hexens × FUEL

SEPT.24

SECURITY REVIEW REPORT FOR FUEL

CONTENTS

- About Hexens
- Executive summary
  - Overview
  - Scope
- Auditing details
- Severity structure
  - Severity characteristics
  - Issue symbolic codes
- Findings summary
- Weaknesses
  - Use forceApprove() instead of safeIncreaseAllowance() in PreDeposits
  - depositWithPermit doesn't work as expected with WETH token
  - Check if balanceAdjusted is more than 0 in PreDeposits._migrate()
  - The function decimals() is not part of ERC-20
  - Inconsistent usage of msg.sender in PreDeposits.sol
  - PreDeposits.sol may emit a Migration event that isn't distinguishing WETH migrations
  - Inconsistency with deprecated depositParam handling
  - WETH deposits from contracts without receive() may not be able to use PreDeposits::migrate()
  - Missing recovery system for accidentally sent ETH

ABOUT HEXENS

Hexens is a cybersecurity company that strives to elevate the standards of security in Web 3.0, create a safer environment for users, and ensure mass Web 3.0 adoption.

Hexens has multiple top-notch auditing teams specialized in different fields of information security, showing extreme performance in the most challenging and technically complex tasks, including but not limited to: Infrastructure Audits, Zero Knowledge Proofs / Novel Cryptography, DeFi and NFTs. Hexens not only uses widely known methodologies and flows, but focuses on discovering and introducing new ones on a day-to-day basis.

In 2022, our team announced the closure of a $4.2 million seed round led by IOSG Ventures, the leading Web 3.0 venture capital. Other investors include Delta Blockchain Fund, Chapter One, Hash Capital, ImToken Ventures, Tenzor Capital, and angels from Polygon and other blockchain projects.

Since Hexens was founded in 2021, it has had an impressive track record and recognition in the industry: Mudit Gupta - CISO of Polygon Technology - the biggest EVM Ecosystem, joined the company advisory board after completing just a single cooperation iteration. Polygon Technology, 1inch, Lido, Hats Finance, Quickswap, Layerswap, 4K, RociFi, as well as dozens of DeFi protocols and bridges, have already become our customers and taken proactive measures towards protecting their assets.

EXECUTIVE SUMMARY

OVERVIEW

This audit covered an update to the pre-deposit contracts of Fuel Network. The update added a points migration function.

Our security assessment was a full review of the updates made, spanning a total of 1 week.

During our audit, we have only identified minor severity vulnerabilities and code optimisations.

Finally, all of our reported issues were fixed or acknowledged by the development team and consequently validated by us.

We can confidently say that the overall security and code quality have increased after completion of our audit.

SCOPE

The analyzed resources are located on:

https://github.com/FuelLabs/predeposit-contracts/tree/1dc275e58f1b8dade136b02798ffae6b5eb2c9ba

The issues described in this report were fixed. Corresponding commits are mentioned in the description.

AUDITING DETAILS

STARTED

23.09.2024

DELIVERED

30.09.2024

Review Led by

NOUREDDINE BENOMARI

Security Researcher | Hexens

HEXENS METHODOLOGY

Hexens methodology involves 2 teams, including multiple auditors of different seniority, with at least 5 security engineers. This unique cross-checking mechanism helps us provide the best quality in the market.

SEVERITY STRUCTURE

The vulnerability severity is calculated based on two components

- Impact of the vulnerability
- Probability of the vulnerability

Impact	Probability			
	rare	unlikely	likely	very likely
Low/Info	Low/Info	Low/Info	Medium	Medium
Medium	Low/Info	Medium	Medium	High
High	Medium	Medium	High	Critical
Critical	Medium	High	Critical	Critical

SEVERITY CHARACTERISTICS

Smart contract vulnerabilities can range in severity and impact, and it's important to understand their level of severity in order to prioritize their resolution. Here are the different types of severity levels of smart contract vulnerabilities:

Critical

Vulnerabilities with this level of severity can result in significant financial losses or reputational damage. They often allow an attacker to gain complete control of a contract, directly steal or freeze funds from the contract or users, or permanently block the functionality of a protocol. Examples include infinite mints and governance manipulation.

High

Vulnerabilities with this level of severity can result in some financial losses or reputational damage. They often allow an attacker to directly steal yield from the contract or users, or temporarily freeze funds. Examples include inadequate access control integer overflow/underflow, or logic bugs.

Medium

Vulnerabilities with this level of severity can result in some damage to the protocol or users, without profit for the attacker. They often allow an attacker to exploit a contract to cause harm, but the impact may be limited, such as temporarily blocking the functionality of the protocol. Examples include uninitialized storage pointers and failure to check external calls.

Low

Vulnerabilities with this level of severity may not result in financial losses or significant harm. They may, however, impact the usability or reliability of a contract. Examples include slippage and front-running, or minor logic bugs.

Informational

Vulnerabilities with this level of severity are regarding gas optimizations and code style. They often involve issues with documentation, incorrect usage of EIP standards, best practices for saving gas, or the overall design of a contract. Examples include not conforming to ERC20, or disagreement between documentation and code.

ISSUE SYMBOLIC CODES

Every issue being identified and validated has its unique symbolic code assigned to the issue at the security research stage. Cause of the vulnerability reporting flow design, some of the rejected issues could be missing.

FINDINGS SUMMARY

Severity	Number of Findings
Critical	0
High	0
Medium	0
Low	3
Informational	6

Total: 9

WEAKNESSES

This section contains the list of discovered weaknesses.

FUEL7-13

USE FORCEAPPROVE() INSTEAD OF SAFEINCREASEALLOWANCE() IN PREDEPOSITS

SEVERITY: Low

PATH:

contracts/PreDeposits/PreDeposits.sol#L173-L176

REMEDIATION:

Replace the safeIncreaseAllowance call with forceApprove.

STATUS: Fixed

DESCRIPTION:

The allowance increase is not necessary in this case because the old allowance will never be used by **FuelERC20Gateway**.

The stricter **forceApprove()** has fewer safety assumptions (no risk of leftover approvals usage) and uses less gas.

```
IERC20(token).safeIncreaseAllowance(
    fuelERC20Gateway,
    balanceAdjusted
);
```

DEPOSITWITHPERMIT DOESN'T WORK AS EXPECTED WITH WETH TOKEN

SEVERITY: Low

PATH:

contracts/PreDeposits/PreDeposits.sol#L61-L94

REMEDIATION:

Add a check in the depositWithPermit function to detect if the token being deposited is WETH. If WETH is detected, skip the permit function call entirely since it is not supported by WETH. Instead, require manual approval for WETH transfers.

STATUS: Acknowledged

DESCRIPTION:

The **depositWithPermit** function is designed to approve a spender by verifying a valid signature using the **permit** function. Most ERC20 tokens implement this functionality, allowing users to approve transfers without needing to send a separate transaction.

However, **WETH (Wrapped Ether)** does not implement the **permit** function. Despite this, when **permit** is called on the WETH token, the transaction executes without any errors because WETH's **fallback** function is triggered instead. This silent execution gives the appearance that the **permit** function succeeded, but in reality, the approval does not occur. As a result, the subsequent transfer in the **depositWithPermit** function will fail, causing the transaction to revert when the approval limit is insufficient.

```
    function depositWithPermit(
        address token,
        uint240 amount,
        uint16 depositParam, // deprecated param
        uint256 deadline,
        uint8 v,
        bytes32 r,
        bytes32 s
    ) external whenNotPaused {
        DepositRecord memory _tokenDeposit = deposits[_msgSender()][token];

        _tokenDeposit.balance += amount;
        _tokenDeposit.depositParam = depositParam;
        deposits[_msgSender()][token] = _tokenDeposit;

        // (hexens fuel4-2 fix: Permit can be ddos'ed)
        // Adding a try-catch clause allows to skip ddos transactions here
        // We are not interested in either catching the error or
implementing
        // a success flow, we just continue and let safeTransferFrom revert
        try
            ERC20Permit(token).permit(
                _msgSender(),
                address(this),
                amount,
                deadline,
                v,
                r,
                s
            )
        {} catch {}

        IERC20(token).safeTransferFrom(_msgSender(), address(this), amount);
        emit Deposit(_msgSender(), token, amount, depositParam);
    }
```

CHECK IF BALANCEADJUSTED IS MORE THAN 0 IN PREDEPOSITS._MIGRATE()

SEVERITY: Low

PATH:

contracts/PreDeposits/PreDeposits.sol#L166-L166

REMEDIATION:

Add a check for balanceAdjusted > 0.

STATUS: Fixed

DESCRIPTION:

The **PreDeposits._migrate()** will create an empty cross-chain transfer in case **_adjustDepositDecimals** returns zero for ETH or WETH, which is possible if the amount to send is less than 1e9.

Zero transfer in the case of other ERC20 tokens will result in a revert in **FuelERC20GatewayV4**, but **FuelMessagePortal** doesn't incorporate such checks.

```
uint256 balanceAdjusted = _adjustDepositDecimals(balance, token);
```

THE FUNCTION DECIMALS() IS NOT PART OF ERC-20

SEVERITY: Informational

PATH:

PreDeposits.sol#L255

REMEDIATION:

Consider handling the tokens without decimals() function.

STATUS: Acknowledged

DESCRIPTION:

The **decimals()** function is not a part of the ERC-20 standard.
It is just an optional extension - see here: https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/token/ERC20/extensions/IERC20Metadata.sol

Thus some ERC20 tokens may not support this interface, which potentially makes it unsafe to blindly cast all tokens to this interface.

Note that the **PreDeposits.deposit()** function will not revert for ERC20 tokens without the **decimals()** function, but when trying to migrate these tokens, it may revert.

```
function _adjustDepositDecimals(
  uint256 amount,
  address token
 ) internal returns (uint256) {
  uint256 tokenDecimals;
  if (token == address(0)) {
   tokenDecimals = 18;
  } else {
   tokenDecimals = IERC20Metadata(token).decimals();
  }
```

INCONSISTENT USAGE OF MSG.SENDER IN PREDEPOSITS.SOL

SEVERITY: Informational

PATH:

PreDeposits.sol#L265-L267

REMEDIATION:

Consider adjusting PreDeposits.sol by using _msgSender() instead of msg.sender for better consistency in the codebase.

STATUS: Fixed

DESCRIPTION:

In the **PreDeposits** contract the usage of msg.sender is inconsistent.

```
    IERC20(token).safeTransfer(msg.sender, dust);
  } else {
    (bool sent, ) = address(msg.sender).call{ value: dust }(
```

PREDEPOSITS.SOL MAY EMIT A MIGRATION EVENT THAT ISN'T DISTINGUISHING WETH MIGRATIONS

SEVERITY: Informational

PATH:

PreDeposits.sol#L161-L184

REMEDIATION:

Consider adjusting PreDeposits::_migrate() to emit the WETH token address when migrating WETH tokens.

STATUS: Fixed

DESCRIPTION:

When a user migrates their WETH from PreDeposits.sol, a **Migration** event may be emitted suggesting that ETH tokens **address(0)** were migrated, since **token** is set to **address(0)** in this case inside **PreDeposits::_migrate()** on line 163 in PreDeposits.sol.

```
if (token == address(weth)) {
  weth.withdraw(balance);
  token = address(0);
}
...
emit Migration(_msgSender(), token, balanceAdjusted);
```

INCONSISTENCY WITH DEPRECATED DEPOSITPARAM HANDLING

SEVERITY: Informational

PATH:

PreDeposits::depositWithPermit()#L73

PreDeposits::_depositWithoutCommitment()#L210

REMEDIATION:

Inside PreDeposits::depositWithPermit() set depositParam to 0.

STATUS: Fixed

DESCRIPTION:

PreDeposits::depositWithPermit() still sets the **depositParam** whereas **PreDeposits::deposit()** is setting **depositParam** to 0.

```
_tokenDeposit.depositParam = depositParam;
```

```
_tokenDeposit.depositParam = 0;
```

WETH DEPOSITS FROM CONTRACTS WITHOUT RECEIVE() MAY NOT BE ABLE TO USE PREDEPOSITS::MIGRATE()

SEVERITY: Informational

PATH:

PreDeposits.sol#L267-L272

REMEDIATION:

Consider mentioning the following for example in Fuel docs so that other developers are aware of this:

To avoid funds getting stuck, the contract which uses PreDeposits.sol with WETH should implement either:

1. receive function
2. a function to withdraw from PreDeposits
3. deposit without dust amount

STATUS: Fixed

DESCRIPTION:

If a contract without **receive()** deposits WETH with some possible dust amount, it cannot use **PreDeposits::migrate()** since it will revert.

```
(bool sent, ) = address(msg.sender).call{ value: dust }(
    ""
);
if (!sent) {
    revert TransferReverted();
}
```

MISSING RECOVERY SYSTEM FOR ACCIDENTALLY SENT ETH

SEVERITY: Informational

PATH:

contracts/PreDeposits/PreDeposits.sol#L44

REMEDIATION:

It is recommended to implement a recovery mechanism that allows for the retrieval of these funds.

STATUS: Acknowledged

DESCRIPTION:

In the **PreDeposits** contract, the **receive() external payable {}** function is used to allow the contract to accept ETH. However, if a user accidentally sends ETH to this contract, there is currently no mechanism to recover the funds, causing the ETH to become stuck in the contract.

```
receive() external payable {}
```

hexens × FUEL