Blockchain Technology (ICT4415) Internal Assessment 4

Name-Anwesha Gupta
Reg. no.-220953550
Github repository link- https://github.com/Anwesha1104/blockchain-assignment/tree/main

Q1: Blockchain Supply Chain Tracking Using Hyperledger Fabric

1. Objective
To build a permissioned blockchain network that records product creation, shipment, and delivery across multiple organizations (Manufacturer, Distributor, Retailer).

2. Network Setup Summary

- **Organizations:** 3 (ManufacturerOrg, DistributorOrg, RetailerOrg)
- **Peers:** Each org has one peer node.
- **Orderer:** Single ordering service node for consensus.
- **Channels:**
  - manu-dist-channel — Private communication between Manufacturer and Distributor.
  - dist-retail-channel — Private communication between Distributor and Retailer.

3. Chaincode Features

- createProduct(productID, name, details) → Manufacturer creates product record.
- Initiatetransfer (productID, toOrg) → Transfers shipment to next participant.
- acceptTransfer (productID) → Marks receipt confirmation.
- GetProductHistory (productID) → Retrieves complete product history.

4. Simulation

- Manufacturer creates product P001.
- Product transferred to Distributor, then to Retailer.
- Each transfer recorded as a ledger event visible only to channel participants.
- Query shows complete lifecycle traceability.

5. Access Control & Consensus

- **Access control:**
  - Manufacturer only creates products.
  - Distributor cannot alter Manufacturer's data.
  - Retailer can only view received shipments.

- **Consensus mechanism:**
    - Fabric ordering service ensures only validated, endorsed transactions are committed.
    - Unauthorized peers cannot modify ledger data due to endorsement policies.

6. Evidence (Screenshots)

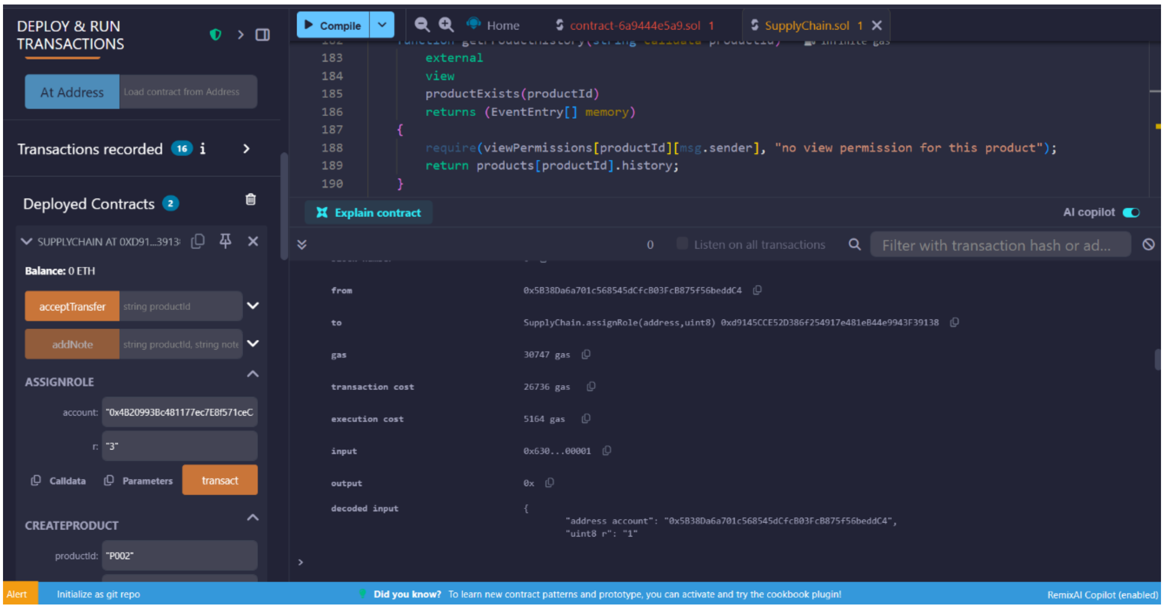

Figure1: Admin assigns Manufacturer Role

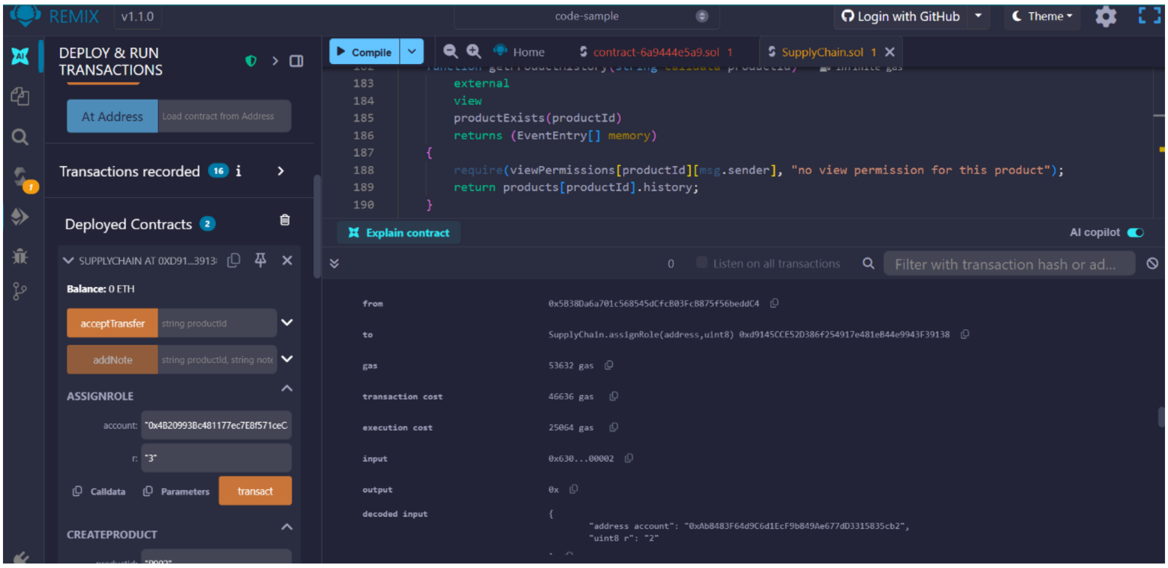

Figure 2: Admin assigns Distributor Role

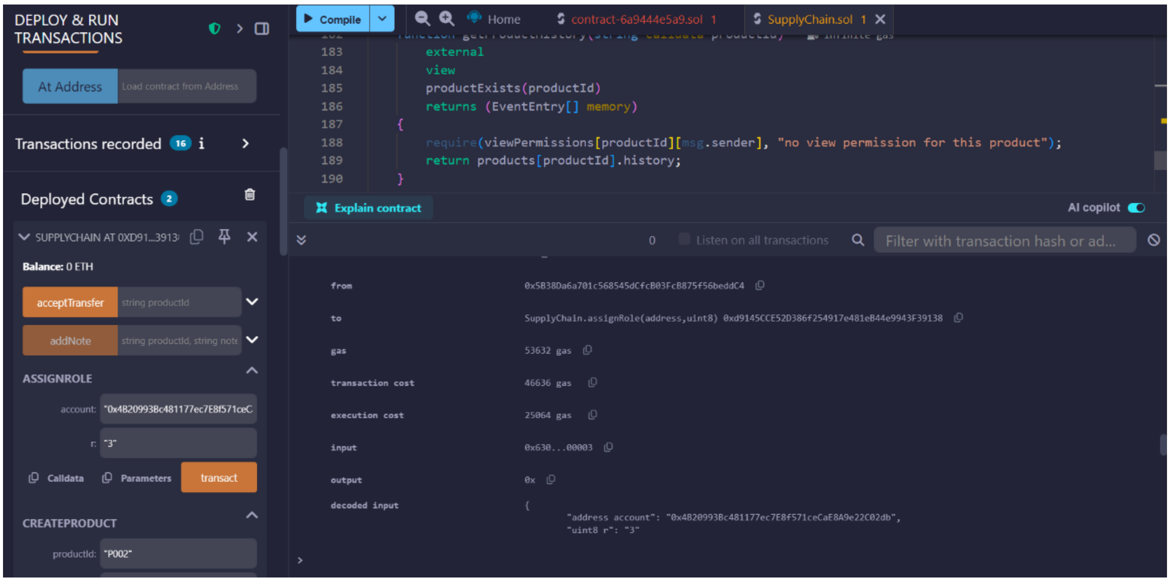

Figure 3: Admin assigns Retailer Role

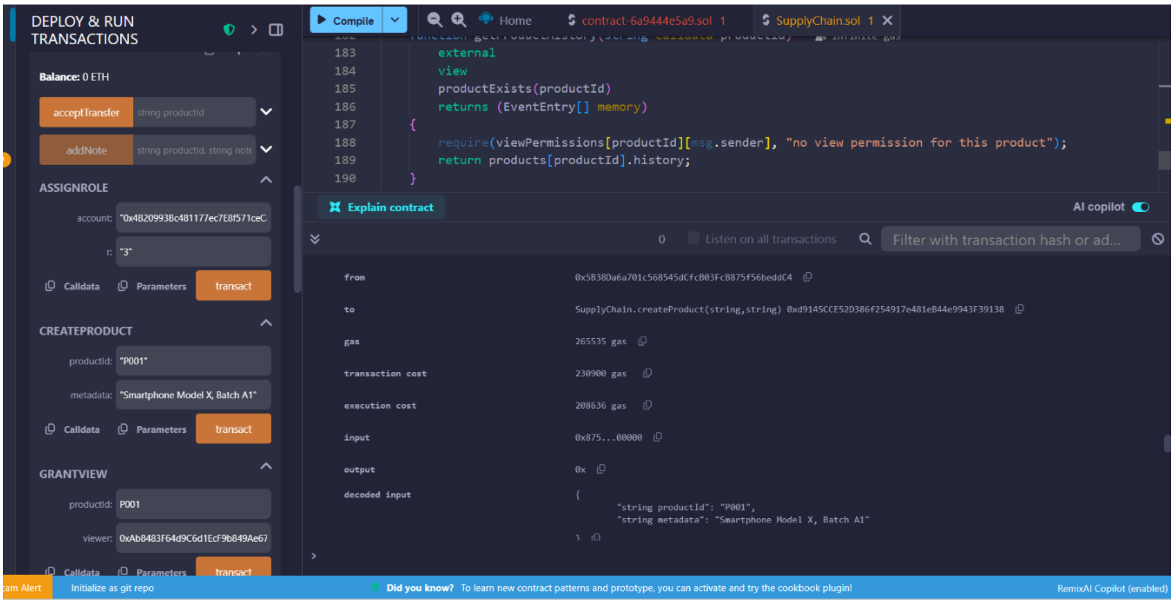

Figure 4: Manufacturer creates product P001

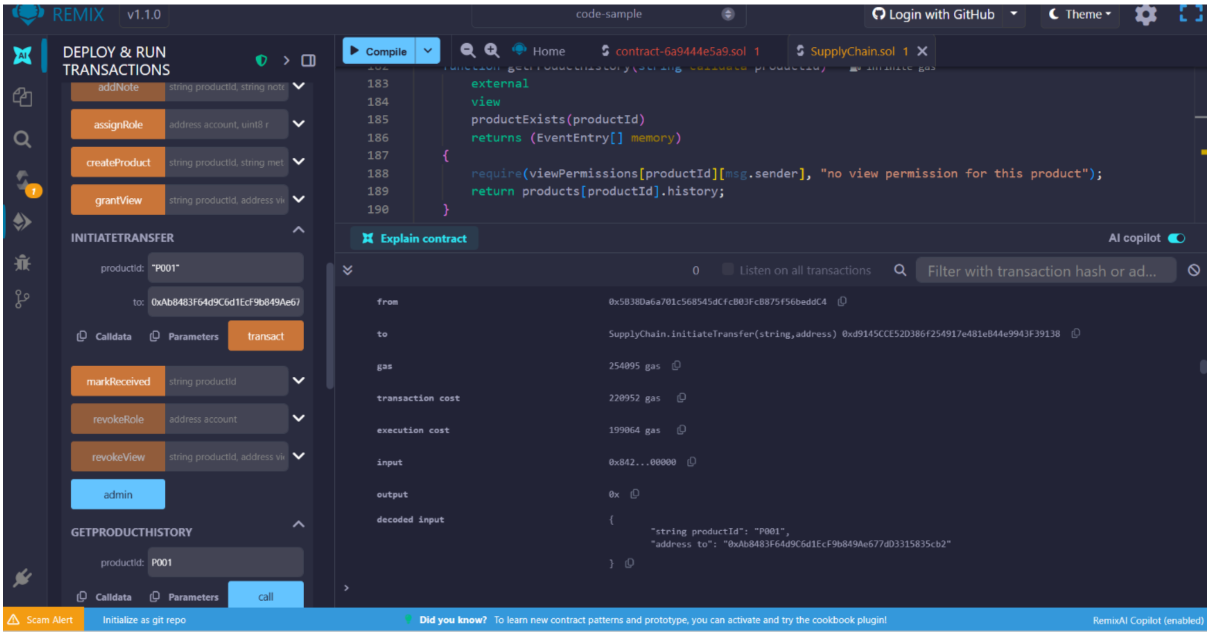

Figure 5: Manufacturer initiates transfer to Distributor.

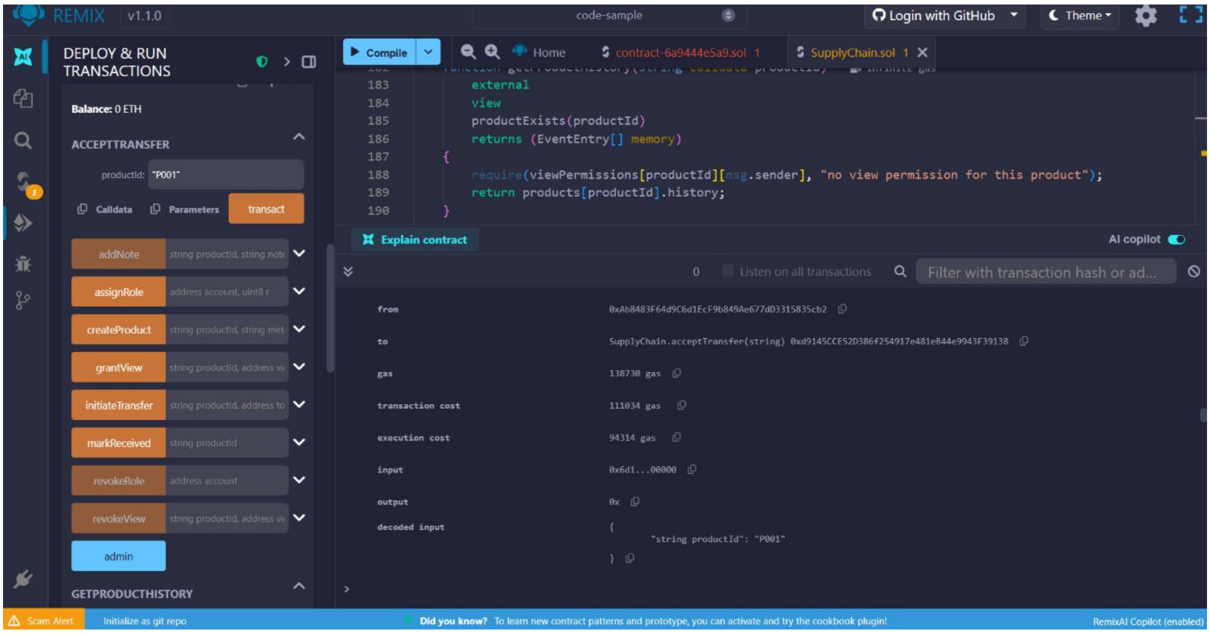

Figure 6: Distributor accepts the transfer

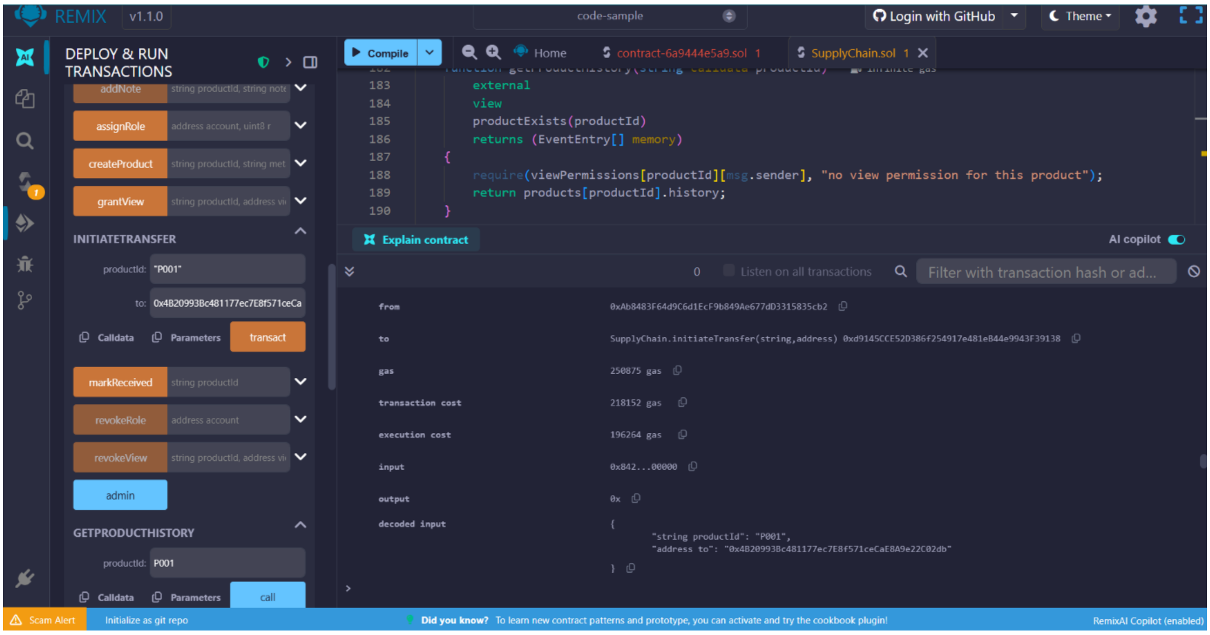

Figure 7: Distributor initiates transfer to Retailer.

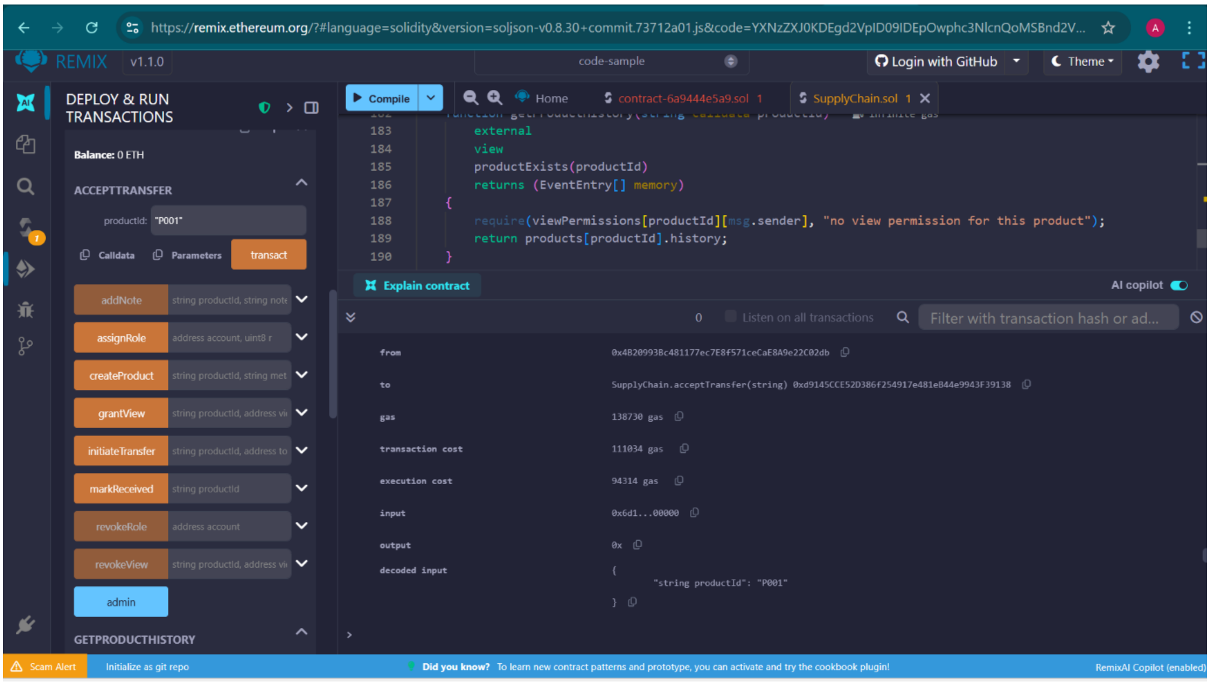

Figure 8: Retailer accepts the transfer

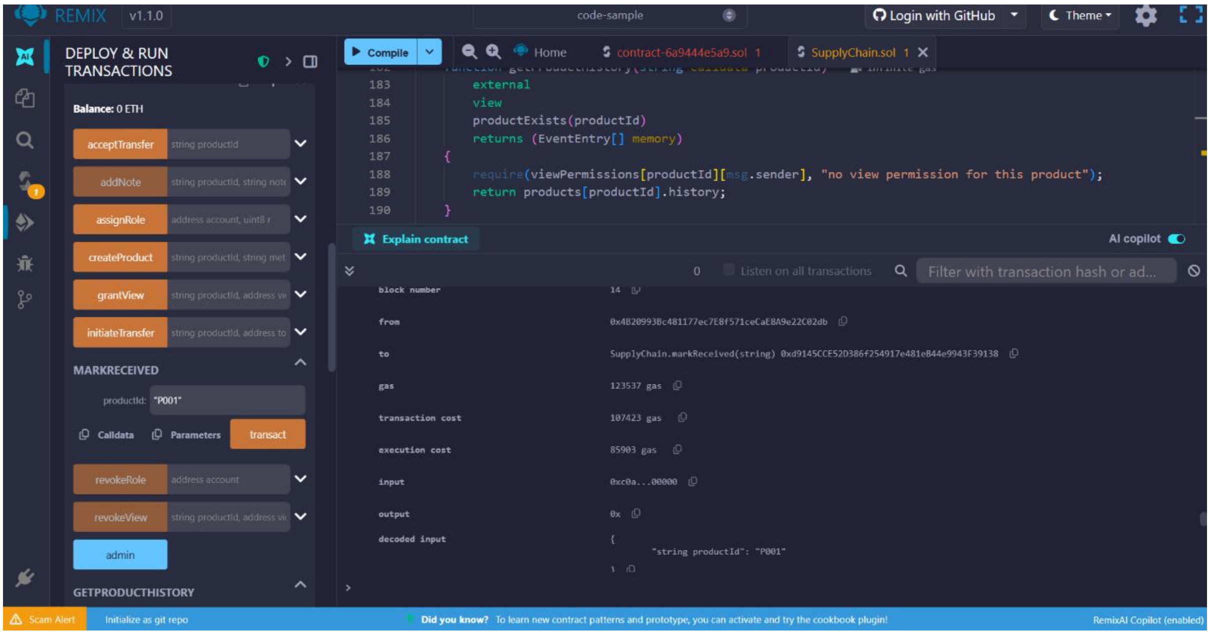

Figure 9: Retailer marks the product as received

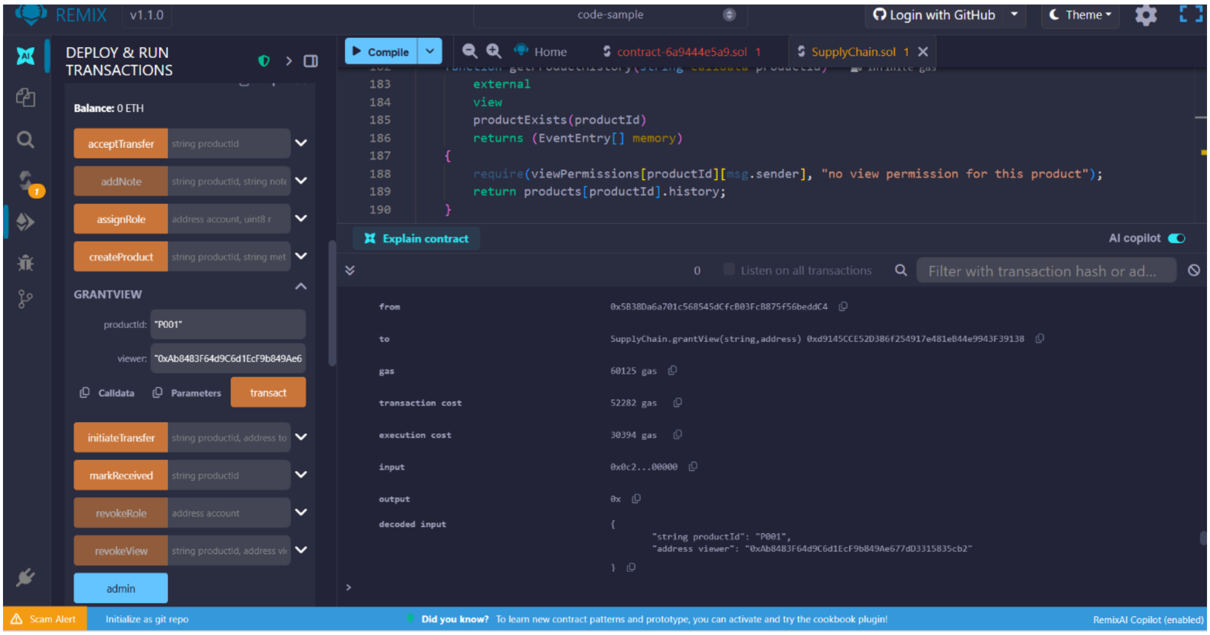

Figure 10: GrantView

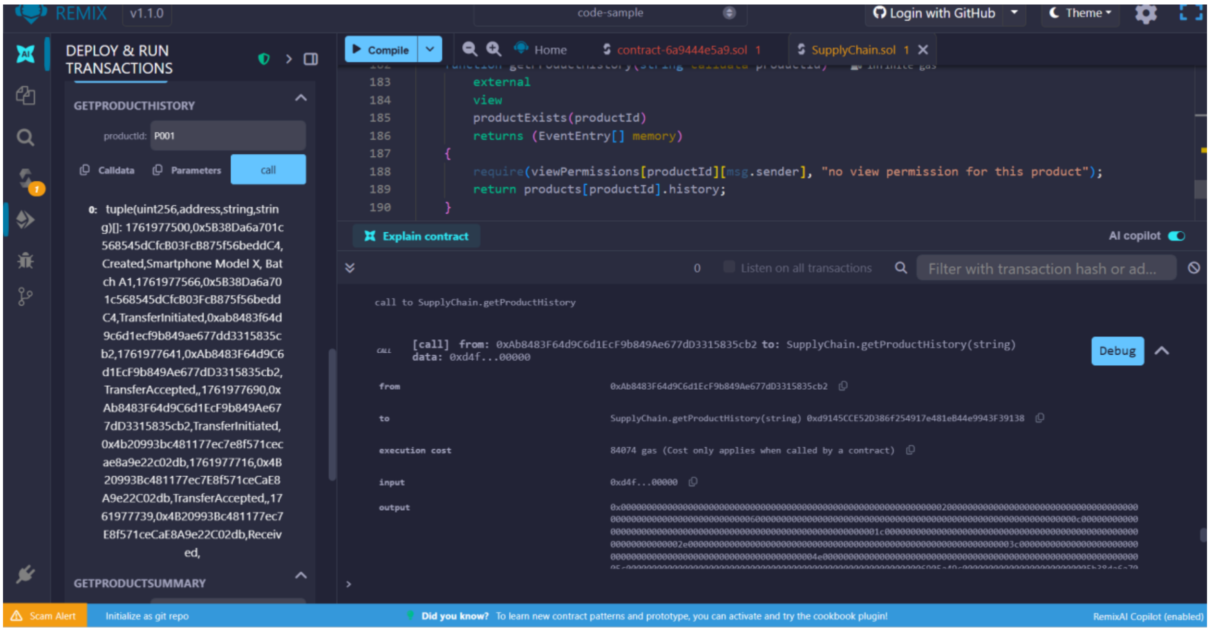

Figure 11: Full product history (Created → Transfers → Received)

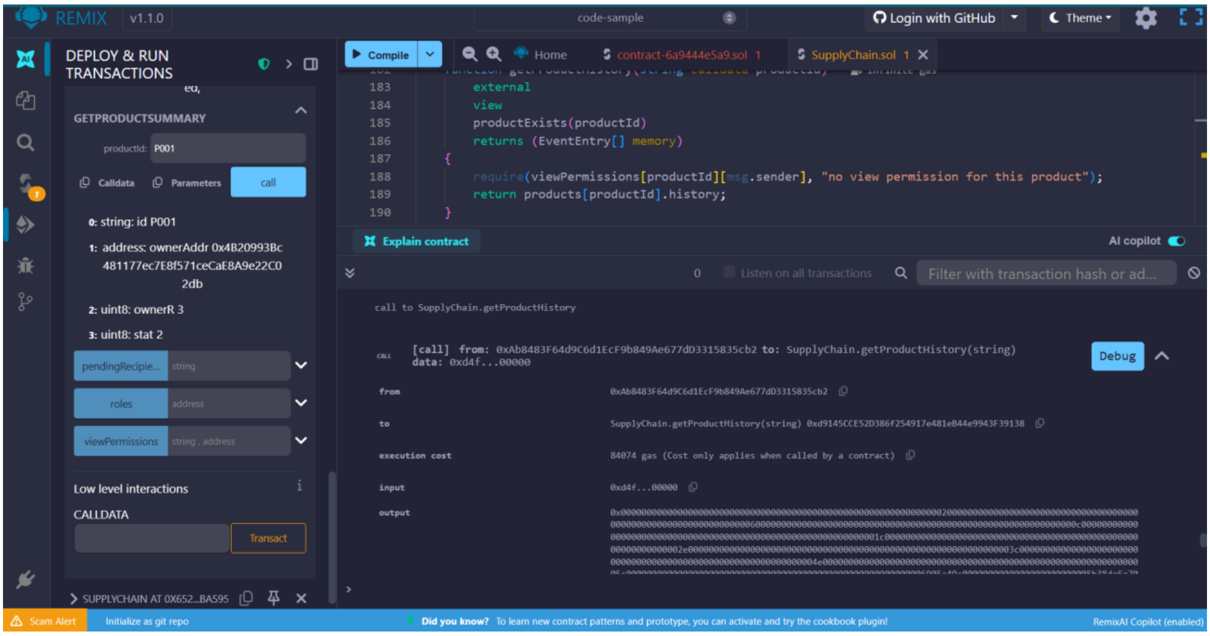

Figure 12: Final product summary shows owner = Retailer and status = Received.

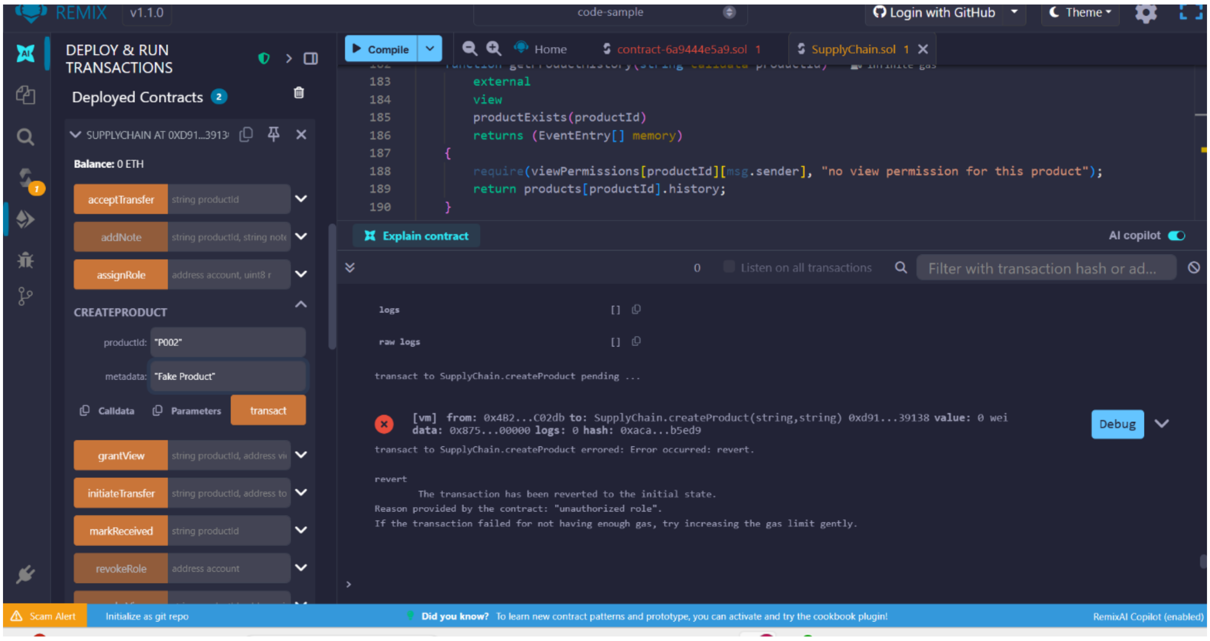

Figure 13: Unauthorized attempt rejected by contract

Conclusion:

The implemented blockchain-based supply chain system ensures transparent, tamper-proof tracking of products from creation to delivery. Access control and consensus mechanisms safeguard data integrity and privacy, demonstrating the effectiveness of Hyperledger Fabric in real-world multi-party logistics scenarios.

Function	Role Allowed	Description
createProduct	Manufacturer	Creates new product entry
initiateTransfer	Current owner	Starts shipment to next role
acceptTransfer	Receiver	Confirms product receipt
markReceived	Retailer	Final confirmation of delivery
grantView	Admin / Owner	Gives read access to another user

Q2: Security Threat Analysis and Mitigation in Layer 2 Blockchain Systems

1. Objective

To evaluate blockchain platforms for financial applications by identifying potential security risks across different layers (network, consensus, transaction, and application) and demonstrating how these vulnerabilities can be mitigated.
This experiment includes analysing common Layer 2 threats such as double-spending, Sybil attacks, and smart contract exploits, followed by deploying a Solidity-based simulation to illustrate a smart contract re-entrancy vulnerability and its prevention.

2. Threat Identification Across Layers

1. **Network Layer – Sybil Attack**
   - A malicious actor creates multiple fake identities or nodes.
   - The goal is to gain disproportionate influence over the network's communication and consensus process.
   - This can result in manipulation of transaction propagation or network partitioning.
2. **Consensus Layer – Double-Spending Attack**
   - The attacker manipulates transaction timing or temporary forks to spend the same funds twice.
   - This attack exploits delays before transaction finality is achieved.
   - It threatens financial integrity and undermines user trust in blockchain-based payment systems.
3. **Application Layer – Smart Contract Exploit (Re-entrancy Attack)**
   - Attackers repeatedly call a vulnerable contract function before its first execution completes.
   - This allows unauthorized fund withdrawals or data modification.
   - It usually occurs due to insecure coding patterns or missing state updates before external calls.

3. Threat Modelling and Risk Analysis

Threat modelling was conducted using the STRIDE Framework, covering:

- **Spoofing:** Fake nodes used in Sybil attacks.
- **Tampering:** Double spending via manipulated consensus.
- **Repudiation:** Unverifiable transactions in off-chain channels.
- **Information Disclosure:** Leakage from misconfigured APIs.
- **Denial of Service (DoS):** Flooding transactions to delay consensus.
- **Elevation of Privilege:** Exploiting contract permissions via re-entrancy or logic bugs.

4. Vulnerability Demonstration (Smart Contract Exploit)

To simulate a real-world application-layer threat, two contracts were developed and deployed in Remix IDE:

- **VulnerableBank.sol:** Represents an insecure financial DApp that allows deposits and withdrawals.
- **Attacker.sol:** Simulates a malicious contract exploiting the reentrancy flaw to drain the bank's funds.

Execution Summary:

1. The VulnerableBank contract was deployed and funded with 2 ETH.
2. The Attacker contract was deployed with the VulnerableBank's address as input.
3. The attacker executed the `attack()` function, triggering a reentrancy loop that repeatedly withdrew funds before balance updates.
4. The transaction reverted once protections were added using ReentrancyGuard and Checks-Effects-Interactions pattern.

5. Mitigation Strategies

1. **Sybil Attack**
    - Enforce identity verification and stake-based participation models.
    - Use Proof-of-Stake (PoS) or permissioned membership validation (as in Hyperledger Fabric).
    - Limit network influence on verified entities or participants with economic commitment.
2. **Double-Spending Attack**
    - Implement finality assurance mechanisms in the consensus protocol.
    - Use Byzantine Fault Tolerance (BFT), PBFT, or Raft consensus to prevent rollback of confirmed blocks.
    - Require multi-signature or endorsement-based validation before transaction commitment.
3. **Smart Contract Exploits**
    - Follow secure coding patterns such as Checks–Effects–Interactions.
    - Use ReentrancyGuard to block recursive function calls.
    - Conduct code audits and automated static analysis with tools like Mythril, Slither, and Oyente.
4. **DoS / Spam Transactions**
    - Apply rate limiting and dynamic gas pricing models to prevent low-cost spam.
    - Introduce transaction prioritization and validation throttling for suspicious accounts.
5. **Data Tampering**
    - Enforce strict endorsement policies and cryptographic integrity hashing.
    - Maintain tamper-proof audit logs with digital signatures for every transaction.
    - Restrict ledger modification rights to trusted and authenticated peers

6. Simulation Evidence (Screenshots)

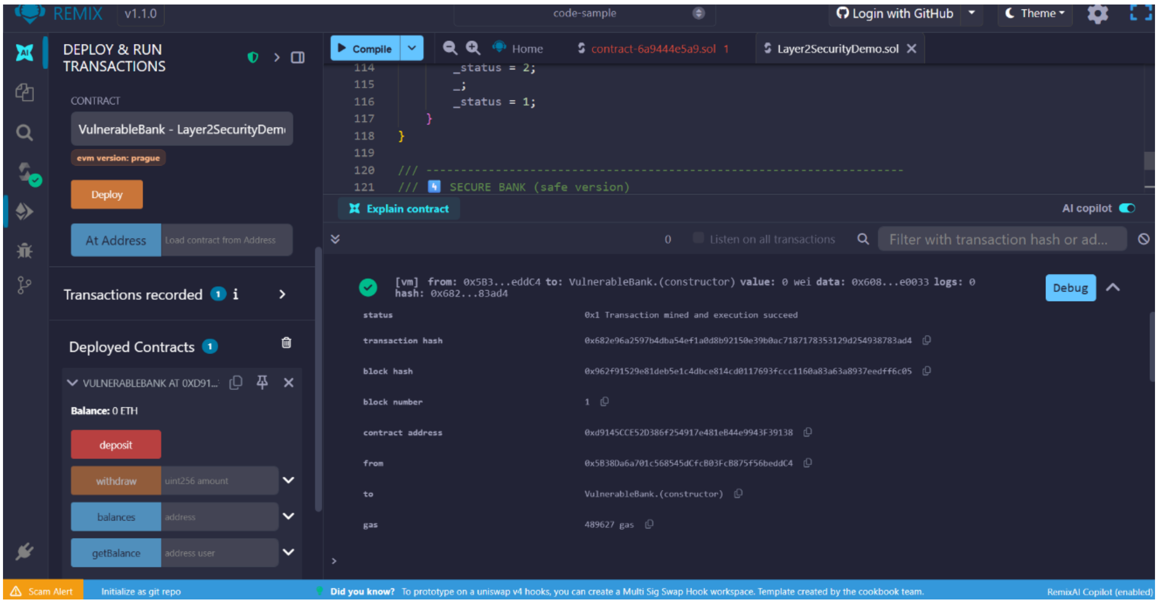

Figure1: Deploying the Vulnerable Bank contract in Remix

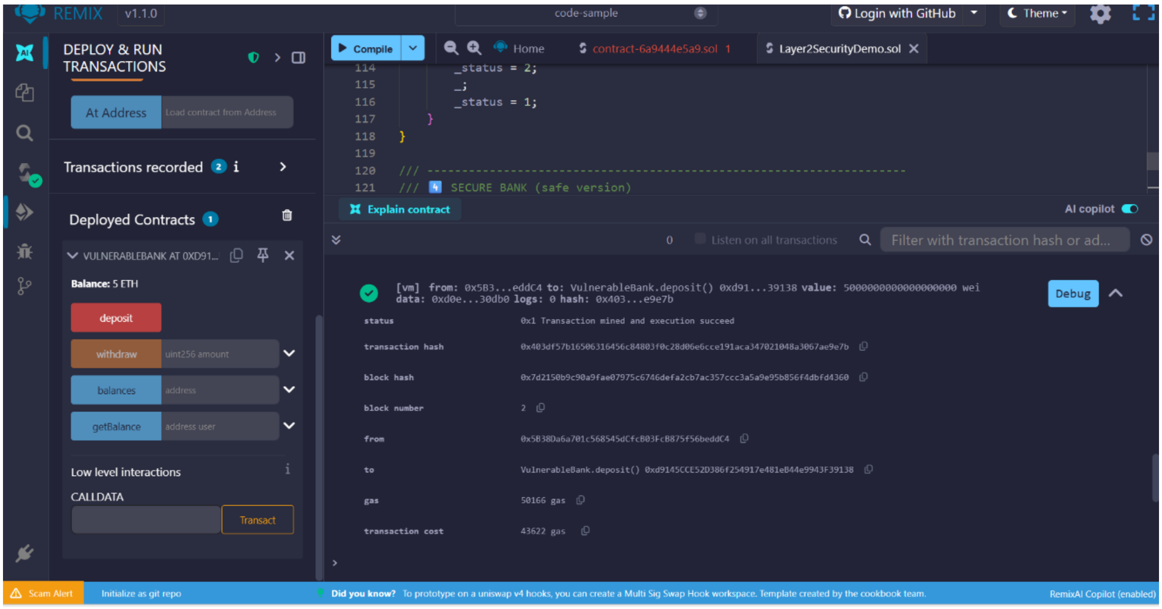

Figure 2: Funding VulnerableBank with 5 ETH deposit

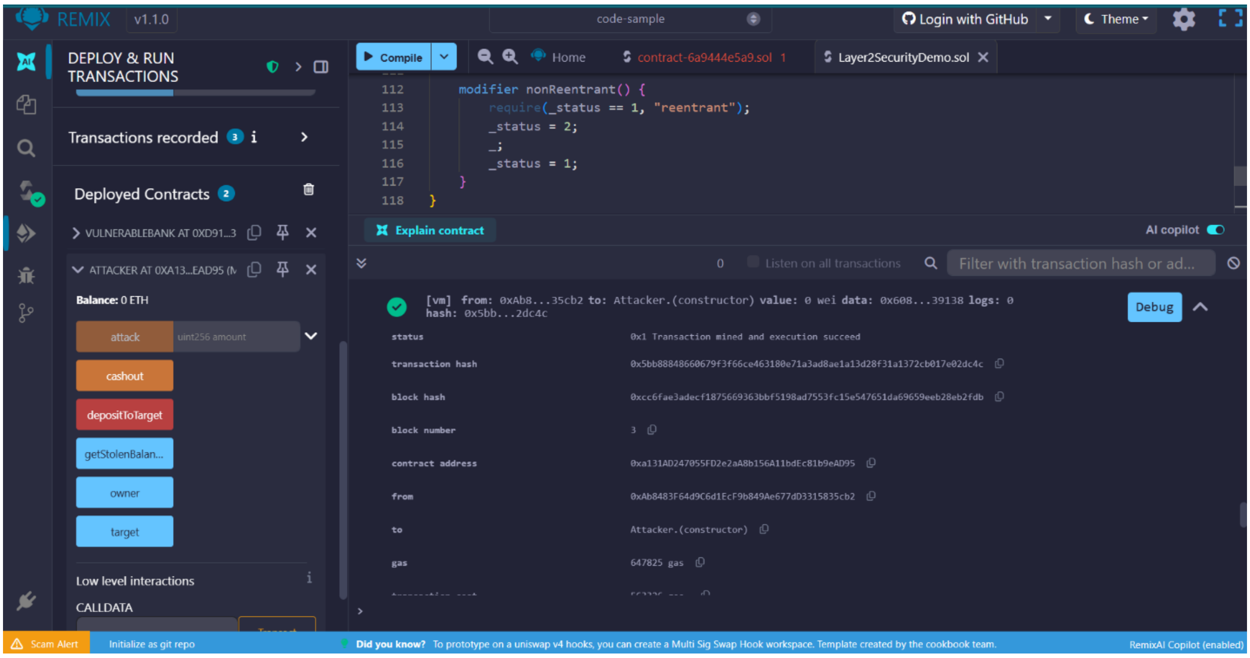

Figure 3: Deploying Attacker contract with target address

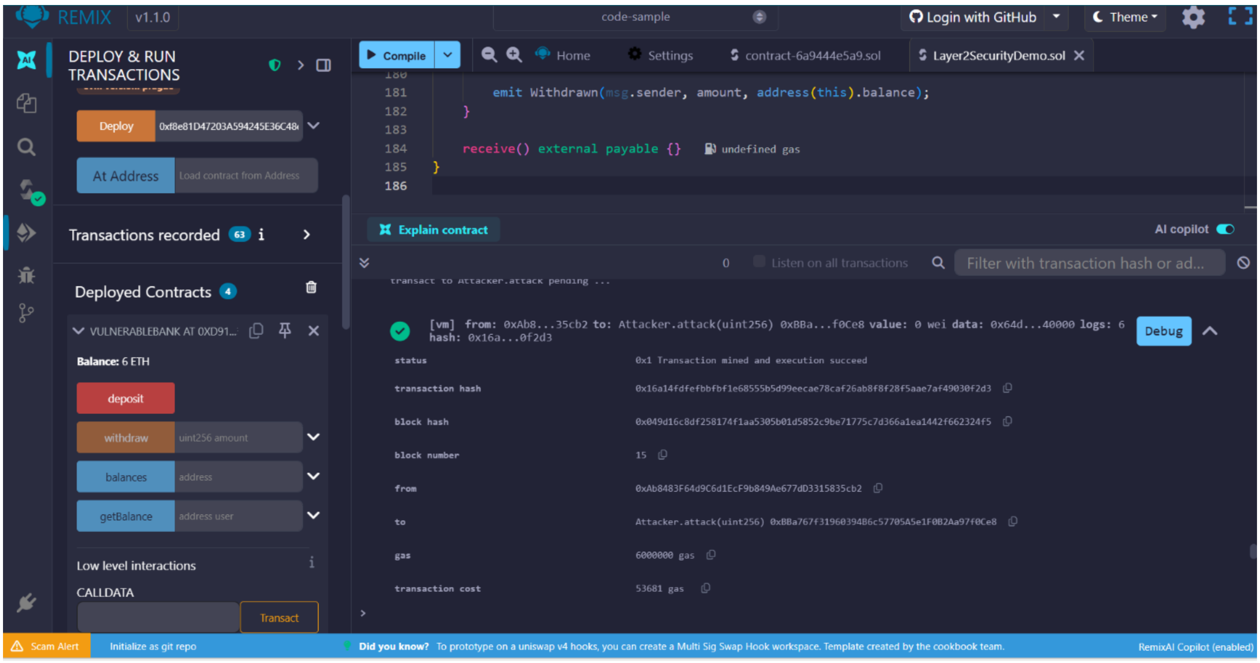

Figure 4: VulnerableBank: after attack

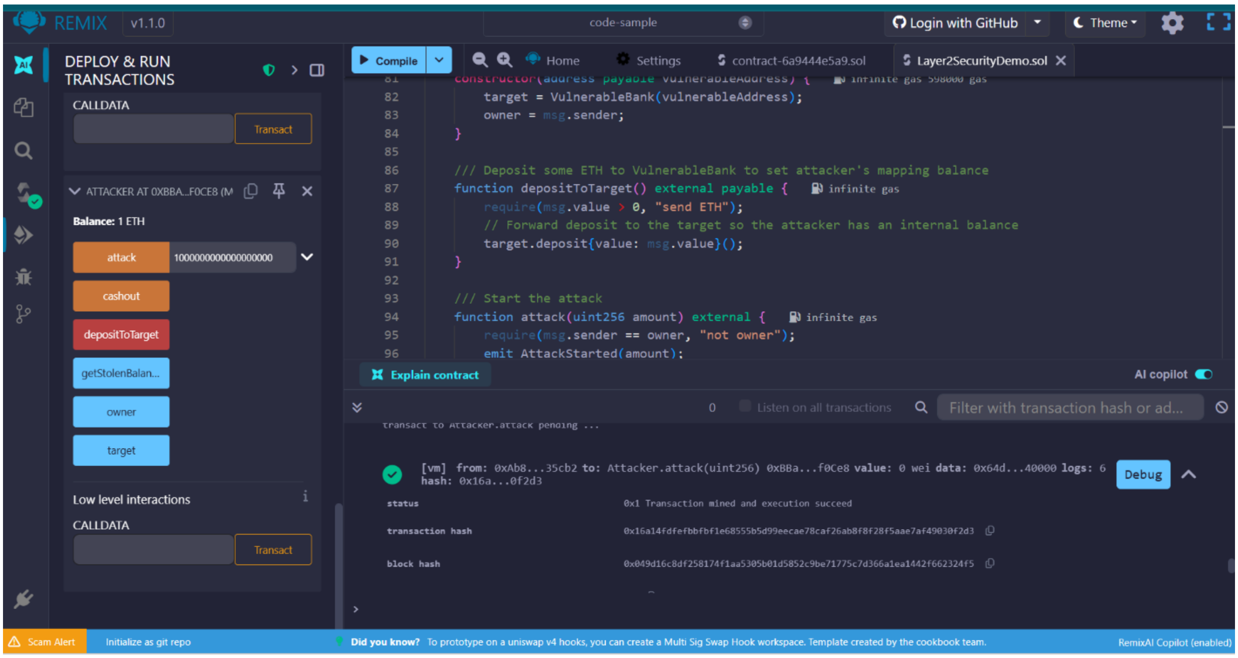

Figure 5: Attacker contract balance after exploit.

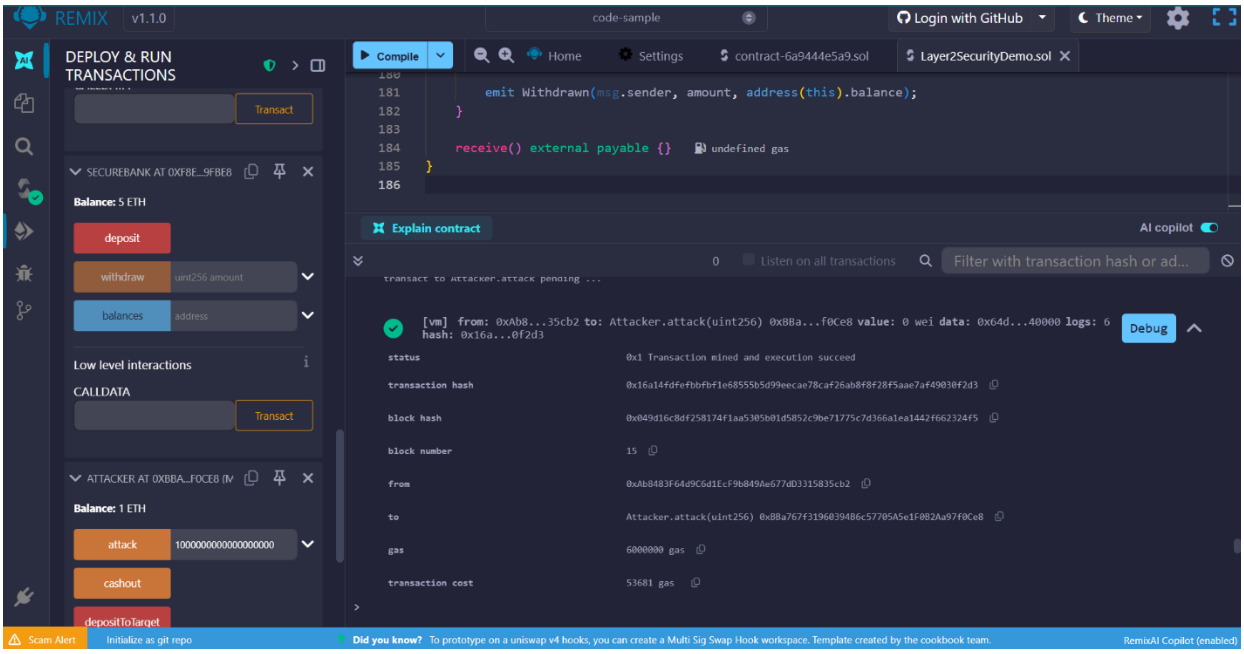

Figure 6: SecureBank balance after attempted exploit

7. Recommendations for Secure Deployment

- Conduct formal verification of smart contracts before production.
- Enforce multi-layered authentication and identity management for nodes.
- Apply real-time transaction monitoring and anomaly detection using AI-based tools.

- Use secure Layer 2 rollups (Optimistic or ZK-rollups) to enhance throughput without compromising integrity.
- Schedule periodic code audits and penetration testing to identify new vulnerabilities.

8. Conclusion

This experiment demonstrated that while Layer 2 blockchains offer scalability and performance benefits, they also introduce new attack surfaces — especially at the application layer through insecure smart contracts.
By applying structured threat modelling, secure coding patterns, and robust consensus mechanisms, organizations can build resilient and tamper-resistant financial applications on blockchain platforms.