BLOCK AUDIT

Automated AI-Powered Smart Contract Security Analysis with On-Chain Audit Logging

Presented by:

22k-4751 Muhammad Sohaib
22k-4760 Syeda Eizza Sakina

1. Introduction

Smart contracts are at the core of decentralized applications, but their immutability makes vulnerabilities catastrophic. This project builds an **automated audit pipeline** combining **static analysis**, **AI-based interpretation**, and **blockchain-based audit logging**, allowing developers to audit contracts and immutably store audit results for future validation.

2. Project Objectives

- Automate the detection of smart contract vulnerabilities.
- Generate human-readable audit summaries using Groq’s LLaMA-4.
- Store audit hashes and metadata on a local blockchain (Ganache).
- Fetch verified contracts directly from Etherscan or support `.sol` uploads.

3. System Architecture

Input Layer

- Upload local Solidity file or fetch contract source from Etherscan.

Processing Layer

1. **Static Analysis** with Slither (performs deep semantic vulnerability detection).
2. **AI Summary Generation** using Groq's LLaMA-4 (via API).
3. **Hashing and Metadata Generation** (SHA-256, timestamp, address).

Storage Layer

- Results are stored on-chain using a Solidity contract deployed to **Ganache**.

Interaction Layer

- CLI prompts guide the user through every step: upload/fetch, audit, AI analysis, and optional (pretend) report review and blockchain save.

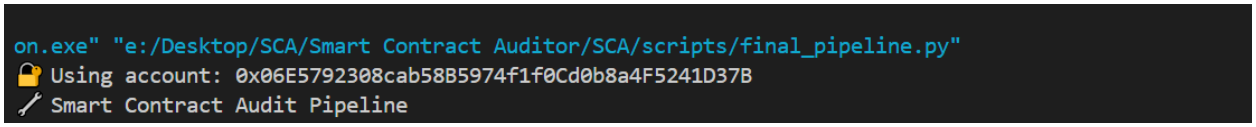
4. Core Components

Component	Description
`final_pipeline.py`	Orchestrates the end-to-end pipeline via CLI
`fetch_from_etherscan.py`	Fetches verified source code using Etherscan API
`analyze_with_slither.py`	Runs static analysis with appropriate Solidity version
`ai_analyze.py`	Uses Groq’s LLaMA-4 for summarizing vulnerabilities
`store_results_onchain.py`	Hashes the audit summary and saves it on-chain
`AuditResults.sol`	Solidity smart contract to store audit metadata

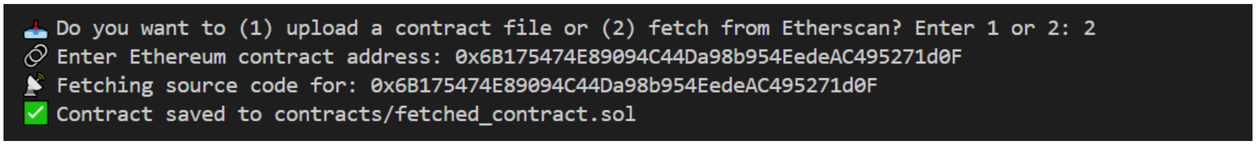
5. Workflow Summary

graph TD

A[Upload .sol file or Fetch from Etherscan]

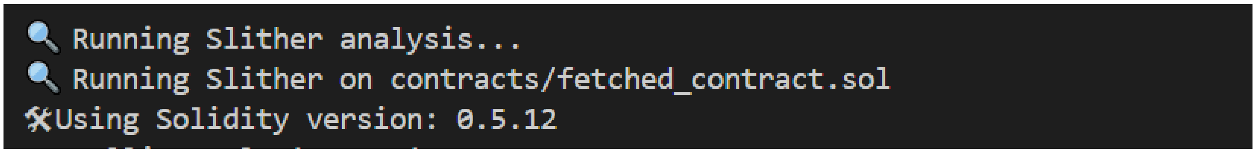


```
on.exe" "e:/Desktop/SCA/Smart Contract Auditor/SCA/scripts/final_pipeline.py"
🔐 Using account: 0x06E5792308cab58B5974f1f0Cd0b8a4F5241D37B
🔧 Smart Contract Audit Pipeline
```

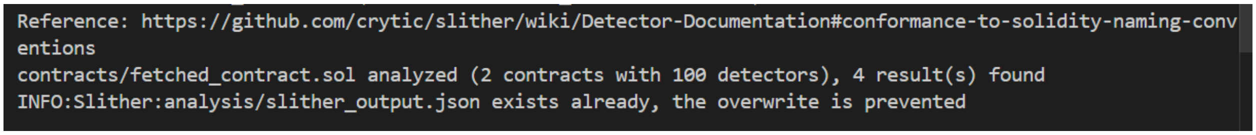


```
📥 Do you want to (1) upload a contract file or (2) fetch from Etherscan? Enter 1 or 2: 2
🔗 Enter Ethereum contract address: 0x6B175474E89094C44Da98b954EedeAC495271d0F
📡 Fetching source code for: 0x6B175474E89094C44Da98b954EedeAC495271d0F
✅ Contract saved to contracts/fetched_contract.sol
```

B[Run Slither Analysis]

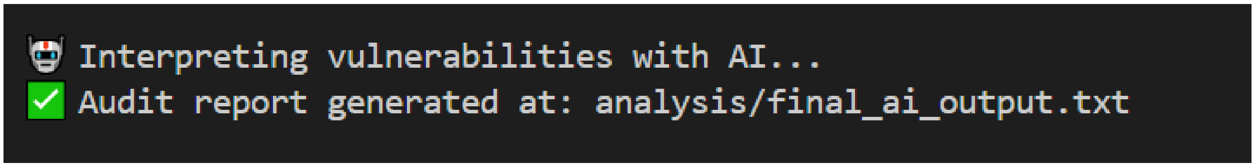


```
🔍 Running Slither analysis...
🔍 Running Slither on contracts/fetched_contract.sol
🛠Using Solidity version: 0.5.12
```



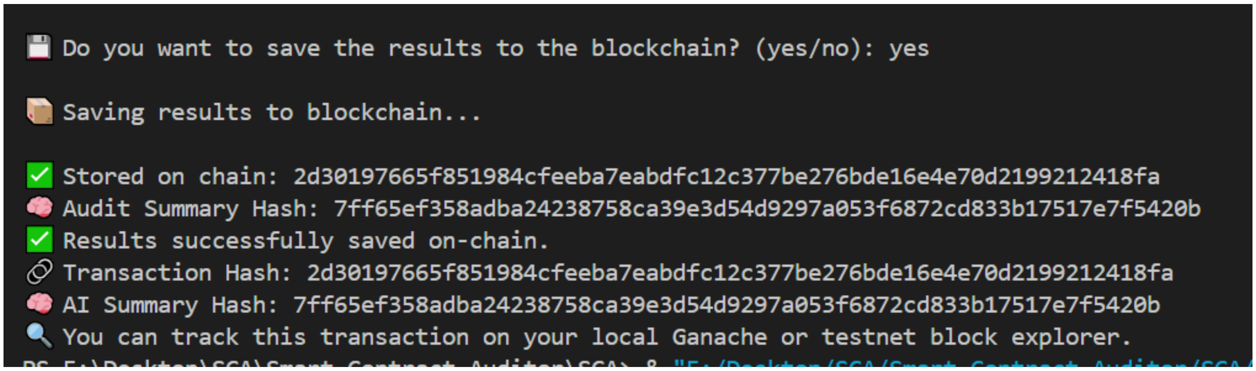
```
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conv
entions
contracts/fetched_contract.sol analyzed (2 contracts with 100 detectors), 4 result(s) found
INFO:Slither:analysis/slither_output.json exists already, the overwrite is prevented
```

C[Generate AI Summary (Groq)]



```
🤖 Interpreting vulnerabilities with AI...
✅ Audit report generated at: analysis/final_ai_output.txt
```

D[Store on Ganache Blockchain]



```
💾 Do you want to save the results to the blockchain? (yes/no): yes

📦 Saving results to blockchain...

✅ Stored on chain: 2d30197665f851984cfeeba7eabdfc12c377be276bde16e4e70d2199212418fa
🧠 Audit Summary Hash: 7ff65ef358adba24238758ca39e3d54d9297a053f6872cd833b17517e7f5420b
✅ Results successfully saved on-chain.
🔗 Transaction Hash: 2d30197665f851984cfeeba7eabdfc12c377be276bde16e4e70d2199212418fa
🧠 AI Summary Hash: 7ff65ef358adba24238758ca39e3d54d9297a053f6872cd833b17517e7f5420b
🔍 You can track this transaction on your local Ganache or testnet block explorer.
```

E[Display]

A --> B --> C --> D --> E

6. Smart Contract: AuditResults.sol

```
struct Audit {
    address submitter;
    string contractAddress;
    string aiSummaryHash;
    string timestamp;
}
function submitAudit(string memory contractAddress, string memory aiSummaryHash, string
memory timestamp) public
```

Stored data:

- Audited contract reference (as string)
- SHA-256 hash of audit summary
- UTC timestamp ,Enables verification of audit history.

7. Technologies Used

Tool	Purpose
Python 3.12	Scripting and automation
Solidity 0.5.x – 0.8.x	Multi-version contract support
Slither	Static vulnerability analysis

Groq API	AI-based report summarization
Ganache	Local Ethereum blockchain
Web3.py	Smart contract interaction
dotenv	Secure configuration handling

🐞 8. Challenges and Fixes

Issue	Solution
`solc` version mismatch	Dynamically installed correct `solc` using `solcx`
ABI mismatch	Ensured `AuditResults.json` ABI matched deployed contract
`.rawTransaction` error	Replaced with correct `.raw_transaction`
Gas errors	Refilled Ganache accounts manually using GUI faucet
Timestamp type mismatch	Converted to string to match ABI expectations

9. Sample Audit Report Snippet

```
================================
SMART CONTRACT AUDIT REPORT
================================

Contract: uploaded_contract.sol
Audited Using: Slither + Groq LLaMA-4

Issue 1: Reentrancy in withdraw()
---------------------------------
Impact: High
Confidence: Medium

 AI Summary:
The withdraw() function lacks reentrancy protection...
```

10. Blockchain Output Example

Results stored on local chain: 0xABC123...
AI Output Hash: 6f2c9b8a0...

11. Future Enhancements

- Mainnet/testnet support (e.g., Sepolia, Polygon).
- IPFS support for decentralized report storage.
- Multi-auditor access control and authentication.
- Dashboard analytics on contract risk profiles.