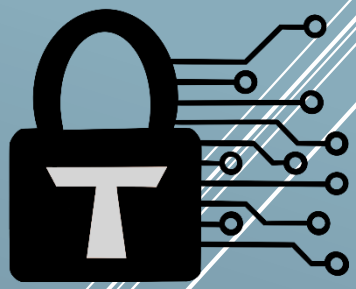
Trust Security

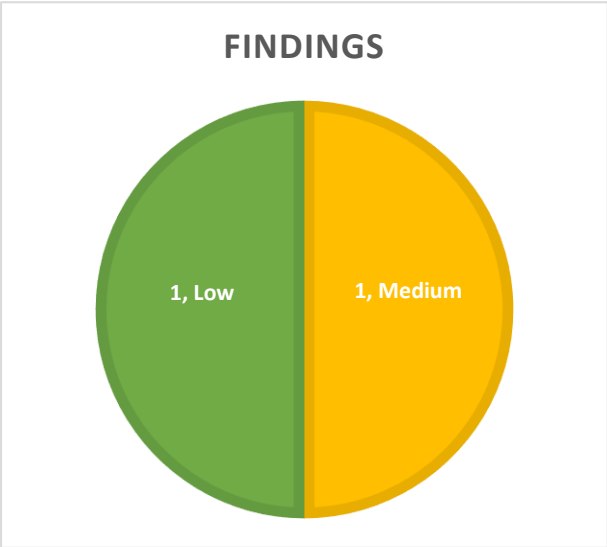


Smart Contract Audit

Abstract Clave Contracts

21/11/2024

Executive summary



Category	Smart Wallet
Audited file count	30
Auditor	HollaDieWaldfee
Time period	05/11/2024 - 11/11/2024

Findings

Severity	Total	Open	Fixed	Acknowledged
High	0	0	0	0
Medium	1	0	1	0
Low	1	0	0	1

Signature

Digitally signed by Trust
Date: 2024.11.21 11:30:41 +02'00'

Document properties

Versioning

Version	Date	Description
0.1	11/11/2024	Client report
0.2	21/11/2024	Mitigation review

Contact

Trust

trust@trust-security.xyz

Introduction

Trust Security has conducted an audit at the customer's request. The audit is focused on uncovering security issues and additional bugs contained in the code defined in scope. Some additional recommendations have also been given when appropriate.

Scope

The codebase is a fork from getclave/clave-contracts.

All changes in files not related to testing are in scope. Some of the below files have simply been removed.

- contracts/AccountFactory.sol
- contracts/ClaveImplementation.sol
- contracts/ClaveProxy.sol
- contracts/ClaveRegistry.sol
- contracts/cns/ClaveNameService.sol
- contracts/cns/IClaveNameService.sol
- contracts/earn/ClaveEarnRouter.sol
- contracts/earn/KoiEarnRouter.sol
- contracts/earn/SyncEarnRouter.sol
- contracts/earn/SyncEarnRouterV2.sol
- contracts/earn/ZtaKe.sol
- contracts/earn/ZtaKeV2.sol
- contracts/handlers/ERC1271Handler.sol
- contracts/helpers/Base64.sol
- contracts/helpers/EIP712.sol
- contracts/interfaces/IClave.sol
- contracts/interfaces/IUpgradeManager.sol
- contracts/libraries/Errors.sol
- contracts/managers/OwnerManager.sol
- contracts/managers/UpgradeManager.sol
- contracts/managers/ValidatorManager.sol
- contracts/modules/recovery/CloudRecoveryModule.sol
- contracts/modules/recovery/SocialRecoveryModule.sol
- contracts/modules/recovery/base/BaseRecovery.sol
- contracts/paymasters/ERC20Paymaster.sol
- contracts/paymasters/GaslessPaymaster.sol
- contracts/validators/EOAValidator.sol
- contracts/validators/PasskeyValidator.sol
- deploy/deploy.ts
- deploy/utils.ts

Repository details

- **Repository URL:** https://github.com/Abstract-Foundation/clave-contracts
- **Commit hash:** b2c88fc704b41430a5225f60416e86161628c178
- **Forked repository:** https://github.com/getclave/clave-contracts/tree/aaeb6ed7272febeef3aa6d3437433a725ef5fad1
- **Mitigation hash:** bbc9b2e09c4dc213681c99534246b9d43545bd67

About Trust Security

Trust Security has been established by top-end blockchain security researcher Trust, in order to provide high quality auditing services. Trust is the leading auditor at competitive auditing service Code4rena, reported several critical issues to Immunefi bug bounty platform and is currently a Code4rena judge.

About the Auditors

HollaDieWaldfee is a renowned security expert with a track record of multiple first places in competitive audits. He is a Lead Auditor for Trust Security and Renascence Labs and Lead Senior Watson at Sherlock.

Disclaimer

Smart contracts are an experimental technology with many known and unknown risks. Trust Security assumes no responsibility for any misbehavior, bugs or exploits affecting the audited code or any part of the deployment phase.

Furthermore, it is known to all parties that changes to the audited code, including fixes of issues highlighted in this report, may introduce new issues and require further auditing.

Methodology

In general, the primary methodology used is manual auditing. The entire in-scope code has been deeply looked at and considered from different adversarial perspectives. Any additional dependencies on external code have also been reviewed.

Findings

Medium severity findings

TRST-M-1: Permissionless deployment of accounts is vulnerable to front-running

- **Category:** Front-running issues
- **Source:** AccountFactory.sol, ClaveImplementation.sol
- **Status:** Fixed

Description

New accounts are deployed with *AccountFactory.deployAccount()*. Users can now deploy accounts in a permissionless manner without relying on the **deployer**. Accounts are deployed with **create2** address derivation, and any user can create an account with any **salt**, there is no address separation between users.

In *ClaveImplementation.initialize()*, it is checked that the **initialK1Owner** that is supplied is equal to the **msg.sender** that has called *AccountFactory.deployAccount()*, only the **deployer** is able to supply any **initialK1Owner**.

```
AccountFactory factory = AccountFactory(msg.sender);
address thisDeployer = factory.accountToDeployer(address(this));
if (thisDeployer != factory.deployer()) {
    if (initialK1Owner != thisDeployer) {
        revert Errors.NOT_FROM_DEPLOYER();
    }
}
```

Furthermore, it is important to understand that there exists the *AccountFactory.claveAccountCreated()* function, which allows the **deployer** to commit to an account address for a user even if the account hasn't been deployed yet.

```
function claveAccountCreated(address accountAddress) external {
    if (msg.sender != deployer) {
        revert Errors.NOT_FROM_DEPLOYER();
    }
    emit ClaveAccountCreated(accountAddress);
}
```

Making account creation permissionless in this way is unsafe because an attacker can supply any **salt** and thus create any account, setting itself as the **initialK1Owner**. If the **deployer** has committed to an account address for a user, and the transaction details have been witnessed by an attacker, the attacker can front-run the account creation by supplying the same **salt**.

The intended functionality is that each user address is mapped to an account address and the account can only be created by the user to which the account belongs or the **deployer**.

Recommended mitigation

To support the intended functionality, it must be checked that each user can only create an account with the **salt** value that corresponds to their own account which is defined as **keccak256(abi.encodePacked(userAddress))**.

ClientImplementation.initialize() must check that **address(this)**, i.e., the account, has been generated with the correct salt.

```
@@ -75,6 +75,9 @@ contract ClaveImplementation is
             }
         }

+        address expectedAddress =
factory.getAddressForSalt(keccak256(abi.encodePacked(initialK1Owner)));
+        require(expectedAddress == address(this));
+
         _k1AddOwner(initialK1Owner);
         _k1AddValidator(initialK1Validator);
```

Furthermore, it must be verified that the **initializer** bytes that are passed to *AccountFactory.deployAccount()* start with the function selector of the initializer function of the current account implementation. The **deployer** can maintain a mapping for this that maps the account implementation address to the allowed function selector to support different account implementations. The check is needed such that the calls to *ClaveImplementation.initialize()* can't be avoided which would allow an attacker to bypass the restrictions and still deploy an account to the address that belongs to a victim.

Note also that the deployment script needs to be adjusted to calculate the correct salt, as the **initialK1Owner** is currently padded and not hashed.

Team response

Fixed in commit 9b661e3.

Mitigation review

The mitigation ensures that whenever an account is created in *deployAccount()*, the initializer function of the account implementation is called successfully. Secondly, the account must be created with **initialK1Owner** as the salt and only **initialK1Owner** itself or the trusted deployer is allowed to perform the deployment.

With all changes combined, a user can't be front run anymore. Each user is assigned a unique account address and only the user or the deployer can deploy it.

The deployment script in *deploy.ts* still needs to be adjusted to support the new salt.

Low severity findings

TRST-L-1: Owner and validator requirements break invariant and can lead to loss of account access

- **Category:** Logical issues
- **Source:** OwnerManager.sol, ValidatorManager.sol
- **Status:** Acknowledged

Description

The codebase from which the protocol has been forked maintains the invariant that each account has at least one r1 validator and at least one r1 owner.

However, this invariant no longer holds, and instead it is only required that there is at least one r1 or k1 validator and at least one r1 or k1 owner.

Previously, the invariant has ensured that there is always a r1 owner and r1 validator which are needed to execute transactions from the account. With the new checks, it is possible to have an r1 owner and k1 validator or k1 owner and r1 validator which together do not allow a transaction to get validated.

A second more severe consequence of the broken invariant lies in the fact that account recoveries downstream call *OwnerManager.resetOwners()*.

```
function resetOwners(bytes calldata pubKey) external override onlySelfOrModule {
    _r1ClearOwners();
    _k1ClearOwners();

    emit ResetOwners();

    _r1AddOwner(pubKey);
}
```

Recoveries remove all r1 and k1 owners and add one r1 owner. Previously, there would be one r1 validator available, thus ensuring that access to the account is maintained. But now, the validator can be a k1 validator which is not compatible with the r1 owner.

Recommended mitigation

Ultimately, users are responsible for configuring their accounts. However, the broken invariant and the new checks are less safe, and in fact can lead to a loss of access if a user relies on them.

A possible direction to mitigate this issue is to establish the invariant that there is always at least one k1 owner and at least one k1 validator. This solution would require a refactoring of the recovery flows.

Another solution is to establish the invariant that there must always be at least one k1 validator and at least one r1 validator, thus allowing for seamless transition of different owners as there is always a compatible validator available.

Before further changes are made, the side effects must be carefully considered. Both directions are determined safe based on the limited understanding that has been acquired during this audit.

Team response

“Established the invariant that at least one k1 signer and k1 owner are present in commit fed2e77.

In addition, it's acknowledged that various social recovery modules/reset owners won't work if there are no active R1 validators. At this point we aren't planning to enable any of that functionality at launch.”

Mitigation review

The new invariant that there is always at least one k1 validator and at least one k1 owner has been established correctly and the security impact with regards to incompatible recovery modules has been acknowledged.

Additional recommendations

TRST-R-1: *deployContract()* takes deployment type parameter but salt defaults to zero

In the *deployContract()* function, which is defined in *deploy/utils.ts*, the **deploymentType** parameter has been added.

```
export const deployContract = async (
    hre: HardhatRuntimeEnvironment,
    contractArtifactName: string,
    constructorArguments?: Array<unknown>,
    options?: DeployContractOptions,
    deploymentType?: DeploymentType,

): Promise<ethers.Contract> => {
```

The **deploymentType** can be any of the following four values:

```
export type DeploymentType =
  | 'create'
  | 'createAccount'
  | 'create2'
  | 'create2Account';
```

In *deploy/deploy.ts*, the *deployContract()* function is used with the **create** and **create2** deployment types. If the deployment type is **create2**, the downstream *deploy()* function in hardhat-zksync allows for a **salt** parameter to be passed, but *deployContract()* does not accept such a parameter. As a result, **salt** defaults to zero bytes.

It is recommended to extend **DeployContractOptions** with a **salt** parameter and to pass it through to *deploy()* in case it is specified.

TRST-R-2: Verification of *ClaveProxy* uses wrong constructor parameter

In *utils/deploy.ts*, the account that is created is verified.

```
await verifyContract(hre, {
    address: accountAddress,
    contract: "contracts/ClaveProxy.sol:ClaveProxy",
    constructorArguments: zeroPadValue(accountAddress, 32),
    bytecode: accountProxyArtifact.bytecode
})
```

Providing **accountAddress** for the constructor arguments is wrong since *ClaveProxy* is deployed with the *ClaveImplementation* address as the constructor argument.

Centralization risks

The changes from the forked codebase have not introduced any new centralization risks.

Systemic risks

The changes from the forked codebase have not introduced any new systemic risks.