Audit Report

Hybrid Custody Smart Contracts

v1.0

July 03, 2023

Table of Contents

Table of Contents **2**
License **3**
Disclaimer **3**
Introduction **5**
Purpose of This Report 5
Codebase Submitted for the Audit 5
Methodology 6
Functionality Overview 6
How to Read This Report **7**
Code Quality Criteria 8
Summary of Findings **9**
Detailed Findings **10**
1. getAllPrivate function incorrectly returns public capabilities 10
2. Deny list filter allows retrieving invalid capabilities 10
3. Capabilities are not checked to be valid 11
4. Replaying publishToParent causes the ProxyAccount resource to be overwritten 11
5. Potential incorrect owner query before ownership acceptance 12
6. Removing nonexistent capabilities emits events 12
7. Transferring ownership does not emit an AccountUpdated event 12
8. Unlinking the public proxy account resource path is unnecessary 13
9. Named parameters are not used for known functionalities 13
10. Duplicate function can be removed 13
11. Default manager capability filter cannot be updated 14
12. Codebase readability can be improved 14
13. addFactory function overwrites existing types 15
14. Outstanding TODO comments in the codebase 15
Appendix A: Test Cases **16**
1. Test case for “Deny list filter allows retrieving invalid capabilities” 16

License

Disclaimer

This audit has been performed by

Oak Security

https://oaksecurity.io/
info@oaksecurity.io

Introduction

Purpose of This Report

Oak Security has been engaged by Dapper Labs Inc. to perform a security audit of Hybrid Custody Smart Contracts.

The objectives of the audit are as follows:

1. Determine the correct functioning of the protocol, in accordance with the project specification.
2. Determine possible vulnerabilities, which could be exploited by an attacker.
3. Determine smart contract bugs, which might lead to unexpected behavior.
4. Analyze whether best practices have been applied during development.
5. Make recommendations to improve code safety and readability.

This report represents a summary of the findings.

As with any code audit, there is a limit to which vulnerabilities can be found, and unexpected execution paths may still be possible. The author of this report does not guarantee complete coverage (see disclaimer).

Codebase Submitted for the Audit

The audit has been performed on the following target:

Repository	https://github.com/onflow/hybrid-custody
Commit	`07143557bb22649b67d696b74b1a800a1800c283`
Scope	• `contracts/factories/*` • `contracts/CapabilityFactory.cdc` • `contracts/CapabilityFilter.cdc` • `contracts/CapabilityProxy.cdc` • `contracts/HybridCustody.cdc`

Methodology

The audit has been performed in the following steps:

1. Gaining an understanding of the code base's intended purpose by reading the available documentation.
2. Automated source code and dependency analysis.
3. Manual line-by-line analysis of the source code for security vulnerabilities and use of best practice guidelines, including but not limited to:
    a. Race condition analysis
    b. Under-/overflow issues
    c. Key management vulnerabilities
4. Report preparation

Functionality Overview

The Hybrid Custody model allows developers to facilitate a user experience beyond a single authenticated account, involving multiple linked "parent" and "child" accounts. This setup allows users to interact with decentralized applications without needing a pre-configured wallet, and it abstracts the complexity of managing assets across multiple accounts.

How to Read This Report

This report classifies the issues found into the following severity categories:

Severity	Description
Critical	A serious and exploitable vulnerability that can lead to loss of funds, unrecoverable locked funds, or catastrophic denial of service.
Major	A vulnerability or bug that can affect the correct functioning of the system, lead to incorrect states or denial of service.
Minor	A violation of common best practices or incorrect usage of primitives, which may not currently have a major impact on security, but may do so in the future or introduce inefficiencies.
Informational	Comments and recommendations of design decisions or potential optimizations, that are not relevant to security. Their application may improve aspects, such as user experience or readability, but is not strictly necessary. This category may also include opinionated recommendations that the project team might not share.

The status of an issue can be one of the following: **Pending,** **Acknowledged**, or **Resolved**.

Note that audits are an important step to improving the security of smart contracts and can find many issues. However, auditing complex codebases has its limits and a remaining risk is present (see disclaimer).

Users of the system should exercise caution. In order to help with the evaluation of the remaining risk, we provide a measure of the following key indicators: **code complexity**, **code readability**, **level of documentation**, and **test coverage**. We include a table with these criteria below.

Note that high complexity or low test coverage does not necessarily equate to a higher risk, although certain bugs are more easily detected in unit testing than in a security audit and vice versa.

Code Quality Criteria

The auditor team assesses the codebase's code quality criteria as follows:

Criteria	Status	Comment
Code complexity	**Medium**	The codebase utilizes complex Account Inboxes and `AuthAccount` capabilities.
Code readability and clarity	**Medium-High**	Most functions are well-documented with clear and concise comments.
Level of documentation	**High**	Detailed documentation is available at https://developers.flow.com/concepts/hybrid-custody
Test coverage	**Medium-High**	`flow-cli` reports 72.5% coverage of statements

Summary of Findings

No	Description	Severity	Status
1	`getAllPrivate` function incorrectly returns public capabilities	Major	Resolved
2	Deny list filter allows retrieving invalid capabilities	Major	Resolved
3	Capabilities are not checked to be valid	Minor	Resolved
4	Replaying `publishToParent` causes the `ProxyAccount` resource to be overwritten	Minor	Resolved
5	Potential incorrect owner query before ownership acceptance	Minor	Resolved
6	Removing nonexistent capabilities emits events	Informational	Resolved
7	Transferring ownership does not emit an `AccountUpdated` event	Informational	Resolved
8	Unlinking the public proxy account resource path is unnecessary	Informational	Acknowledged
9	Named parameters are not used for known functionalities	Informational	Resolved
10	Duplicate function can be removed	Informational	Resolved
11	Default manager capability filter cannot be updated	Informational	Resolved
12	Codebase readability can be improved	Informational	Partially Resolved
13	`addFactory` function overwrites existing types	Informational	Resolved
14	Outstanding TODO comments in the codebase	Informational	Resolved

Detailed Findings

1. `getAllPrivate` function incorrectly returns public capabilities

Severity: Major

In `contracts/CapabilityProxy.cdc:62-64`, the `getAllPrivate` function returns all capabilities stored in the `self.publicCapabilities` dictionary. This is incorrect because the function should return all private capabilities instead of public ones.

Consequently, the function will always return incorrect types of capabilities.

We classify this issue as major because it affects the correct functioning of the system.

Recommendation

We recommend modifying the function to return `self.privateCapabilities.values`.

Status: Resolved

2. Deny list filter allows retrieving invalid capabilities

Severity: Major

In `contracts/CapabilityFilter.cdc:44`, the `allowed` function returns `true` when the capability cannot be borrowed. This is problematic because a malicious parent account can store invalid capabilities and use them once the underlying resource becomes borrowable. Consequently, this allows the parent account to bypass the filter restrictions created by the child account.

Please refer to the appendix to reproduce the issue. While the provided test case demonstrates a situation in which the parent account can bypass the manager capability filter, this issue can similarly lead to bypassing the capability filter in the proxy account.

Recommendation

We recommend returning `false` to prevent the parent from retrieving invalid capabilities.

Status: Resolved

3. Capabilities are not checked to be valid

Severity: Minor

In several instances of the codebase, capabilities are not validated to be borrowable before storing them. The following code locations should have the capability validated:

- `addCapability` function in `contracts/CapabilityProxy.cdc:87-94`.
- Initialization phase in `contracts/HybridCustody.cdc:354` where the `filter` is not `nil`.
- `setManagerCapabilityFilter` function in `contracts/HybridCustody.cdc:418` where the `managerCapabilityFilter` is not `nil`.

Consequently, the capabilities might fail to borrow the underlying resource reference when used, which is inefficient.

Recommendation

We recommend validating the capabilities mentioned above.

Status: Resolved

4. Replaying `publishToParent` causes the `ProxyAccount` resource to be overwritten

Severity: Minor

In `contracts/HybridCustody.cdc:593`, no validation ensures the `publishToParent` function is not called towards the same parent address more than once. If the function was called twice for the same parent address, the old `ProxyAccount` resource will be removed, as seen in line `621`. This is inefficient because the child account should call the `removeParent` function to overwrite an existing parent.

Recommendation

We recommend adding a precondition check that ensures `self.parents[parentAddress] == nil` before publishing to parent.

Status: Resolved

5. Potential incorrect owner query before ownership acceptance

Severity: Minor

In `contracts/HybridCustody.cdc:707`, the `giveOwnership` function sets the `acctOwner` to the recipient to indicate they own this child account. However, there is a possibility that the recipient does not claim the published capability from the child's account. Consequently, the `getOwner` function in line `690` would still show the account owner is the recipient, which is incorrect.

Recommendation

We recommend only setting the account owner to the recipient once they have called the `addOwnedAccount` function.

Status: Resolved

6. Removing nonexistent capabilities emits events

Severity: Informational

In `contracts/CapabilityFilter.cdc:34-37` and lines `72-75`, the `removeType` function removes the capability from the dictionary without checking its existence. This is problematic because the `FilterUpdated` event would be emitted accordingly to indicate the capability is inactive, which is incorrect. After all, the capability was never added before.

Recommendation

We recommend only emitting the event if the capability is found inside the dictionary, similar to the implementation in `contracts/HybridCustody.cdc:310-317`.

Status: Resolved

7. Transferring ownership does not emit an `AccountUpdated` event

Severity: Informational

In `contracts/HybridCustody.cdc:327-333`, the `giveOwnerShip` function calls the child account to transfer ownership to another user. However, the `AccountUpdated` event is not emitted to notify event listeners that there is a change in the owned account.

Recommendation

We recommend emitting the `AccountUpdated` event with `proxy` and `active` attributes as `false`.

Status: Resolved

8. Unlinking the public proxy account resource path is unnecessary

Severity: Informational

In `contracts/HybridCustody.cdc:672`, the `removeParent` function unlinks the public path for the proxy account identifier. This is unnecessary because public paths are not linked during the creation of the proxy account resource.

Recommendation

We recommend removing the unneeded line.

Status: Acknowledged

9. Named parameters are not used for known functionalities

Severity: Informational

In `contracts/HybridCustody.cdc:801`, the `display` metadata view is stored in a dictionary with a hardcoded key "`display`". Since the field and the functionality is already known, hardcoding the parameter can be avoided.

Recommendation

We recommend using dedicated fields for any known variables.

Status: Resolved

10. Duplicate function can be removed

Severity: Informational

In `contracts/HybridCustody.cdc:279`, the `getAddresses` function performs the same action as `getChildAddresses`. This is inefficient because calling both functions returns the same functionality and result.

Recommendation

We recommend removing the unused function.

Status: Resolved

11. Default manager capability filter cannot be updated

Severity: Informational

In `contracts/HybridCustody.cdc:223`, the `filter` variable acts as a default filter value passed to any newly added child account. Since the manager resource owner cannot modify this, any new filter the manager intends to add requires calling the `setManagerCapabilityFilter` again. This can easily get complicated when the number of child accounts increases.

Recommendation

We recommend allowing the manager to modify the default capability filter.

Status: Resolved

12. Codebase readability can be improved

Severity: Informational

The readability of the project can be further improved in the following contexts in `contracts/HybridCustody.cdc`:

1. The variable and function names used to denote the type of account are inconsistent across the contract. Some of them can be useful when referenced within the context it is defined but results in reduced readability in general. Consider explicitly naming the account types and identifiers and keeping them consistent across the contract.

   For example, lines `211` and `363` use both `childAccount` and `account` to specify a child account. In this case, explicitly calling out the child's account can improve readability.

2. The `seal` and `removeOwned` functions do not sound as cautious as they need to be, possibly causing their impact to be undermined. Consider making their importance more explicit in addition to the comments already given in the contract.

Recommendation

We recommend applying the recommendations mentioned above.

Status: Partially Resolved

13. `addFactory` function overwrites existing types

Severity: Informational

In `contracts/CapabilityFactory.cdc:17`, the function `addFactory` doesn't check if the type that is being added already exists or not. If the type to be added already exists, it may be overwritten by mistake.

Recommendation

We recommend implementing a `updateFactory` function which allows updating existing factory types and modifying the `addFactory` function only to accept new types. Alternatively, we recommend documenting the intended behavior of the function to educate developers.

Status: Resolved

14. Outstanding TODO comments in the codebase

Severity: Informational

In several instances of the codebase, many unimplemented functionalities are marked as TODO. This decreases the readability of the codebase.

Recommendation

We recommend implementing the required functionalities or resolving them for better code practices.

Status: Resolved

Appendix A: Test Cases

1. Test case for "Deny list filter allows retrieving invalid capabilities"

To reproduce the issue, please follow along with the setup process.

a. Create `transactions/example-nft/unlink_resource.cdc` and paste the following contents. This transaction simulates the child account removing the resource from the private capability.

```
import "NonFungibleToken"
import "MetadataViews"

import ExampleNFT from "ExampleNFT"

transaction {
    prepare(acct: AuthAccount) {
        let d = ExampleNFT.resolveView(Type<MetadataViews.NFTCollectionData>())!
as! MetadataViews.NFTCollectionData

        let collection_res <- acct.load<@ExampleNFT.Collection>(from:
d.storagePath)
        destroy collection_res
    }
}
```

b. Create `scripts/hybrid-custody/get_nft_provider_capability_without_validate.cdc` and paste the following contents. This script simulates the same behavior as `scripts/hybrid-custody/get_nft_provider_capability.cdc` with the difference of not borrowing the capability.

```
import "HybridCustody"

import "NonFungibleToken"
import "MetadataViews"
import "ExampleNFT"

pub fun main(parent: Address, child: Address) {
    let acct = getAuthAccount(parent)
    let m = acct.borrow<&HybridCustody.Manager>(from:
HybridCustody.ManagerStoragePath)
        ?? panic("manager does not exist")

    let childAcct = m.borrowAccount(addr: child) ?? panic("child account not
found")

    let d = ExampleNFT.resolveView(Type<MetadataViews.NFTCollectionData>())! as!
MetadataViews.NFTCollectionData

    let nakedCap = childAcct.getCapability(path: d.providerPath, type:
Type<&{NonFungibleToken.Provider}>())
        ?? panic("capability not found")

    let cap = nakedCap as! Capability<&{NonFungibleToken.Provider}>

    // cannot borrow because resource underneath is invalid
    // cap.borrow() ?? panic("unable to borrow nft provider capability")
}
```

c. Create `transactions/example-nft/save_capability.cdc` and paste the following contents. This transaction retrieves the capability and saves it to the custom storage path.

```
import "HybridCustody"

import "NonFungibleToken"
import "MetadataViews"
import "ExampleNFT"

transaction(child: Address) {
    prepare(parent: AuthAccount) {
        let acct = parent
        let m = acct.borrow<&HybridCustody.Manager>(from:
HybridCustody.ManagerStoragePath)
        ?? panic("manager does not exist")

        let childAcct = m.borrowAccount(addr: child) ?? panic("child account not
found")

        let d = ExampleNFT.resolveView(Type<MetadataViews.NFTCollectionData>())!
as! MetadataViews.NFTCollectionData

        let nakedCap = childAcct.getCapability(path: d.providerPath, type:
Type<&{NonFungibleToken.Provider}>())
        ?? panic("capability not found")

        let cap = nakedCap as! Capability<&{NonFungibleToken.Provider}>

        // save capability
        acct.save<Capability<&AnyResource{NonFungibleToken.Provider}>>(cap, to:
/storage/stolenCapability)
    }
}
```

d. Create `transactions/example-nft/use_capability.cdc` and paste the following contents. This transaction loads the stored capability and uses it.

```
import "HybridCustody"

import "NonFungibleToken"
import "MetadataViews"
import "ExampleNFT"

transaction {
    prepare(parent: AuthAccount) {
        let acct = parent
        let m = acct.borrow<&HybridCustody.Manager>(from:
HybridCustody.ManagerStoragePath)
        ?? panic("manager does not exist")

        // load capability
        let cap =
acct.load<Capability<&AnyResource{NonFungibleToken.Provider}>>(from:
/storage/stolenCapability)
        ?? panic("Could not load capability from storage")

        cap.borrow()!
    }
}
```

e. Copy and paste the following test case into `test/HybridCustody_tests.cdc`. This test case utilizes the previously created transactions and scripts to demonstrate the issue.

```
pub fun testPotentialAllowedBypass() {
    let child = blockchain.createAccount()
    let parent = blockchain.createAccount()

    setupChildAndParent_FilterKindAll(child: child, parent: parent)

    setupNFTCollection(child)

    scriptExecutor("hybrid-custody/get_nft_provider_capability.cdc",
[parent.address, child.address])

    // create deny filter
    let filter = getTestAccount(FilterKindDenyList)
    setupFilter(filter, FilterKindDenyList)

    // build identifier
    let nftIdentifier = buildTypeIdentifier(getTestAccount(exampleNFT),
exampleNFT, "Collection")

    // add to deny filter
```

```
    addTypeToFilter(filter, FilterKindDenyList, nftIdentifier)

    // set deny filter to child addr
    setManagerFilterOnChild(child: child, parent: parent, filterAddress:
filter.address)

    // expect err
    var error =
expectScriptFailure("hybrid-custody/get_nft_provider_capability.cdc",
[parent.address, child.address])
    assert(contains(error, "Capability is not allowed by this account's
Parent"), message: "failed to find expected error message")

    // remove resource under provider path
    txExecutor("example-nft/unlink_resource.cdc", [child], [], nil, nil)

    // no error, means we can retrieve the capability but cannot borrow it

scriptExecutor("hybrid-custody/get_nft_provider_capability_without_validate.cdc"
, [parent.address, child.address])

    // save capability to custom storage
    txExecutor("example-nft/save_capability.cdc", [parent], [child.address],
nil, nil)

    // link back resource
    setupNFTCollection(child)

    // still error when retreiving
    error =
expectScriptFailure("hybrid-custody/get_nft_provider_capability.cdc",
[parent.address, child.address])
    assert(contains(error, "Capability is not allowed by this account's
Parent"), message: "failed to find expected error message")

    // retrieve capability from storage and use it, effectively bypassing above
restriction
    txExecutor("example-nft/use_capability.cdc", [parent], [], nil, nil)
}
```

f. Run the following command in the terminal to reproduce the issue.

```
flow test --cover test/HybridCustody_tests.cdc
```