses *users.length* as the boundary value to control loop iterations, but the code does not limit individual user minimum investment amounts or total fundraising amounts. This allows batch creation of new addresses with small individual investments, then conducting resource exhaustion DoS attacks at completion.

```
    */
    function refundSenior(
        address[] memory users
    ) external nonReentrant whenCanOperate {
        require(
            IAssetPool(assetPool).status() == IAssetPool.Status.Failed ||
                IAssetPool(assetPool).status() == IAssetPool.Status.Funded,
            "Asset pool must be failed or funded"
        );
        // 获取资产标的 token
        IERC20 token = IERC20(_fundingInfo.assetAddress);

        for (uint256 i = 0; i < users.length; i++) {
            address user = users[i];
            bool isFailed = IAssetPool(assetPool).status() ==
                IAssetPool.Status.Failed;
            uint256 amount = isFailed
                ? userSeniorSubscriptions[user]
                : userSeniorRefundAmount[user];

            if (amount > 0) {
                if (isFailed) {
                    userSeniorSubscriptions[user] = 0;
                    _fundingInfo.seniorAmount -= amount;
                } else {
                    userSeniorRefundAmount[user] = 0;
                    // 注意：在Funded状态下，_fundingInfo.seniorAmount已经在completeFunding中被正
                    // 这里不需要再次减少，因为userSeniorRefundAmount中的金额本来就不包含在当前的se
                }
                // 转账本金给用户
                require(token.transfer(user, amount), "Transfer failed");
                emit Withdrawn(user, assetPool, amount, true);
            }
        }
    }
```

Figure 14: refundSenior function with unrestricted loop length

Remediation Recommendations:

- Limit array size
- Add pagination handling patterns
- Use pull pattern, where users actively claim instead of contract actively sending

2.3 Medium Risk Vulnerabilities

Medium Risk

Vulnerability ID: PENCIL-007
Title: Fees may be manipulated
Risk Level: Medium
Status: Acknowledged
Affected Contract: AssetPool.sol

Description: AssetPool uses block.timestamp to calculate repayment periods. Miners can modify execution and validation clients to make block.timestamp return incorrect time.

```
     * @return 获取需要还款的期数
     */
    function needToCount() public view returns (uint256) {
        uint256 count = (block.timestamp - fundingEndTime) /
            (repaymentPeriod * 86400);

        if (count > repaymentCount) {
            count = repaymentCount;
        }
        return count > 0 ? count : 1;
    }
```

Figure 15: AssetPool using block.timestamp for fee calculation

As long as miners forge a value, such as directly forging fundingEndTime, they can make count equal to 1. The if condition won't trigger, and the final result will only return 1. Since platform fees are calculated by needToCount, hacker miners can pay only one period's fees:

```
// 还有多余的钱，给平台交手续费
if (amount > 0) {
    IFunding.FundingInfo memory fundingInfo = IFunding(fundingContract)
        .fundingInfo();
    uint256 needCount = needToCount();
    uint256 _totalAmount = fundingInfo.seniorAmount +
        fundingInfo.juniorAmount; // 总贷款本金
    uint256 installment = _totalAmount /
        repaymentCount +
        (_totalAmount * repaymentRate) /
        10000; // 每期应该还的本金加利息
    uint256 platformFeeTotalAmount = (installment *
        platformFee *
        needCount) / 10000; // 已到期的应还总手续费
    uint256 feeNeed = platformFeeTotalAmount - platformFeeAmount; // 需要交的手续费

    if (feeNeed > 0) {
        uint256 feeValue = amount > feeNeed ? feeNeed : amount; // 实际要交的手续费
        require(
            assetToken.transferFrom(
                msg.sender,
                ISystemConfig(_systemConfig).treasuryProxy(),
                feeValue
```

Figure 16: Fee manipulation through timestamp forging

Remediation Recommendations: Use block.number or other manipulation-resistant methods to calculate time. Assuming an average block time of 3 seconds, similar methods can be used for calculation:

```solidity
/**
 * @dev 获取需要还款的期数，基于区块号计算
 * @return 需要还款的期数
 */
function needToCount() public view returns (uint256) {
    // 假设出块时间为3秒，则一天的区块数为 86400/3 = 28800 个区块
    uint256 blocksPerDay = 28800;

    // 假设fundingEndTime已经存储为区块号
    uint256 blocksPassed = block.number - fundingEndTime;

    // 计算经过了多少个还款周期
    uint256 count = blocksPassed / (repaymentPeriod * blocksPerDay);

    // 确保不超过总还款期数
    if (count > repaymentCount) {
        count = repaymentCount;
    }

    // 确保至少返回1（保持原函数逻辑）
    return count > 0 ? count : 1;
}
```

Figure 17: Recommended fix using block.number instead of timestamp

2.4 Low Risk Vulnerabilities

Low Risk

Vulnerability ID: PENCIL-008
Title: Missing boundary checks causing function execution anomalies
Risk Level: Low
Status: Acknowledged
Affected Contract: Multiple contracts

Description: When calculating interest, the *calculateInterest* function is called, which calculates time differences between current timestamp and funding end time. However, before calculation, it doesn't check whether current time exceeds the funding end time, which may cause overflow in certain situations, leading to abnormal function logic execution.

```
        IERC20Metadata assetTokenContract = IERC20Metadata(
            assetPoolContract.assetAddress()
        );

        uint256 interest = calculateInterest(amount); // 计算利息
        uint256 totalAmount = amount + interest; // 要退出的本金加利息

        uint256 fee = (totalAmount *
```

Figure 18: calculateInterest function without boundary checks

```
    */
    function calculateInterest(uint256 amount) public view returns (uint256) {
        IAssetPool assetPoolContract = IAssetPool(assetPool);
        // 计算已经过了多少期
        uint256 count = (block.timestamp - assetPoolContract.fundingEndTime()) /
            (assetPoolContract.repaymentPeriod() * 86400);

        if (count > assetPoolContract.repaymentCount()) {
            count = assetPoolContract.repaymentCount();
        }

        uint256 seniorFixedRate = assetPoolContract.seniorFixedRate();

        return (amount * seniorFixedRate * count) / 10000;
    }
```

Figure 19: Potential overflow scenario in time calculation

Remediation Recommendations: Check the relationship between current time and funding end time before calculation: $if(block.timestamp <= fundingEndTime)return$

Low Risk

Vulnerability ID: PENCIL-009
Title: Missing zero address checks
Risk Level: Low
Status: Fixed
Affected Contract: Multiple contracts

Description: Multiple contracts' initialization functions lack zero address checks, including:

- *FirstLossPool* initialization function doesn't check *_juniorNFT* and *_assetPool* addresses
- Factory's multiple set functions lack zero address checks
- SeniorPool's initialize function lacks zero address checks
- JuniorInterestPool's initialize function lacks zero address checks
- Funding's initialize function lacks zero address checks

```
 */
function initialize(address _owner, address _assetPool, address _juniorNFT, a
    __Ownable_init(_owner);
    __ReentrancyGuard_init();
    __UUPSUpgradeable_init();
    __AccessControl_init(_systemConfig);
    juniorNFT = _juniorNFT;
    assetPool = _assetPool;
}
```

Figure 20: FirstLossPool initialization without zero address checks

Factory contract multiple set functions lack zero address checks:

```
function setAssetPoolImplementation(
    address _implementation
) external override onlySystemAdmin {
    assetPoolImplementation = _implementation;
}

/**
 * @dev 设置融资实现合约地址
 * @param _implementation 新的实现合约地址
 * 只有系统管理员可以调用此函数
 */
function setFundingImplementation(
    address _implementation
) external override onlySystemAdmin {
    fundingImplementation = _implementation;
}

/**
 * @dev 设置优先池实现合约地址
 * @param _implementation 新的实现合约地址
 * 只有系统管理员可以调用此函数
 */
function setSeniorPoolImplementation(
    address _implementation
) external override onlySystemAdmin {
    seniorPoolImplementation = _implementation;
}

/**
 * @dev 设置首损池实现合约地址
 * @param _implementation 新的实现合约地址
 * 只有系统管理员可以调用此函数
 */
function setFirstLossPoolImplementation(
    address _implementation
) external override onlySystemAdmin {
    firstLossPoolImplementation = _implementation;
}
```

Figure 21: Factory setFirstLossPoolImplementation and other set functions

```
 */
function setJuniorInterestPoolImplementation(
    address _implementation
) external override onlySystemAdmin {
    juniorInterestPoolImplementation = _implementation;
}

/**
 * @dev 设置 GROW 代币实现合约地址
 * @param _implementation 新的实现合约地址
 * 只有系统管理员可以调用此函数
 */
function setGROWTokenImplementation(
    address _implementation
) external override onlySystemAdmin {
    growTokenImplementation = _implementation;
}

/**
 * @dev 设置次级 NFT 实现合约地址
 * @param _implementation 新的实现合约地址
 * 只有系统管理员可以调用此函数
 */
function setJuniorNFTImplementation(
    address _implementation
) external override onlySystemAdmin {
    juniorNFTImplementation = _implementation;
}

/**
 * @dev 设置系统配置合约地址
 * @param newSystemConfig 新的系统配置合约地址
 * 注意：该方法保留 onlyOwner 修饰符，因为系统配置地址是权限管理的基础，
 * 如果交由 SystemConfig 合约中的管理员控制可能导致循环依赖问题
 */
function setSystemConfig(
    address newSystemConfig
) external override onlyOwner {
    _setSystemConfig(newSystemConfig);
}
```

Figure 22: Factory additional set functions without zero checks

SeniorPool initialization function lacks zero address checks:

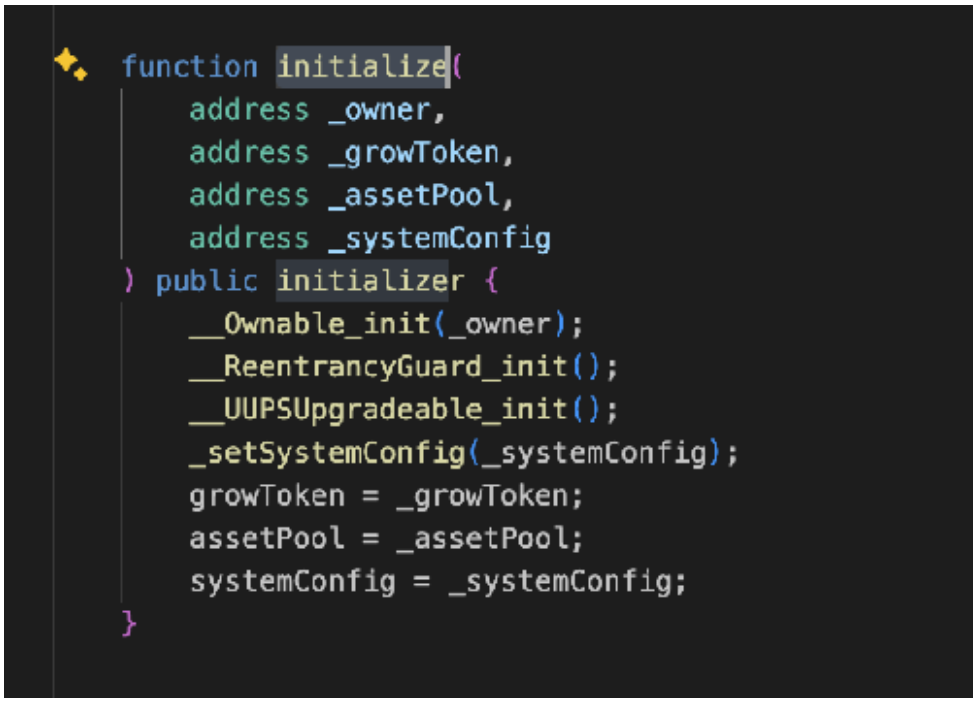


Figure 23: SeniorPool initialize function without zero checks

JuniorInterestPool initialization function lacks zero address checks:

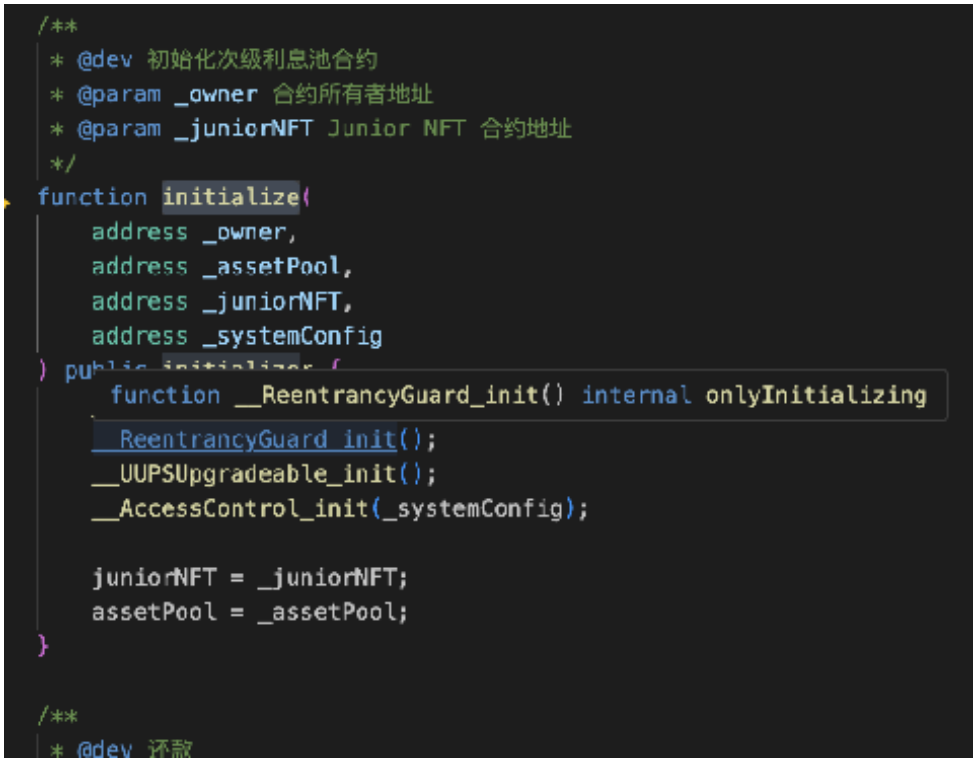


Figure 24: JuniorInterestPool initialize function without zero checks

Funding initialization function lacks zero address checks:

```
function initialize(
    address _owner,
    address _assetPool,
    address _assetAddress,
    uint256 _targetAmount,
    uint256 _minAmount,
    uint256 _startTime,
    uint256 _endTime,
    uint256 _minJuniorRatio
) public initializer {
    __Ownable_init(_owner);
    __ReentrancyGuard_init();
    __UUPSUpgradeable_init();

    assetPool = _assetPool;

    // 初始化AccessControl
    address systemConfig = IAssetPool(_assetPool).systemConfig();
    __AccessControl_init(systemConfig);

    _fundingInfo = FundingInfo({
        assetAddress: _assetAddress,
        targetAmount: _targetAmount,
        seniorAmount: 0,
        juniorAmount: 0,
        minAmount: _minAmount,
        startTime: _startTime,
        endTime: _endTime,
        minJuniorRatio: _minJuniorRatio
    });
}
```

Figure 25: Funding initialize function without zero checks

Remediation Recommendations: Add zero address checks:

```
require(address != address(0), "Invalid address");
```

```
function initialize(address _owner, address _assetPool, address _juniorNFT
    require(_assetPool != address(0), "Invalid asset pool address");
    require(_juniorNFT != address(0), "Invalid junior NFT address");

    __Ownable_init(_owner);
    __ReentrancyGuard_init();
    __UUPSUpgradeable_init();
    __AccessControl_init(_systemConfig);
    juniorNFT = _juniorNFT;
    assetPool = _assetPool;
}
```

Figure 26: Funding initialize function without zero checks

3 Summary and Recommendations

3.1 Key Findings Summary

The main issues identified in this audit include:

1. **Improper Access Control**: Factory contract's initialize function lacks protection
2. **ERC20 Compatibility Issues**: Multiple contracts don't check transfer return values
3. **DoS Attack Risk**: Unrestricted loop length may cause resource exhaustion
4. **Time Manipulation Risk**: Dependence on block.timestamp may be manipulated by miners
5. **Insufficient Input Validation**: Missing zero address and boundary checks

3.2 Overall Security Recommendations

1. **Immediately fix critical and high-risk vulnerabilities**: Prioritize access control and ERC20 transfer-related issues
2. **Strengthen input validation**: All external inputs should undergo proper validation and boundary checks
3. **Use secure ERC20 libraries**: Recommend using OpenZeppelin's SafeERC20 library
4. **Implement access control best practices**: Use mature access control patterns like OpenZeppelin's Ownable
5. **Add emergency pause mechanisms**: Consider adding emergency pause functionality for sudden security incidents
6. **Comprehensive testing**: Conduct thorough unit and integration testing after fixes
7. **Continuous monitoring**: Implement continuous security monitoring after deployment

3.3 Fix Verification

After fixing all identified issues, the following verification is recommended:

- Conduct re-audit
- Perform penetration testing
- Deploy to testnet for thorough testing
- Consider establishing a bug bounty program

4 Disclaimer

This audit report is based on analysis of the provided smart contract code and has limited scope. This report cannot guarantee discovery of all possible security vulnerabilities, nor can it guarantee code security in all situations. The project team is advised to conduct multiple rounds of security audits before deployment and establish continuous security monitoring mechanisms.