SHERLOCK

Security Review For Crestal Network

Public Audit Contest Prepared For:	**Crestal Network**
Lead Security Expert:	**0x73696d616f**
Date Audited:	**March 11 - March 14, 2025**
Final Commit:	**87a48fc**

Introduction

Crestal Network is building a platform for anyone to create a productive AI agent with The Nation, an ecosystem where agents autonomously generate revenue, manage wallets, launch tokens, and access powerful skills to create value both on-chain and off-chain. The set of contracts in scope are the core ”agent creation and update” management contracts. They are upgradeable contracts supporting gasless transactions (erc-4337), plus a few simple functions to accept on-chain ERC20-based payments. The set of contracts in scope are the core ”agent creation and update” management contracts. They are upgradeable contracts supporting gasless transactions (erc-4337), plus a few simple functions to accept on-chain ERC20-based payments.

Scope

Repository: crestalnetwork/crestal-omni-contracts

Audited Commit: dc45e98af5e247dce5bbe53b0bd5b1f256884f84

Final Commit: 87a48fc89ae6ebff658c84ff08a9598050a934fb

Files:

- src/Blueprint.sol
- src/BlueprintCore.sol
- src/BlueprintV5.sol
- src/EIP712.sol
- src/Payment.sol

Final Commit Hash

87a48fc89ae6ebff658c84ff08a9598050a934fb

Findings

Each issue has an assigned severity:

- Medium issues are security vulnerabilities that may not be directly exploitable or may require certain conditions in order to be exploited. All major issues should be addressed.
- High issues are directly exploitable security vulnerabilities that need to be fixed.

Issues Found

High	Medium
1	6

Issues Not Fixed and Not Acknowledged

High	Medium
0	0

Security experts who found valid issues

0x180db
0x23r0
0x41head
0x73696d616f
0xAadi
0xDarko
0xDemon
0xEkko
0xGondar
0xGutzzz
0xYjs
0xadarum
0xaudron
0xc0ffEE
0xgee
0xhiros
0xjarix
0xlucky
0xpetern
Anirruth
Arav
Artur
AuditKiller
Buggx0
CasinoCompiler
Chaindecompiler
ChaosSR
Cybrid
Darinrikusham
DeLaSoul
DharkArtz
Edoscoba
FalseGenius
GODSPEED
HarryBarz
Harry_cryptodev
Harsh
HolyHak
IzuMan
JasonBPMIASN
KiroBrejka
Kodyvim
KungFuPanda
Kwesi
MSK
Matic68
OlaHamid
OpaBatyo
Oxpreacher
Pelz
Pihu
Praise03
Ryonen
Trepid
ZOL
ZafiN
abhishek_thakur
anchabadze
auditism
bube
c3phas
calc1f4r
d33p
dennis
dreamcoder
eLSeR17
edger
farismaulana
gegul
ggbond
gxh191
hirugohan
hrmneffdii
ifeco445
ihtishamsudo
ilyadruzh
j3x
jacopod
jprod15
krot-0025
lom_ack
makeWeb3safe
moray5554
octeezy

oualidpro
patitonar
ph
phrax
princekay
pushkarm029
radcipher
redtrama
resosiloris
roccomania
roshark
sa9933
sabanaku77
seeques
shady4
skid0016
skipper
t0x1c
tachida2k
thimthor
udo
undefined_joe
vivekd
w33kEd
x0rc1ph3r
y4y
yaioxy
zxriptor

Issue H-1: Anyone who is approving `BlueprintV5` contract to spend ERC20 can get drained because `Payment::payWithERC20`

Source:
https://github.com/sherlock-audit/2025-03-crestal-network-judging/issues/260

Found by

0x180db, 0x73696d616f, 0xAadi, 0xDemon, 0xEkko, 0xGondar, 0xGutzzz, 0xYjs, 0xadarum, 0xaudron, 0xc0ffEE, 0xhiros, 0xjarix, 0xlucky, 0xpetern, Anirruth, Arav, Artur, Buggx0, CasinoCompiler, Chaindecompiler, ChaosSR, Cybrid, Edoscoba, FalseGenius, GODSPEED, Harry_cryptodev, Harsh, HolyHak, IzuMan, JasonBPMIASN, Kodyvim, KungFuPanda, Kwesi, MSK, Matic68, OlaHamid, OpaBatyo, Oxpreacher, Pelz, Pihu, Praise03, Ryonen, Trepid, ZOL, ZafiN, abhishek_thakur, anchabadze, auditism, bube, c3phas, dennis, dreamcoder, eLSeR17, farismaulana, gegul, ggbond, gxh191, hirugohan, ihtishamsudo, ilyadruzh, j3x, jacopod, jprod15, krot-0025, lom_ack, makeWeb3safe, patitonar, ph, phrax, princekay, radcipher, redtrama, resosiloris, roccomania, sa9933, sabanaku77, seeques, shady4, skid0016, skipper, tachida2k, thimthor, vivekd, w33kEd, x0rc1ph3r, y4y, yaioxy, zxriptor

Summary

`payWithERC20` is supposed to be used inside `BlueprintV5` contract to handle payment. But this function also can be used to drain anyone who is interact with `BlueprintV5` and using it to approve payment token when creating an agent.

Root Cause

Payment.sol#L25-L32

```
@>    function payWithERC20(address erc20TokenAddress, uint256 amount, address
 ↪  fromAddress, address toAddress) public {
        // check from and to address
        require(fromAddress != toAddress, "Cannot transfer to self address");
        require(toAddress != address(0), "Invalid to address");
        require(amount > 0, "Amount must be greater than 0");
        IERC20 token = IERC20(erc20TokenAddress);
        token.safeTransferFrom(fromAddress, toAddress, amount);
    }
```

the root cause simply because this function is public function, meaning anyone can call this and supply valid token address, then fill `fromAddress` with any address that still have allowance/approving the payment token to be spend by `BlueprintV5` contract

Internal Pre-conditions

1. admin enable usdc or any erc20 token as payment by calling `Blueprint::addPaymentAddress`

External Pre-conditions

1. victim approve the spending of usdc or any erc20 token set in last step for `BlueprintV5` contract address proxy
2. the amount approved should be greater than the amount used for creating agent with token cost
3. victim call the function to create agent (optional)

Attack Path

1. attacker call `payWithERC20` supplying the parameter with usdc address, victim address and sufficient amount to be sent into attacker address

Impact

user/victim who interacted would lose their funds drained by attacker

PoC

No response

Mitigation

make the `Payment::payWithERC20` internal

Discussion

spidemen2024

Got it

sherlock-admin2

The protocol team fixed this issue in the following PRs/commits:
https://github.com/crestalnetwork/crestal-omni-contracts/pull/16

Issue M-1: `createCommonProjectIDAndDeploymentRequest()` hardcodes request id index to 0, leading to lost requests for users

Source:
https://github.com/sherlock-audit/2025-03-crestal-network-judging/issues/205

Found by

0x73696d616f

Summary

`createCommonProjectIDAndDeploymentRequest()` is called by `createAgent()`, in which the user pays fees to create an agent. The index is supposed to protect the user from overwritting a requestId with the same requestId but different serverURL. However, it is hardcoded to 0.

Root Cause

In `BlueprintCore:373`, index is 0.

Internal Pre-conditions

None.

External Pre-conditions

None.

Attack Path

1. User creates an agent for a certain projectId, base64Proposal, server url.
2. User creates an agent (at the same block) with the same projectId, base64Proposal but different server url.
3. First request is overwritten.

Impact

First request is overwritten and one of them will not be finalized as `submitProofOfDeployment()` and `submitDeploymentRequest()` can only be called once as part of the final steps by the worker. However, the user paid fees for both requests, but only one of them will go through.

PoC

See above.

Mitigation

Index should be increment in a user mapping.

Discussion

sherlock-admin2

The protocol team fixed this issue in the following PRs/commits: https://github.com/crestalnetwork/crestal-omni-contracts/pull/28

Issue M-2: Signatures missing some parameters being vulnerable to attackers using them coupled with malicious parameters

Source: https://github.com/sherlock-audit/2025-03-crestal-network-judging/issues/225

Found by

0x73696d616f

Summary

createAgentWithSigWithNFT() for example signs `projectId, base64RecParam, serverURL`. However, it does not sign `privateWorkerAddress` or tokenId. This is an issue because although Base has a private mempool, the protocol integrates with Biconomy, which leverages ERC4337 and has a mempool for bundlers. Hence, the signatures will be available in the mempool and anyone can fetch them and submit it directly to base with other malicious `tokenId` or `privateWorkerAddress`.

Thus, users can be forced to create agents with token ids they didn't intend to use or use invalid worker addresses, DoSing them. Workers have incentive to do this as they can censor other workers this way from using other workers and only they will be able to make the deployments, censoring other workers. The protocol intends to benefit workers from their work, so they have incentive to do so.

If `[createAgentWithTokenWithSig()](https://github.com/sherlock-audit/2025-03-crestal-network/blob/main/crestal-omni-contracts/src/BlueprintCore.sol#L491)`, the token address used can be another one that has a bigger cost and users end up paying more.

Root Cause

In `createAgentWithSigWithNFT()` and similar, `tokenAddress`, `tokenId`, `privateWorkerAddress` are not signed.

Internal Pre-conditions

None.

External Pre-conditions

None.

Attack Path

1. User sends signature to be used on `createAgentWithSigWithNFT()` or `createAgentWithTokenWithSig()` to the offchain protocol, which forwards it to Biconomy, adding the user operation to the mempool.
2. Attacker picks up the signature from the eip4337 mempool and submits the onchain transaction with other malicious inputs.

Impact

Worker censors other workers, DoSes users, makes them pay fees without getting services and ultimately forces users to use the attacker worker's services, who gets illegitimate fees. Or, attacker steals tokens from users by specifying a different token address. Or, another token id ownership is used.

PoC

Here is how the biconomy bundler works (which is the same as the typical bundler):

> Aggregating userOps in an alternative mempool to normal Ethereum Transactions

Attacker can become a bundler and listen to the same mempool and perform the attack.

Mitigation

Sign all parameters.

Discussion

sherlock-admin2

The protocol team fixed this issue in the following PRs/commits:
https://github.com/crestalnetwork/crestal-omni-contracts/pull/18

Issue M-3: Signature Replay attack possible on `updateWorkerDeploymentConfigWithSig()` in Blueprint-core.sol which leads to users lose the funds

Source: https://github.com/sherlock-audit/2025-03-crestal-network-judging/issues/391

Found by

0x73696d616f, 0xYjs, AuditKiller, Cybrid, IzuMan, calc1f4r, d33p, gegul, j3x, moray5554, patitonar, ph, pushkarm029, sabanaku77, seeques, shady4, zxriptor

Summary

The lack of replay protection in the `updateWorkerDeploymentConfigWithSig` function will cause a significant loss of funds for users as a malicious actor will replay a signed transaction to repeatedly transfer funds from the deployment owner to the fee collection wallet. The protocol didn't have the functionality to refund these funds to the respective users if this issue occurs. So anyway user is gonna lose their fund.

Root Cause

In `BlueprintCore.sol` at the updateWorkerDeploymentConfigWithSig function, the function verifies a signature using `getRequestDeploymentDigest` but does not include a nonce, timestamp, or chain ID in the signed message. This allows a valid signature to be reused indefinitely, triggering multiple calls to `updateWorkerDeploymentConfigCommon` and its `payWithERC20` payment logic.

```
function updateWorkerDeploymentConfigWithSig(
    address tokenAddress,
    bytes32 projectId,
    bytes32 requestID,
    string memory updatedBase64Config,
    bytes memory signature
) public {
    bytes32 digest = getRequestDeploymentDigest(projectId, updatedBase64Config,
 ↪  "app.crestal.network");
    address signerAddr = getSignerAddress(digest, signature);
    updateWorkerDeploymentConfigCommon(tokenAddress, signerAddr, projectId,
 ↪  requestID, updatedBase64Config);
}
```

Internal Pre-conditions

1. The `updateWorkerDeploymentConfigWithSig` function remains public and unchanged in the deployed contract.
2. A user (deployment owner) has approved the `BlueprintCore.sol` contract to spend their ERC-20 tokens via `approve` on the token contract.
3. The user has signed a valid message (with `projectId`, `updatedBase64Config`, "app.crestal.network") and submitted it to update a deployment configuration for a `requestID` with `status` not equal to `Init` or `Issued`.
4. The `paymentOpCostMp[tokenAddress][UPDATE_AGENT_OP]` returns a non-zero `cost`.

External Pre-conditions

1. The Base blockchain allows transaction replay if the signature remains valid, which is standard behavior unless mitigated.
2. The ERC-20 token contract at `tokenAddress` supports `safeTransferFrom` and doesn't prevent replay.

Attack Path

1. A user (deployment owner) signs a message to update a deployment configuration (projectId, requestID, updatedBase64Config) and submits it via `updateWorkerDeploymentConfigWithSig`, paying $token to `feeCollectionWalletAddress` via `payWithERC20`.
2. The transaction succeeds, updating the configuration and emitting `UpdateDeploymentConfig`, with the signature recorded on-chain.
3. A malicious actor captures the signature and replays the transaction by calling `updateWorkerDeploymentConfigWithSig` with the same parameters (tokenAddress, projectId, requestID, updatedBase64Config, signature).
4. Each replay re-executes `updateWorkerDeploymentConfigCommon`, transferring another $token from the user to `feeCollectionWalletAddress` (if funds/allowance remain) and resetting status to `Pickup` if it was `GeneratedProof`, repeatable until the user's funds are drained or allowance is revoked.

Impact

The user suffers an approximate loss of $token per replay. If replayed indefinitely (e.g., 20 times), the loss could reach 20x more, potentially draining 100% of approved funds. The attacker gains no direct funds but indirectly benefits `feeCollectionWalletAddress`, incurring only gas costs per replay. The protocol didn't have the functionality to refund this token to the respective users if this issue occurs. So anyway user is gonna lose their fund.

PoC

No response

Mitigation

Add a nonce to the signed message and track it per user:

```
mapping(address => uint256) public userNonces;
function updateWorkerDeploymentConfigWithSig(
    address tokenAddress,
    bytes32 projectId,
    bytes32 requestID,
    string memory updatedBase64Config,
    bytes memory signature
) public {
    bytes32 digest = keccak256(abi.encode(
        keccak256("UpdateDeploymentConfig(bytes32 projectId,string updatedBase64Config,string domain,uint256 nonce)"),
        projectId,
        keccak256(bytes(updatedBase64Config)),
        keccak256(bytes("app.crestal.network")),
        userNonces[msg.sender]
    ));
    address signerAddr = getSignerAddress(digest, signature);
    updateWorkerDeploymentConfigCommon(tokenAddress, signerAddr, projectId, requestID, updatedBase64Config);
    userNonces[signerAddr]++;
}
```

Discussion

sherlock-admin2

The protocol team fixed this issue in the following PRs/commits: https://github.com/crestalnetwork/crestal-omni-contracts/pull/17

Issue M-4: Lack of access control in `setWorkerPublicKey()` in BlueprintCore.sol which results users to lose funds

Source:
https://github.com/sherlock-audit/2025-03-crestal-network-judging/issues/467

Found by

j3x, sabanaku77

Summary

The lack of access control in the setWorkerPublicKey function will cause a significant loss of funds for users as a malicious actor will register a fake Worker public key to intercept and disrupt deployment payments.

Root Cause

In `BlueprintCore.sol` at the setWorkerPublicKey function, the function is declared as `public` without any restrictions, allowing any address to register or update a public key in the `workersPublicKey` mapping and add themselves to the `workerAddressesMp` list.

```
@> function setWorkerPublicKey(bytes calldata publicKey) public {
     if (workersPublicKey[msg.sender].length == 0) {
         workerAddressesMp[WORKER_ADDRESS_KEY].push(msg.sender);
     }
     workersPublicKey[msg.sender] = publicKey;
 }
```

Internal Pre-conditions

1. The `setWorkerPublicKey` function remains public and unchanged in the deployed contract.
2. Users rely on the `workerAddressesMp` list or `getWorkerPublicKey` to select Workers for private deployments (via createProjectIDAndPrivateDeploymentRequest or createAgent).
3. The `createAgentWithToken` function is callable, requiring payment (via payWithERC20) for agent creation linked to a Worker.

External Pre-conditions

1. The Base blockchain (per the README) allows any address to send transactions to the contract, which is standard behavior for public networks.

Attack Path

1. A malicious actor calls `setWorkerPublicKey` with a fake public key, registering their address in `workersPublicKey` and adding it to `workerAddressesMp["worker_address_key"]`.
2. A user queries `getWorkerAddresses` or `getWorkerPublicKey` and selects the malicious actor's address as the privateWorkerAddress for a deployment.
3. The user pays in ERC-20 tokens to create an agent, encrypting `base64Proposal` with the malicious Worker's public key and triggering a deployment request.
4. The malicious actor receives the deployment request (status set to `Pickup`) but does not deploy the agent, either keeping the encrypted data or ignoring the request, causing the deployment to fail.

Impact

- The user suffers the loss of there ERC-20 tokens. The attacker gains no direct funds but may extract value from the encrypted base64Proposal (sensitive data), incurring only gas costs.
- Also, the transfered token will go to `feeCollectionWalletAddress` and protocol didn't have a functionality to refund this token to the respective users if this issue occurs. So anyway user is gonna lose their fund.

PoC

No response

Mitigation

- Whitelist Approach: Add a modifier to limit calls to registered Workers, managed by the owner:

```
mapping(address => bool) public registeredWorkers;
modifier onlyRegisteredWorker() {
    require(registeredWorkers[msg.sender], "Not a registered Worker");
    _;
}
function setWorkerPublicKey(bytes calldata publicKey) public onlyRegisteredWorker {
    if (workersPublicKey[msg.sender].length == 0) {
        workerAddressesMp[WORKER_ADDRESS_KEY].push(msg.sender);
    }
    workersPublicKey[msg.sender] = publicKey;
}
function registerWorker(address worker) public onlyOwner {
    registeredWorkers[worker] = true;
}
```

Discussion

sherlock-admin2

The protocol team fixed this issue in the following PRs/commits: https://github.com/crestalnetwork/crestal-omni-contracts/pull/29

Issue M-5: Worker-Induced Denial-of-Service in Deployment Requests Due to Lack of a Cancellation Mechanism

Source:
https://github.com/sherlock-audit/2025-03-crestal-network-judging/issues/509

Found by

0x180db, 0x23r0, 0x73696d616f, 0xDarko, 0xYjs, 0xhiros, AuditKiller, Cybrid, DeLaSoul, DharkArtz, FalseGenius, HarryBarz, HolyHak, KiroBrejka, MSK, anchabadze, edger, gegul, ifeco445, ilyadruzh, j3x, jacopod, lom_ack, octeezy, patitonar, pushkarm029, roshark, sabanaku77, t0x1c, udo, undefined_joe, zxriptor

The BlueprintCore contract enforces a single deployment request per project by using a check in the deploymentRequest function:

```
require(projects[projectId].requestDeploymentID == 0, "deployment requestID already exists");
```

Once a worker picks up the deployment request via the `submitDeploymentRequest` function, the contract sets the request status to `Pickup` and assigns the worker's address:

```
require(requestDeploymentStatus[requestID].status != Status.Pickup, "requestID already picked by another worker");
requestDeploymentStatus[requestID].status = Status.Pickup;
requestDeploymentStatus[requestID].deployWorkerAddr = msg.sender;
```

There is no mechanism to cancel or reset the request if the assigned worker fails to submit the deployment proof through `submitProofOfDeployment`, leaving the request in an indefinite `Pickup` state. Consequently, the project's deployment process becomes permanently stalled, as further deployment requests cannot be initiated because the project's `requestDeploymentID` remains set.

Primary Root Cause:
The root cause is the contract's design, which permits only one active deployment request per project and lacks a timeout or cancellation function to reset a stalled request when the assigned worker does not complete the process.

Impact:
The project owner cannot progress the deployment, effectively halting the project lifecycle. Funds or NFT-based agent creation fees become unusable as the deployment never completes.

Mitigation:

Implement a timeout mechanism that allows the deployment owner to cancel and reset a stalled deployment request if no proof is submitted within a defined period (e.g., 7 days).

Discussion

sherlock-admin2

The protocol team fixed this issue in the following PRs/commits:
https://github.com/crestalnetwork/crestal-omni-contracts/pull/19

Issue M-6: Non whitelisted user can also create agent by calling createAgentWithNFT instead of createAgentWithWhitelistUsers affecting the motive of protocol to only allow whitelisted user to create agent

Source:
https://github.com/sherlock-audit/2025-03-crestal-network-judging/issues/576

Found by

0x41head, 0xgee, Darinrikusham, Ryonen, dreamcoder, eLSeR17, hrmneffdii, jprod15, oualidpro, patitonar

Summary

Non whitelisted user can also create agent by calling createAgentWithNFT instead of createAgentWithWhitelistUsers affecting the motive of protocol to only allow whitelisted user to create agent

Root Cause

createAgentWithWhitelistUsers function is designed by protocol with motive to only allow a particular amount of whitelisted users to create agent but this motive can be bypassed by anyone by calling createAgentWithNFT function instead.

Internal Pre-conditions

NA

External Pre-conditions

NA

Attack Path

1. Non whitelisted user can call `createAgentWithNFT` function instead of `createAgentWithWhitelistUsers` function and can create agent breaking the whitelist check

Impact

Non whitelisted user can also create agent breaking the motive of protocol to only allow whitelisted users to create agent.

PoC

No response

Mitigation

- Implement pausability feature in createAgentWithNFT function so that admin can pause the access of it until whitelist period for creation of agent and later can enable it.

Discussion

sherlock-admin2

The protocol team fixed this issue in the following PRs/commits:
https://github.com/crestalnetwork/crestal-omni-contracts/pull/24/files

Disclaimers

Sherlock does not provide guarantees nor warranties relating to the security of the project.

Usage of all smart contract software is at the respective users’ sole risk and is the users’ responsibility.