DetectBox
Powered by UNSNARL

#DECENTRALISED PROACTIVE AUDIT WAR-ROOMS

Protocol Contract Security Assessment

PAYANT ESCROW LTD

AUGUST 29TH, 2023

TABLE OF CONTENTS

SUMMARY

This report has been prepared for Payant Escrow LTD to discover issues and vulnerabilities in the source code of the protocol contracts as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Static Analysis and Manual Review 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 experts.

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:

- 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

CERTIFICATION

This is to Certify that the **Payant Security Report** was prepared using gold standard web3 security standards with all standard and advanced checks deployed.

The **Main Auditors** Involved in the Audit was -
Samrat Gupta (@Sm4rty_)(Lead Auditor)
Rohan Jha (@rohan16___)

The Detect Warden Involved in the Audit was -
33Audits (@solidityauditor)

The Auditor mentioned hereby was chosen after self due-dilligence from the project's end.

All reported issues were either acknowledged or fixed by the project.

DetectBox's audit mitigation & residual vulnerability check did not find any major vulnerability to report within the Scope and requirements of the audit.

From
Team DetectBox

Payant Escrow Ltd
Security Assessment Report

ABOUT DETECTBOX

DetectBox by **UNSNARL** brings to you world's first Decentralised Audit War-Rooms. An intense collaborative Environment where chosen auditors/audit team, project's decision makers, Projects developers and Detect-Wardens join hands to conduct rigorous security audits with absolute transparency followed by a **Proof of Audit (POA).**

We at DetectBox not only find bugs in smart-contracts but also take a deep look into the project's **tokenomics**, **white-paper**, **business logic**, **functional flows** and overall development health.

At DetectBox, you get to choose from a pool of Independent Security researchers which is curated by strong due-diligence. List your project requirements, verify the auditor's past work from their profiles and choose the auditor that rightly fits your project's needs and your budget.

Our auditors have successfully completed **200+ audits**, found over **2200+ vulnerabilities** and secured over **$102M+ worth of TVL.**

To know more about us visit - detectbox.io
To see our docs - https://unsnarl.gitbook.io/detectbox/

Yours Securely
UNSNARL

OVERVIEW

Project Summary

Payant is a platform that facilitates seamless invoicing, payment, and fund withdrawal processes for clients and contractor. It employs smart contracts to automate these processes while ensuring security, transparency, and trust through blockchain technology.

Audit Summary

A time-boxed independent security assessment of the Payant Protocol contract was done by Samrat Gupta (@Sm4rty_), Rohan Jha (@rohan16___) and Team DetectBox with a focus on the security aspects of the application's implementation.
We performed the security assessment based on the agreed scope, following our approach and methodology. Based on our scope and our performed activities, our security assessment revealed 2 High and 3 Low severity issues. Additionally, 4 Informational suggestion was also made which, if resolved appropriately, may improve the quality of the Project’s Smart contract.

Audit Timeline: 16th August’23 - 24th August’23

Website:
https://www.payant.io/

OVERVIEW

Audit Scope

The code under review is composed of 922 nLOC in the Solidity language. It also includes 354 nLOC of scripts written in the Rain language.

File	Lines	nSLOC	Complex. Score
contracts/flow/basic/Flow.sol	76	54	32
contracts/factory/CloneFactory.sol	**34**	**23**	16
contracts/flow/erc721/FlowERC721.sol	294	226	130
contracts/interpreter/shared/RainterpreterExpressionDeployer.sol	348	191	100
contracts/interpreter/shared/RainterpreterStore.sol	58	27	20
contracts/interpreter/shared/Rainterpreter.sol	109	69	36
contracts/interpreter/ops/context/OpContext.sol	47	21	5
contracts/interpreter/ops/context/OpContextColumnHash.sol	53	22	7
contracts/interpreter/ops/error/OpEnsure.sol	43	27	5
contracts/interpreter/ops/evm/OpTimestamp.sol	29	14	3
contracts/interpreter/ops/math/logic/OpEqualTo.sol	38	21	11
contracts/interpreter/ops/math/logic/OpAny.sol	52	35	17
contracts/interpreter/ops/math/logic/OpEvery.sol	51	34	17
contracts/interpreter/ops/math/logic/OpGreaterThan.sol	36	21	11
contracts/interpreter/ops/math/logic/OpLessThan.sol	36	21	11
contracts/interpreter/ops/math/logic/OpIsZero.sol	36	21	9
contracts/interpreter/ops/store/OpSet.sol	45	30	7
contracts/interpreter/ops/store/OpGet.sol	66	38	13
Total	**1451**	895	**450**

OVERVIEW

RainScripts:

File	Lines	nSLOC
addDeliverables.template.rain	38	30
approveDeliverables.template.rain	40	31
cancel.template.rain	40	28
cancelMediation.template.rain	37	27
clientWithdraw.template.rain	49	37
contractorWithdraw.template.rain	45	36
feedbackDeliverables.template.rain	40	32
mediationClientWithdraw.template.rain	44	34
mediationContractorWithdraw.template.rain	44	34
mediationResult.template.rain	45	34
startMediation.template.rain	40	31
Total:	**462**	**354**

Lead Auditor -

Samrat Gupta - https://app.detectbox.io/profile/sm4rty

Team -

Rohan Jha - https://app.detectbox.io/profile/Rohan16

Detect Warden -

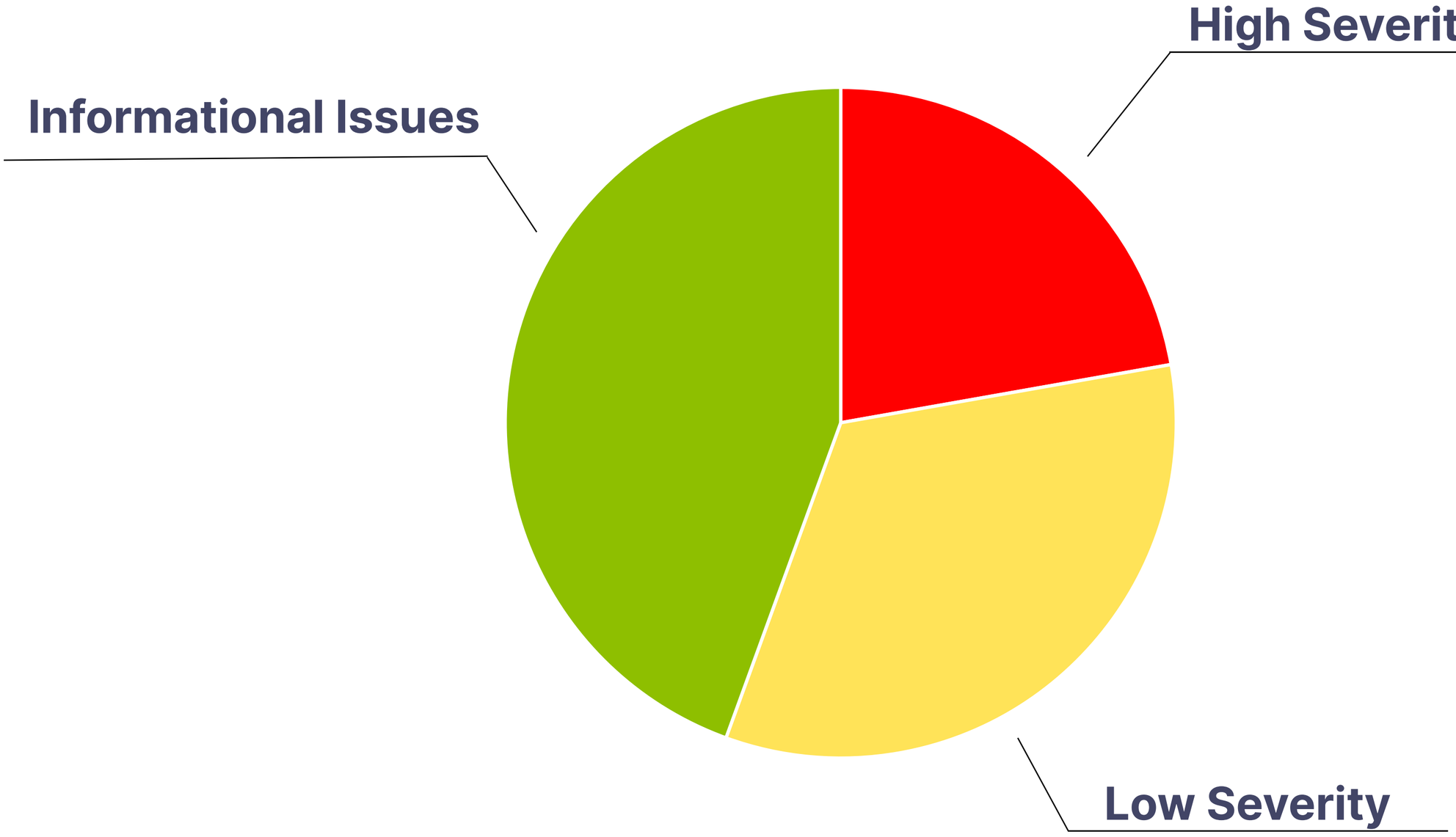
33Audits - https://app.detectbox.io/profile/33audits

FINDINGS

Detailed Summary of Findings

Sl. No.	Name	Severity
H-01	Contractor can cancel the contract after mediation has been started leading to fund loss for Protocol.	High
H-02	Client can approve the Deliverables after mediation has been started leading to the fund loss for Protocol.	High
L-01	Use the latest version of open zeppelin libraries	Low
L-02	Missing Natspec	Low
L-03	Incomplete Documentation	Low
I-01	Multiple typos in the Payant contracts	Informational
I-02	Removal of Commented-Out Code for Better Code Quality	Informational
I-03	Using unnamed mappings	Informational
I-04	Event is not properly Indexed	Informational

FINDINGS



Static Analysis

No major issues were found. Some false positive errors were reported by the tool. All the other issues have been categorized above according to their level of severity.

Manual Review

High Severity Issues

H-01. Contractor can cancel the contract after mediation has been started leading to fund loss for Protocol.

FINDINGS

The Payant contract allows the contractor to cancel the contract after the mediation process has been started. This can be exploited by the contractor to avoid paying the mediation fees, which are paid by the payant protocol in advance.

How does the Mediation Process work?

According to the Project, There's a fee related to mediation that Payant will pay in advance. However, when the result of the dispute is implemented, Payant will take out the cut of the fee with no additional charge. Example: Invoice is $1000, the fee is $100. Side A wins and can claim the full amount. We will send $900 to side A and $100 to the wallet managing the Kleros integration

Impact:

If this vulnerability is exploited, the payant protocol could lose the mediation fees that were paid in advance. This could have a significant financial impact on the protocol.

Code Snippets:

```
: ensure(equal-to(signer contractor)),
: ensure(equal-to(signedcontexthash datahash)),
: ensure(equal-to(context-column-hash<1>() datahash)),
: ensure(equal-to(operation cancel-operation)),
: ensure(is-zero(get(one))),
```

FINDINGS

Proof of Concept with Test-Cases

- A client and a contractor enter into a contract.
- The contract includes a mediation clause that allows the parties to resolve disputes through mediation.
- The payant protocol pays the mediation fees in advance.
- The mediation process has been started.
- After some mutual agreement between both parties, the contractor cancels the contract.
- The client receives the full amount of the contract without paying the mediation fees and the payant lost the mediation fees paid in advanced.

```
import { SignerWithAddress } from "@nomiclabs/hardhat-ethers/signers";
import {
  EvaluableStructOutput,
  FlowInitializedEvent,
} from "../typechain/contracts/flow/FlowCommon";
import { Flow, ReserveToken18 } from "../typechain";
import { ethers } from "hardhat";
import { RESERVE_ONE, assertError, basicDeploy, getEvents } from "../utils";
import { arrayify, keccak256, solidityKeccak256 } from "ethers/lib/utils";
import {
  ADD_DELIVERABLE_OPERATION,
  D7,
  CANCEL_OPERATION,
  START_MEDIATION_OPERATION,
  APPROVE_DELIVERABLE_OPERATION,
  MEDIATION_RESULT_OPERATION,
  createFlowConfig,
} from "./utils";
import { deployFlowClone } from "../utils/deploy/flow/basic/deploy";
import { cloneFactory, implementation } from "./deploy.test";
import assert from "assert";
import { SignedContextV1Struct } from "../typechain/contracts/lobby/Lobby";
```

FINDINGS

```
describe("start Mediation test", () => {
  let signers: SignerWithAddress[];
  let deployer: SignerWithAddress;
  let client: SignerWithAddress; // caller
  let contractor: SignerWithAddress; // caller
  let erc20: ReserveToken18;
  let flowContract: Flow;
  let endDate: number;
  let invoiceDataHash: string;
  let dispatchCancel: EvaluableStructOutput,
    dispatchClientWithdraw: EvaluableStructOutput,
    dispatchContractorWithdraw: EvaluableStructOutput,
    dispatchAddDeliverables: EvaluableStructOutput,
    dispatchApproveDeliverables: EvaluableStructOutput,
    dispatchFeedbackDeliverables: EvaluableStructOutput,
    dispatchStartMediation: EvaluableStructOutput,
    dispatchClientMediationWithdraw: EvaluableStructOutput,
    dispatchContractorMediationWithdraw: EvaluableStructOutput,
    dispatchMediationResult: EvaluableStructOutput;

  beforeEach(async () => {
    signers = await ethers.getSigners();
    deployer = signers[0];
    client = signers[1];
    contractor = signers[2];

    erc20 = (await basicDeploy("ReserveToken18", {})) as ReserveToken18;
    await erc20.initialize();

    endDate = Date.now();
```

```
    invoiceDataHash = solidityKeccak256(
      ["uint256[]"],
      [[client.address, contractor.address, RESERVE_ONE, endDate]]
    );

    const flowConfig = await createFlowConfig(
      erc20.address,
      invoiceDataHash,
      endDate,
      D7,
      deployer.address
    );

    const { flow } = await deployFlowClone(
      deployer,
      cloneFactory,
      implementation,
      flowConfig
    );

    const flowInitialized = (await getEvents(
      flow.deployTransaction,
      "FlowInitialized",
      flow
    )) as FlowInitializedEvent["args"][];
```

FINDINGS

```
  flowContract = flow;
  dispatchCancel = flowInitialized[0].evaluable;
  dispatchClientWithdraw = flowInitialized[1].evaluable;
  dispatchContractorWithdraw = flowInitialized[2].evaluable;
  dispatchAddDeliverables = flowInitialized[3].evaluable;
  dispatchApproveDeliverables = flowInitialized[4].evaluable;
  dispatchFeedbackDeliverables = flowInitialized[5].evaluable;
  dispatchStartMediation = flowInitialized[6].evaluable;
  dispatchClientMediationWithdraw = flowInitialized[7].evaluable;
  dispatchContractorMediationWithdraw = flowInitialized[8].evaluable;
  dispatchMediationResult = flowInitialized[9].evaluable;
});

it.only("contractor should be able to cancel the contract after starting mediation", async () => {
  await erc20.transfer(flowContract.address, RESERVE_ONE);

  assert.deepEqual(await erc20.balanceOf(flowContract.address), RESERVE_ONE);

  const context = [client.address, contractor.address, RESERVE_ONE, endDate];

  const addDeliverableContext = [
    invoiceDataHash,
    ADD_DELIVERABLE_OPERATION,
    ethers.BigNumber.from(
      keccak256([
        ...Buffer.from(
          "https://emn178.github.io/online-tools/keccak_256.html"
        ),
      ])
    ),
  ];

  const addDeliverableHash = solidityKeccak256(["uint256[]"], [addDeliverableContext]);
  const addDeliverableSignature = await contractor.signMessage(arrayify(addDeliverableHash));
  const addDeliverableSignedContext: SignedContextV1Struct[] = [
    {
      signature: addDeliverableSignature,
      signer: contractor.address,
      context: addDeliverableContext,
    },
  ];
  await flowContract
    .connect(contractor)
    .flow(dispatchAddDeliverables, context, addDeliverableSignedContext);
    console.log("The contractor added the Deliverables");


  const startMediationcontext = [invoiceDataHash, START_MEDIATION_OPERATION];

  const startMediationhash = solidityKeccak256(["uint256[]"], [startMediationcontext]);
  const startMediationSignature = client.signMessage(arrayify(startMediationhash));
  const startMediationsignedContext: SignedContextV1Struct[] = [
    {
      signature: startMediationSignature,
      signer: client.address,
      context: startMediationcontext,
    },
  ];

  await flowContract
    .connect(client)
    .flow(dispatchStartMediation, context, startMediationsignedContext);
    console.log("Client Started Mediation");
```

FINDINGS

```
    const cancelContext = [invoiceDataHash, CANCEL_OPERATION];
    const cancelHash = solidityKeccak256(["uint256[]"], [cancelContext]);
    const cancelSignature = await contractor.signMessage(arrayify(cancelHash));
    const cancelSignedContext: SignedContextV1Struct[] = [
      {
        signature: cancelSignature,
        signer: contractor.address,
        context: cancelContext,
      },
    ];
    await flowContract
      .connect(contractor)
      .flow(dispatchCancel, context, cancelSignedContext);
      console.log("The contractor cancelled the contract");

      const client_balance = await erc20.balanceOf(client.address);
      console.log("Balance of Client:", client_balance);

      const flow_contract_balance = await erc20.balanceOf(flowContract.address);
      console.log("Balance of flow contract:", flow_contract_balance);

    assert.deepEqual(await erc20.balanceOf(client.address), RESERVE_ONE);

  });
});
```

```
  start Mediation test
The contractor added the Deliverables
Client Started Mediation
The contractor cancelled the contract
Balance of Client: BigNumber { value: "1000000" }
Balance of flow contract: BigNumber { value: "0" }
    ✓ contractor should be able to cancel the contract after starting mediation
```

Remediation:

The protocol should add a check to make sure that the mediation process has not been started before allowing the contractor to cancel the contract.

FINDINGS

H-02. Client can approve the Deliverables after mediation has been started leading to the fund loss for Protocol.

The Payant contract allows the client to approve deliverables to the contractor after the mediation process has been started. This can be exploited by the client to avoid paying the mediation fees, which are paid by the payant protocol in advance.

How does the Mediation Process work?

According to the Project, There's a fee related to mediation that Payant will pay in advance. However, when the result of the dispute is implemented, Payant will take out the cut of the fee with no additional charge. Example: Invoice is $1000, the fee is $100. Side A wins and can claim the full amount. We will send $900 to side A and $100 to the wallet managing the Kleros integration

Impact:

If this vulnerability is exploited, the payant protocol could lose the mediation fees that were paid in advance. This could have a significant financial impact on the protocol.

Code Snippets:

```
: ensure(is-zero(get(one))), /** Ensure contract is not setteled */
: ensure(get(deliverables-flag)), /** Ensure deliverables are added */
: ensure(equal-to(client signer)), /** Ensure client is calling the function*/
: ensure(equal-to(datahash signedcontexthash)), /** Ensure contract is for same invoice using datahash */
: ensure(equal-to(signedcontexthash context-column-hash<1>())), /** Ensure signedContext is same as callerContext */
: ensure(equal-to(operationhash approve-deliverable-operation)), /** Script only for approve delivrables */
```

FINDINGS

Proof of Concept with Test-Cases

- A client and a contractor enter into a contract.
- The contract includes a mediation clause that allows the parties to resolve disputes through mediation.
- The payant protocol pays the mediation fees in advance.
- The mediation process has been started.
- After some mutual agreement between both parties, the client approves the deliverables of Contractor.
- The contractor receives the full amount of the contract without paying the mediation fees and the payant lost the mediation fees paid in advanced.

```
import { SignerWithAddress } from "@nomiclabs/hardhat-ethers/signers";
import {
  EvaluableStructOutput,
  FlowInitializedEvent,
} from "../typechain/contracts/flow/FlowCommon";
import { Flow, ReserveToken18 } from "../typechain";
import { ethers } from "hardhat";
import { RESERVE_ONE, assertError, basicDeploy, getEvents } from "../utils";
import { arrayify, keccak256, solidityKeccak256 } from "ethers/lib/utils";
import {
  ADD_DELIVERABLE_OPERATION,
  D7,
  CANCEL_OPERATION,
  START_MEDIATION_OPERATION,
  APPROVE_DELIVERABLE_OPERATION,
  MEDIATION_RESULT_OPERATION,
  createFlowConfig,
} from "./utils";
import { deployFlowClone } from "../utils/deploy/flow/basic/deploy";
import { cloneFactory, implementation } from "./deploy.test";
import assert from "assert";
import { SignedContextV1Struct } from "../typechain/contracts/lobby/Lobby";

describe("start Mediation test", () => {
  let signers: SignerWithAddress[];
  let deployer: SignerWithAddress;
  let client: SignerWithAddress; // caller
  let contractor: SignerWithAddress; // caller
  let erc20: ReserveToken18;
  let flowContract: Flow;
  let endDate: number;
  let invoiceDataHash: string;
```

FINDINGS

```
let dispatchCancel: EvaluableStructOutput,
  dispatchClientWithdraw: EvaluableStructOutput,
  dispatchContractorWithdraw: EvaluableStructOutput,
  dispatchAddDeliverables: EvaluableStructOutput,
  dispatchApproveDeliverables: EvaluableStructOutput,
  dispatchFeedbackDeliverables: EvaluableStructOutput,
  dispatchStartMediation: EvaluableStructOutput,
  dispatchClientMediationWithdraw: EvaluableStructOutput,
  dispatchContractorMediationWithdraw: EvaluableStructOutput,
  dispatchMediationResult: EvaluableStructOutput;

beforeEach(async () => {
  signers = await ethers.getSigners();
  deployer = signers[0];
  client = signers[1];
  contractor = signers[2];

  erc20 = (await basicDeploy("ReserveToken18", {})) as ReserveToken18;
  await erc20.initialize();

  endDate = Date.now();

  invoiceDataHash = solidityKeccak256(
    ["uint256[]"],
    [[client.address, contractor.address, RESERVE_ONE, endDate]]
  );

  const flowConfig = await createFlowConfig(
    erc20.address,
    invoiceDataHash,
    endDate,
    D7,
    deployer.address
  );

  const { flow } = await deployFlowClone(
    deployer,
    cloneFactory,
    implementation,
    flowConfig
  );

  const flowInitialized = (await getEvents(
    flow.deployTransaction,
    "FlowInitialized",
    flow
  )) as FlowInitializedEvent["args"][];

  flowContract = flow;
  dispatchCancel = flowInitialized[0].evaluable;
  dispatchClientWithdraw = flowInitialized[1].evaluable;
  dispatchContractorWithdraw = flowInitialized[2].evaluable;
  dispatchAddDeliverables = flowInitialized[3].evaluable;
  dispatchApproveDeliverables = flowInitialized[4].evaluable;
  dispatchFeedbackDeliverables = flowInitialized[5].evaluable;
  dispatchStartMediation = flowInitialized[6].evaluable;
  dispatchClientMediationWithdraw = flowInitialized[7].evaluable;
  dispatchContractorMediationWithdraw = flowInitialized[8].evaluable;
  dispatchMediationResult = flowInitialized[9].evaluable;
});
```

FINDINGS

```
it.only("client should be able to approve the contractor after starting mediation", async () => {
  await erc20.transfer(flowContract.address, RESERVE_ONE);

  assert.deepEqual(await erc20.balanceOf(flowContract.address), RESERVE_ONE);

  const context = [client.address, contractor.address, RESERVE_ONE, endDate];

const addDeliverableContext = [
  invoiceDataHash,
  ADD_DELIVERABLE_OPERATION,
  ethers.BigNumber.from(
    keccak256([
      ...Buffer.from(
        "https://emn178.github.io/online-tools/keccak_256.html"
      ),
    ])
  ),
];

const addDeliverableHash = solidityKeccak256(["uint256[]"], [addDeliverableContext]);
const addDeliverableSignature = await contractor.signMessage(arrayify(addDeliverableHash));
const addDeliverableSignedContext: SignedContextV1Struct[] = [
  {
    signature: addDeliverableSignature,
    signer: contractor.address,
    context: addDeliverableContext,
  },
];
await flowContract
  .connect(contractor)
  .flow(dispatchAddDeliverables, context, addDeliverableSignedContext);
  console.log("The contractor added the Deliverables");

const startMediationcontext = [invoiceDataHash, START_MEDIATION_OPERATION];

const startMediationhash = solidityKeccak256(["uint256[]"], [startMediationcontext]);
const startMediationSignature = client.signMessage(arrayify(startMediationhash));
const startMediationsignedContext: SignedContextV1Struct[] = [
  {
    signature: startMediationSignature,
    signer: client.address,
    context: startMediationcontext,
  },
];

await flowContract
  .connect(client)
  .flow(dispatchStartMediation, context, startMediationsignedContext);
  console.log("Client Started Mediation");


const approveDeliverableContext = [
  invoiceDataHash,
  APPROVE_DELIVERABLE_OPERATION,
  ethers.BigNumber.from(
    keccak256([
      ...Buffer.from(
        "https://emn178.github.io/online-tools/keccak_256.html"
      ),
    ])
  ),
];
```

FINDINGS

```
    const approveDeliverableHash = solidityKeccak256(
      ["uint256[]"],
      [approveDeliverableContext]
    );
    const approveDeliverableSignature = client.signMessage(
      arrayify(approveDeliverableHash)
    );
    const approveDeliverableSignedContext: SignedContextV1Struct[] = [
      {
        signature: approveDeliverableSignature,
        signer: client.address,
        context: approveDeliverableContext,
      },
    ];

    await flowContract
      .connect(client)
      .flow(
        dispatchApproveDeliverables,
        context,
        approveDeliverableSignedContext
      );
      console.log("The client approved the Deliverables");

      const flow_contract_balance = await erc20.balanceOf(flowContract.address);
      console.log("Balance of Flow Contract:", flow_contract_balance);

      const contractor_balance = await erc20.balanceOf(contractor.address);
      console.log("Balance of Contractor:", contractor_balance);

    assert.deepEqual(await erc20.balanceOf(contractor.address), RESERVE_ONE);

  });
});
```

```
  start Mediation test
The contractor added the Deliverables
Client Started Mediation
The client approved the Deliverables
Balance of Flow Contract: BigNumber { value: "0" }
Balance of Contractor: BigNumber { value: "1000000" }
    ✓ client should be able to approve the contractor after starting mediation
```

Remediation:

The protocol should add a check to make sure that the mediation process has not been started before allowing the client to approve the deliverables.

FINDINGS

Low Severity Issues

L-01. Use the latest version of open zeppelin libraries

The contract is using an outdated version of the OpenZeppelin (OZ) libraries. The latest version is 4.9.2, while the contract is using 0.8.3.
The latest release usually contains a fix for lots of vulnerabilities and is a lot more optimized. Check out here for more details.

Impact:

Using an outdated version of the OZ libraries can increase the risk of vulnerabilities in the contract. The latest release of the OZ libraries usually contains fixes for known vulnerabilities, as well as performance improvements.

Instances:

package.json#L56-L57

```
"@openzeppelin/contracts-upgradeable": "=4.8.3",
```

Recommended Mitigation Steps:

The contract should be upgraded to the latest version of the OZ libraries. This can be done by updating the @openzeppelin/contracts-upgradeable dependency in the package.json file.

FINDINGS

L-02. Missing Natspec

Solidity contracts can use a special form of comments to provide rich documentation for functions, return variables and more. This special form is named the Ethereum Natural Language Specification Format (NatSpec).
There are multiple functions in a few smart contracts in scope that lack NatSpec comments, which are essential for providing clear and comprehensive documentation for functions, return variables, and other contract elements.

Instances:

The following smart contracts in scope lack NatSpec comments in multiple functions:
CloneFactory.sol#L16
Flow.sol#L31
FlowERC721.sol#L139
OpEnsure.sol#L23

Recommended Mitigation Steps:

Although it's Optional, To enhance the quality and usability of the smart contracts, it is strongly recommended to add NatSpec comments to various parts of the code.

L-03. Incomplete Documentation

FINDINGS

The README.md file in the Payant GitHub repo contains incomplete documentation for the following:

- Mediation process
- Client Feedback process
- Many other details of the functionalities
- The markdown is not properly rendered
- The documentation is not in proper format
- There are many spelling mistakes

Impact:

Without proper documentation, it will be difficult for users or developers to understand the contract's intentions, leading to mistakes, bugs, and security vulnerabilities during maintenance or updates

Instances:

https://github.com/Payant/payant/blob/751e1d24419eb8ebaab7f66615a5a7705ef75fb3/README.md#L1-L2

Recommended Mitigation Steps:

- An overview of each functionality
- A detailed explanation of the mediation process and feedback process, including the steps involved and the actors involved.
- A list of all the functionalities that are not documented, along with a brief explanation of each functionality
- The markdown is not properly reendered. Need to Fix that.
- Add a section for actors involved like contractors, clients, and mediators.
- Check for all spelling mistakes and fix it.

FINDINGS

Informational Issues

I-01. Multiple typos in the Payant contracts

There are multiple places in contracts and scripts where there is a typing mistake. The typos should be corrected.

Instances:

In the approveDeliverables.template.rain script, the word "deliverables" is misspelled as "delivrables".

```
: ensure(equal-to(operationhash approve-deliverable-operation)), /*
* Script only for approve delivrables */
```

In the cancelMediation.test.ts test file, the word "already" is misspelled as "alredy" and canceled is misspelled as “canceld”

```
it("should fail to cancel mediation if alredy canceld or not starte
d", async () => {
```

Recommended Mitigation Steps:

In the README file, there are several typos. Check it out in the below link:
https://github.com/Payant/payant/blob/751e1d24419eb8ebaab7f66615a5a7705ef75fb3/README.md

FINDINGS

I-02. Removal of Commented-Out Code for Better Code Quality

The approveDeliverables.template.rain script contains lines of code that are commented out but serve no purpose. Commented-out code segments do not contribute to the functionality of the contracts but can clutter the codebase, making it harder for developers to understand the actual logic of the contracts.

Instances:

approveDeliverables.template.rain#L22-L23

```
/* signedfeedbackhash: context<3 2>(), * keccak256 hash of delivera
bles */
```

I-03. Using unnamed mappings

The contracts use unnamed mappings, which can make it difficult to understand the purpose of each mapping. Consider using named mappings to make it easier to understand the purpose of each mapping. This can make the code less readable and maintainable.

Instances:

interpreter/shared/RainterpreterStore.sol:28:

```
mapping(FullyQualifiedNamespace => mapping(uint256 => uint256))
```

FINDINGS

I-04. Event is not properly Indexed

Multiple events in the RainInterpreterExpressionDeployer.sol contract are not indexed efficiently. Each event should use three indexed fields if there are three or more fields. This means that off-chain tools that parse events will have to scan more data to find the information they need.

Instances:

RainterpreterExpressionDeployer.sol#L89

```
event NewExpression(
        address sender,
        bytes[] sources,
        uint256[] constants,
        uint256[] minOutputs
    );
```

RainterpreterExpressionDeployer.sol#L101

POST AUDIT CONCLUSION

Fixing the Findings

Sl. No.	Name	Status
H-01	Contractor can cancel the contract after mediation has been started leading to fund loss for Protocol.	Fixed
H-02	Client can approve the Deliverables after mediation has been started leading to the fund loss for Protocol.	Fixed
L-01	Use the latest version of open zeppelin libraries	Fixed
L-02	Missing Natspec	Fixed
L-03	Incomplete Documentation	Acknowledged
I-01	Multiple typos in the Payant contracts	Fixed
I-02	Removal of Commented-Out Code for Better Code Quality	Fixed
I-03	Using unnamed mappings	Fixed
I-04	Event is not properly Indexed	Acknowledged

APPENDIX

Finding Categories

Finding Categories

Centralization/Privilege findings refer to either feature logic or implementation of components that act against the nature of decentralization, such as explicit ownership or specialized access roles in combination with a mechanism to relocate funds.

Gas Optimization

Gas Optimization findings do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.

Logical Issue

Logical Issue findings detail a fault in the logic of the linked code, such as an incorrect notion on how block.timestamp works.

Language Specific

Language Specific findings are issues that would only arise within Solidity, i.e. incorrect usage of private or delete.

Coding Style

Coding Style findings usually do not affect the generated byte-code but rather comment on how to make the codebase more legible and, as a result, easily maintainable.

DISCLAIMER

DetectBox (by UNSNARL) has conducted an independent security review to verify the integrity of and highlight any vulnerabilities, bugs or errors, intentional or unintentional, that may be present in the codebase that were provided for the scope of this Engagement.

Blockchain technology and the cryptographic assets it supports are nascent technologies. This makes them extremely volatile assets. Any assessment report obtained on such volatile and nascent assets may include unpredictable results which may lead to positive or negative outcomes.

In some cases, services provided may be reliant on a variety of third parties. This security review does not constitute endorsement, agreement or acceptance for the Project and technology that was reviewed. Users relying on this security review should not consider this as having any merit for financial advice or technological due diligence in any shape, form or nature.

The veracity and accuracy of the findings presented in this report relate solely to the proficiency, competence, aptitude and discretion of our auditors. UNSNARL and its employees make no guarantees, nor assurance that the contracts are free of exploits, bugs, vulnerabilities, deprecation of technologies or any system / economical / mathematical malfunction.

DISCLAIMER

All the information/opinions/suggestions provided in this report does not constitute financial or investment advice, nor should it be used to signal that any person reading this report should invest their funds without sufficient individual due diligence regardless of the findings presented in this report.

Information in this report is provided 'as is'. UNSNARL is under no covenant to the completeness, accuracy or solidity of the contracts reviewed. UNSNARL's goal is to help reduce the attack vectors/surface and the high level of variance associated with utilizing new and consistently changing technologies.

UNSNARL in no way claims any guarantee, warranty or assurance of security or functionality of the technology that was in scope for this security review.

In no event will UNSNARL, its partners, employees, agents or any parties related to the design/creation of this security review be ever liable to any parties for, or lack thereof, decisions and/or actions with regards to the information provided in this security review.

Cryptocurrencies and all other technologies directly or indirectly related to cryptocurrencies are not standardized, highly prone to malfunction and extremely speculative by nature. No due diligence and/or safeguards may be insufficient and users should exercise maximum caution when participating and/or investing in this nascent industry.

DISCLAIMER

The preparation of this security review has made all reasonable attempts to provide clear and actionable recommendations to the Project team (the “client”) with respect to the rectification, amendment and/or revision of any highlighted issues, vulnerabilities or exploits within the contracts in scope for this engagement.

It is the sole responsibility of the Project team to provide adequate levels of test and perform the necessary checks to ensure that the contracts are functioning as intended, and more specifically to ensure that the functions contained within the contracts in scope have the desired intended effects, functionalities and outcomes, as documented by the Project team.

All services, the security reports, discussions, work product, attack vectors description or any other materials, products or results of this security review engagement is provided "as is" and "as available" and with all faults, uncertainty and defects without warranty or guarantee of any kind.

UNSNARL will assume no liability or responsibility for delays, errors, mistakes, or any inaccuracies of content, suggestions, materials or for any loss, delay, damage of any kind which arose as a result of this engagement/security review.

UNSNARL will assume no liability or responsibility for any personal injury, property damage, of any kind whatsoever that resulted in this engagement and the customer having access to or use of the products, engineers, services, security report, or any other other materials.

For avoidance of doubt, this report, its content, access, and/or usage thereof, including any associated services or materials, shall not be considered or relied upon as any form of financial, investment, tax, legal, regulatory, or any other type of advice.

GLORY OF AUDITORS AT DETECTBOX

DetectBox has created a pool of best auditors across the globe with significant experience in the web3 security industry auditing multiple protocols across multiple chains. Making audits better through **Proof of Talent**.

217+
Projects Audited

$100M+
Amount Secured

2200+
Vulnerabilities Detected

Follow Our Journey on

@unsnarl_secure

UNSNARL

founders@unsnarl.io

www.detectbox.io