

Cybersecurity Implementation Report

Blockchain-Enabled, Smart Contract-Based Adaptive Compliance Enforcement System

Task 4: Risk-Adaptive Compliance for IoT Ecosystems

Submitted By: Ojas

Domain: Cybersecurity | IoT Security | Blockchain Compliance

Supervisor: Selin Tör

Date: June 6, 2025

1. Executive Summary

This report documents the successful development and simulation of a blockchain-based smart contract system designed to enforce risk-adaptive compliance policies in large-scale, multi-domain IoT environments. Implemented in Solidity, the solution leverages a permissioned blockchain model to ensure secure, transparent, and auditable enforcement of compliance controls across heterogeneous industrial and healthcare IoT devices. The project fulfills all blockchain-related requirements as outlined in Task 4.

2. Objective

To design and prototype a **decentralized, blockchain-based automated compliance platform** that uses **smart contracts** to enforce **dynamic, risk-adaptive policies** tailored for heterogeneous IoT devices operating under overlapping industrial and regulatory domains.

3. Blockchain Architecture Overview

3.1. Permissioned Blockchain Design

- **Justification:** A permissioned blockchain (e.g., Quorum or private Ethereum) ensures controlled access, low latency, and organizational accountability.
- **Participants:** Compliance officers, industrial IoT operators, healthcare providers, regulators.
- **Security Posture:** Only authorized entities can deploy rules or update device status, enforced via role-based access control in the contract.

3.2. Smart Contract Role

- **Core Functionality:**
 - Encode regulatory compliance rules.
 - Accept dynamic device risk scores.

- Enforce access decisions (allow/block) based on risk thresholds.
- **Auditability:** All actions (rule additions, status changes) are logged on-chain via immutable event logs.

4. Smart Contract Implementation

4.1. Contract Name: AdaptiveCompliance

4.2. Language: Solidity (v0.8.0)

4.3. Features Implemented:

Feature	Description
Rule Addition	Admins define compliance rules with domain, description, and risk threshold.
Risk-Adaptive Enforcement	Device access is dynamically updated using provided risk scores.
Audit Logging	Events such as RuleAdded and DeviceStatusUpdated ensure auditability.
Access Control	Only admin (contract deployer) can modify rules or device states.

4.4. Core Contract Functions

- `addRule()`: Adds new compliance policies.
- `updateDeviceStatus()`: Updates whether a device is allowed based on its current risk score.
- `isDeviceAllowed()`: View function to verify device permission status.

The screenshot shows the Remix IDE interface with the `base.sol` file open. The code defines a smart contract named `AdaptiveCompliance`. It includes a struct for `ComplianceRule`, a mapping of rules by domain, and a mapping of device allowed status. It features a modifier `onlyAdmin` and two public functions: `addRule` and `updateDeviceStatus`. The `addRule` function adds a new rule to the mapping and emits an event. The `updateDeviceStatus` function updates the allowed status for a given device and also emits an event. The code uses the MIT license and is written in Solidity version 0.8.0.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0; 1/base.sol

contract AdaptiveCompliance {
    // PUSH1 costs 3 gas - this line costs 119 gas - 2978888 gas left
    address public admin;
    uint256 public ruleCount;

    struct ComplianceRule {
        string domain;          // Regulatory domain (Healthcare, Industrial)
        string description;     // Policy description
        uint256 riskThreshold; // Threshold for enforcement
        bool isActive;
    }

    mapping(uint256 => ComplianceRule) public rules;
    mapping(address => bool) public deviceAllowed;

    event RuleAdded(uint256 indexed ruleId, string domain, uint256 riskThreshold);
    event DeviceStatusUpdated(address indexed device, bool allowed);

    modifier onlyAdmin() {
        require(msg.sender == admin, "Only admin allowed");
        _;
    }

    constructor() { 802878 gas 777800 gas
        admin = msg.sender;
    }

    function addRule(string memory domain, string memory description, uint256 riskThreshold) public onlyAdmin { infinite gas
        rules[ruleCount] = ComplianceRule(domain, description, riskThreshold, true);
        emit RuleAdded(ruleCount, domain, riskThreshold);
        ruleCount++;
    }

    function updateDeviceStatus(address device, uint256 riskScore) public onlyAdmin { infinite gas
        bool allowed = true;
    }
}
```

5. Testing and Deployment Logs

The smart contract was deployed and tested using a local Ethereum development environment (e.g., Remix or Hardhat).

Functional Demonstration:

Action	Status	Details
Rule Added	Done	"block_ip" with risk threshold = 1
Device Evaluated	Done	Device with risk score 1 marked as allowed
Status Queried	Done	Device allowed status retrieved as true
Rule Audited	Done	On-chain event logs confirm proper tracking

DEPLOY & RUN TRANSACTIONS

Publish to IPFS

At Address 0x358AA13c52544ECCEF6B

Transactions recorded 9 ⓘ ➤

Deployed Contracts ⚡

▼ ADAPTIVECOMPLIANCE AT 0X3! 🔍 ✖

Balance: 0. ETH

addRule ⚡

domain: "block_ip"

description: ip block

riskThreshold: 1

🕒 Calldata 📅 Parameters **transact**

updateDeviceStatus ⚡

device: 0x358AA13c52544ECCEF6B0A1

riskScore: 1

🕒 Calldata 📅 Parameters **transact**

admin

0: address: 0x5B38Da6a701c568545dCfcB
03FcB875f56beddC4

This screenshot shows a user interface for deploying and running transactions on a blockchain. At the top, there's a section for publishing to IPFS. Below that, a button allows deployment to a specific address (0x358AA13c52544ECCEF6B). A summary shows 9 transactions recorded. The main area displays two deployed contracts: 'ADAPTIVECOMPLIANCE AT 0X3!' and 'updateDeviceStatus'. Each contract has fields for domain, description, risk threshold, and parameters. Buttons for 'Calldata' and 'Parameters' are shown, along with an orange 'transact' button. The 'ADAPTIVECOMPLIANCE' contract also has a 'transact' button. At the bottom, there's an 'admin' button and a message about a deployed address.

Blockchain Logs Extracted:

- Transaction Hash: 0x2d0b5af801...
- Gas Used: 141362 (rule addition), 55005 (risk update)
- Event Logs: Included for both RuleAdded and DeviceStatusUpdated

```
[vm] from: 0x5B3...eddC4 to: AdaptiveCompliance.updateDeviceStatus(address,uint256) 0x358...D5eE3 value: 0 wei data: 0xfee...00001 Debug ▾  
logs: 1 hash: 0x02d...71bf6  
status 0x1 Transaction mined and execution succeed  
transaction hash 0x02da6e07f4b8aa2db372b3680aa387b035fd308f4458760c4f4a59607971bf6 ⓘ  
block hash 0xc0da7b89d303a6d70a71a7e9f2fb45957126b98a8bb61ff89fc01f8064b87b9 ⓘ  
block number 9 ⓘ  
from 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 ⓘ  
to AdaptiveCompliance.updateDeviceStatus(address,uint256) 0x358AA13c52544ECCEF6B0ADD0f801012ADADSeE3 ⓘ  
gas gas ⓘ  
transaction cost 55005 gas ⓘ  
execution cost 33433 gas ⓘ  
input 0xfee...00001 ⓘ  
>  
  
"address": "0x358AA13c52544ECCEF6B0ADD0f801012ADADSeE3",  
"uint256 riskScore": "1"  
)  
decoded output {} ⓘ  
logs []  
[  
    {  
        "from": "0x358AA13c52544ECCEF6B0ADD0f801012ADADSeE3",  
        "topic": "0x239e6a9dd8eeb49ba23bc6c7f8cf9adb8dfea555ae818de3abd7705aaec7e",  
        "event": "DeviceStatusUpdated",  
        "args": {  
            "o": "0x358AA13c52544ECCEF6B0ADD0f801012ADADSeE3",  
            "r": true,  
            "device": "0x358AA13c52544ECCEF6B0ADD0f801012ADADSeE3",  
            "allowed": true  
        }  
    }  
]  
call to AdaptiveCompliance.admin  
  
call [call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: AdaptiveCompliance.admin() data: 0xf85...1a440 Debug ▾  
>  
  
call to AdaptiveCompliance.admin  
  
call [call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: AdaptiveCompliance.admin() data: 0xf85...1a440 Debug ▾  
from 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 ⓘ  
to AdaptiveCompliance.admin() 0x358AA13c52544ECCEF6B0ADD0f801012ADADSeE3 ⓘ  
execution cost 2549 gas (Cost only applies when called by a contract) ⓘ  
input 0xf85...1a440 ⓘ  
decoded input {} ⓘ  
decoded output {  
    "o": "address: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4"  
} ⓘ  
logs [] ⓘ ⓘ ⓘ  
call to AdaptiveCompliance.deviceAllowed  
>
```

```
call  [call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: AdaptiveCompliance.deviceAllowed(address) data: 0xda9...d5ee3 Debug ▾

from          0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 ⓘ
to           AdaptiveCompliance.deviceAllowed(address) 0x358AA13c52544ECCEF6B0ADD0f801012ADAD5eE3 ⓘ
execution cost    2833 gas (Cost only applies when called by a contract) ⓘ
input          0xda9...d5ee3 ⓘ
decoded input   {
    "address": "0x358AA13c52544ECCEF6B0ADD0f801012ADAD5eE3"
}
decoded output  {
    "0": "bool: true"
}
logs          [] ⓘ ⓘ

call to AdaptiveCompliance.isDeviceAllowed
```

```
call  [call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: AdaptiveCompliance.isDeviceAllowed(address) data: 0x629...d5ee3 Debug ▾

from          0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 ⓘ
to           AdaptiveCompliance.isDeviceAllowed(address) 0x358AA13c52544ECCEF6B0ADD0f801012ADAD5eE3 ⓘ
execution cost    2915 gas (Cost only applies when called by a contract) ⓘ
input          0x629...d5ee3 ⓘ
decoded input   {
    "address_device": "0x358AA13c52544ECCEF6B0ADD0f801012ADAD5eE3"
}
decoded output  {
    "0": "bool: true"
}
logs          [] ⓘ ⓘ

call to AdaptiveCompliance.ruleCount
```

```
CALL  [call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: AdaptiveCompliance.ruleCount() data: 0x16d...ce633 Debug ▾

from          0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 ⓘ
to           AdaptiveCompliance.ruleCount() 0x358AA13c52544ECCEF6B0ADD0f801012ADAD5eE3 ⓘ
execution cost    2447 gas (Cost only applies when called by a contract) ⓘ
input          0xf6b...cf633 ⓘ
decoded input   {}
decoded output  {
    "0": "uint256: 1"
}
logs          [] ⓘ ⓘ

call to AdaptiveCompliance.rules

CALL  [call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: AdaptiveCompliance.rules(uint256) data: 0x04d...00001 Debug ▾

>
```

```
CALL  [call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: AdaptiveCompliance.rules(uint256) data: 0x04d...00001 Debug ▾

from          0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 ⓘ
to           AdaptiveCompliance.rules(uint256) 0x358AA13c52544ECCEF6B0ADD0f801012ADAD5eE3 ⓘ
execution cost    10911 gas (Cost only applies when called by a contract) ⓘ
input          0x04d...00001 ⓘ
decoded input   {
    "uint256": "1"
}
decoded output  {
    "0": "string: domain",
    "1": "string: description",
    "2": "uint256: riskThreshold 0",
    "3": "bool: isActive false"
}
logs          [] ⓘ ⓘ
```

```
call  [call]  from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: AdaptiveCompliance.rules(uint256) data: 0x04d...00000

from          0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 ⓘ

to           AdaptiveCompliance.rules(uint256) 0x358AA13c52544ECCEF6B0ADD0f801012ADADSeZ3 ⓘ

execution cost    11413 gas (Cost only applies when called by a contract) ⓘ

input          0x04d...00000 ⓘ

decoded input   {
    "uint256": "0"
} ⓘ

decoded output  {
    "0": "string: domain block_ip",
    "1": "string: description ip block",
    "2": "uint256: riskThreshold 1",
    "3": "bool: isActive true"
} ⓘ
```

▼ ADAPTIVECOMPLIANCE AT 0X3!

Balance: 0. ETH

addRule

domain: "block_ip"

description: ip block

riskThreshold: 1

🕒 Calldata 🕒 Parameters **transact**

updateDeviceStatus

device: 0x358AA13c52544ECCEF6B0A1

riskScore: 1

🕒 Calldata 🕒 Parameters **transact**

admin

0: address: 0x5B38Da6a701c568545dCfcB
03FcB875f56beddC4

deviceAllowed

: 0x358AA13c52544ECCEF6B0A1

🕒 Calldata 🕒 Parameters **call**

0: bool: true

isDeviceAllow... 0x358AA13c52544ECCE ▾

0: bool: true

ruleCount

isDeviceAllow... 0x358AA13c52544ECCE

ruleCount

rules 0

0: string: domain block_ip
1: string: description ip block
2: uint256: riskThreshold 1
3: bool: isActive true

Low level interactions

CALldata

Transact

6. Security Evaluation

Security Element	Assurance
Immutability	Rules and status changes are recorded on-chain and cannot be altered post-factum.
Integrity	Smart contract logic enforces deterministic compliance behavior.
Access Control	onlyAdmin modifier ensures privileged operations are restricted.
Transparency	All compliance decisions are logged and verifiable via blockchain explorers or audit tools.

7. Compliance and Use Case Relevance

This implementation aligns with real-world needs in:

- **Healthcare:** Enforcing policies on sensitive data encryption or device integrity.

- **Industrial IoT:** Disabling or isolating malfunctioning or compromised operational devices.
- **Cross-Domain Risk Governance:** Enabling policy enforcement across regulatory boundaries (e.g., GDPR + HIPAA).

8. Conclusion

The implemented blockchain smart contract solution achieves the core objectives of Task 4. It demonstrates a secure, scalable, and adaptable model for automating compliance enforcement across diverse IoT ecosystems. The system provides real-time responsiveness to risk, immutability of decisions, and traceability for audits—all vital for regulatory alignment and cybersecurity assurance.

This prototype offers a strong foundation for future expansion, including:

- Oracle integration for real-time automated risk feed.
- UI dashboard for compliance officers.
- Scaling to consortium blockchain networks across organizations.

9. Appendices

Appendix A – Smart Contract Code Snippet

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract AdaptiveCompliance {
    address public admin;
    uint256 public ruleCount;

    struct ComplianceRule {
        string domain;          // Regulatory domain (Healthcare, Industrial)
        string description;    // Policy description
        uint256 riskThreshold; // Threshold for enforcement
        bool isActive;
    }
}
```

```
mapping(uint256 => ComplianceRule) public rules;
mapping(address => bool) public deviceAllowed;

event RuleAdded(uint256 indexed ruleId, string domain, uint256 riskThreshold);
event DeviceStatusUpdated(address indexed device, bool allowed);

modifier onlyAdmin() {
    require(msg.sender == admin, "Only admin allowed");
    _;
}

constructor() {
    admin = msg.sender;
}

function addRule(string memory domain, string memory description, uint256 riskThreshold) public onlyAdmin {
    rules[ruleCount] = ComplianceRule(domain, description, riskThreshold, true);
    emit RuleAdded(ruleCount, domain, riskThreshold);
    ruleCount++;
}

function updateDeviceStatus(address device, uint256 riskScore) public onlyAdmin {
    bool allowed = true;
    for (uint i = 0; i < ruleCount; i++) {
        if (rules[i].isActive && riskScore > rules[i].riskThreshold) {
            allowed = false;
            break;
        }
    }
    deviceAllowed[device] = allowed;
    emit DeviceStatusUpdated(device, allowed);
}
```

```
    }

function isDeviceAllowed(address device) public view returns (bool) {
    return deviceAllowed[device];
}

}
```

Appendix B – Sample Transaction Log Summary

- Event: RuleAdded — {domain: "block_ip", riskThreshold: 1}
- Event: DeviceStatusUpdated — {device: 0x358..., allowed: true}
- Block Number: 8–9
- Gas Costs: 119150–33433 (efficient execution)