CERTIK

Security Assessment

SnowFlake

Jul 20th, 2022

Table of Contents

Summary

This report has been prepared for SnowFlake to discover issues and vulnerabilities in the source code of the SnowFlake project 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.

Overview

Project Summary

Project Name	SnowFlake
Platform	Solana
Language	Rust
Codebase	https://github.com/snowflake-so/snowflake-safe-program
Commit	1d1e5f72c207fa749ac91ecac69d16e3e5747935 4039ee827c6018cf063bbe8e9b3b6795ac6158c1

Audit Summary

Delivery Date	Jul 20, 2022 UTC
Audit Methodology	Static Analysis, Manual Review

Vulnerability Summary

Vulnerability Level	Total	Pending	Declined	Acknowledged	Mitigated	Partially Resolved	Resolved
● Critical	0	0	0	0	0	0	0
● Major	0	0	0	0	0	0	0
● Medium	1	0	0	0	0	0	1
● Minor	6	0	0	0	0	0	6
● Optimization	2	0	0	0	0	0	2
● Informational	8	0	0	5	0	0	3
● Discussion	0	0	0	0	0	0	0

Audit Scope

ID	File	SHA256 Checksum
FEE	snowflake-rust-v2/programs/snowflake/src/common/fee.rs	c3edf9f0cc9635f1ea9ecfbbe35c67fc4b42cec66a357caf47bf606b6a15581e
MOD	snowflake-rust-v2/programs/snowflake/src/common/mod.rs	596a14e1aa7efd6800dd7e67278f07a4e3c85a86728e4d341343bd22b70aeab8
SCH	snowflake-rust-v2/programs/snowflake/src/common/schedule.rs	94bbcd69a5facd5eae774620e115c3ebad90b9dd51e25e7131e366c15a217ee6
ABO	snowflake-rust-v2/programs/snowflake/src/instructions/abort_flow.rs	5852758a15acb19e97c4ba40bbc5a010da3c6f91b4148439093d8003975a0f94
APP	snowflake-rust-v2/programs/snowflake/src/instructions/approve_proposal.rs	f060b9364e9305bec9af18b418c492f0e387d7b4c1c98d9d09177c30cb94421b
CRE	snowflake-rust-v2/programs/snowflake/src/instructions/create_flow.rs	521cfb88845d87eb47c20e550216e4431e67e525d83a0f988b7f80990ae43be7
CRA	snowflake-rust-v2/programs/snowflake/src/instructions/create_safe.rs	17d971ea9441df975bcc26a1a20347f8997177f430d721932d7348e062192217
DEL	snowflake-rust-v2/programs/snowflake/src/instructions/delete_flow.rs	fc4dabab6723c106f16bf25bd58d7c618c338f1bf9abea7679a3555db7d8f625
DOE	snowflake-rust-v2/programs/snowflake/src/instructions/do_execute_multisig_flow.rs	1afd875bf2f75ae639b63348c7d4b95ca5bb9a465ee940b77458738e8ea07185
EXE	snowflake-rust-v2/programs/snowflake/src/instructions/execute_multisig_flow.rs	c236fce99cde0c76ad5d9db969a5fa6664cb9a4837cd8408aaddc3b54ca11b8c
EXC	snowflake-rust-v2/programs/snowflake/src/instructions/execute_scheduled_multisig_flow.rs	bce77f89045302137efd1d89e2ba78ab26bbe8e7f932a0240b99eabee4b927ba
MOI	snowflake-rust-v2/programs/snowflake/src/instructions/mod.rs	24be02a4ddaf86331153da9a539785c4e9d1d416421cc02102b10469427ffba1
UPD	snowflake-rust-v2/programs/snowflake/src/instructions/update_safe.rs	8e1f6d6f80fb70e856edfb4a2158c135be49de085009e486b142006ddf6eb7c3
ACT	snowflake-rust-v2/programs/snowflake/src/state/action.rs	66256ae2f1696a719e024054e84fc3c79d8175a89ababb44c01202ffd560e714

ID	File	SHA256 Checksum
APR	snowflake-rust-v2/programs/snowflake/src/state/approval_record.rs	f1fdf75bfb319df4d923925aef063b0daa3ac04fbe45cab2771a8fa830bdadf7
FLO	snowflake-rust-v2/programs/snowflake/src/state/flow.rs	5a6dba47c02cb5f2595e92207de2f9922850cfc021dd2d3b807d0d5111272bba
MOS	snowflake-rust-v2/programs/snowflake/src/state/mod.rs	5a7b75b439bc05902a4a6d2abdf1ac435e2df76e93e67a9d9fef5e2fe22b005e
SAF	snowflake-rust-v2/programs/snowflake/src/state/safe.rs	eb5548f1a95935bf7299f236209451bf2fdf4ad7184c28725dd7b4d0bd0d02ae
STT	snowflake-rust-v2/programs/snowflake/src/state/static_config.rs	a45ea48a1e15207413858ffab7ebea1c9d8fdb6d46b2af552949c0176aa9516d
TAR	snowflake-rust-v2/programs/snowflake/src/state/target_acount_spec.rs	9ac3e8ac8b7bc0c9c36db092c08775d4dba22a5752ddb202ec75459125c9ac0c
ERR	snowflake-rust-v2/programs/snowflake/src/error.rs	aee1302756d8559dd3deadfa3b8aa426316f75e72859181618541d789177f29c
LIB	snowflake-rust-v2/programs/snowflake/src/lib.rs	5bf12c0f69cccc61d3d2c3bdf4c725918f56ee0e79e102df578ac1e12d3c29f9

Logic Overview

Snowflake Safe is a system designed with the aim of providing support for deferred instruction execution of Solana programs on Sealevel, the Solana runtime. It is based on an approval mechanism for proposals which are modelled using an abstract construct called `Flow`. The instructions to be executed are stored as `Action`s inside the `Flow`.

The execution of such `Action`s (i.e the execution of a `Flow`) is governed by an additional construct called `Safe`. Every `Flow` is assigned to a `Safe`, which governs its lifecycle. A `Safe` has a set of owners, any of whom can initiate the creation of a new `Flow`.

For a `Flow` to be executed, it must be approved by a set of owners from the `Safe` it points to. The number of owners whose approval is required is dictated by the `approvals_required` field of the `Safe` struct.

Finally, after a `Flow` is approved, according to its `TriggerType`, it can be executed once by any of the `Safe` owners or periodically by any account, according to the timing constraints specified in the `Flow` data.

Any Solana account can create a `Safe` with a custom set of owners, and then leverage the Snowflake logic. Once created, any modification to the `Safe` needs to be included in a `Flow` and undergo the approval mechanism of the `Safe` itself.

Implementation Overview

The implementation is based on the Anchor Framework and instructions' code is divided into distinct files, in the `instructions` folder, grouping instructions' logic with the corresponding Anchor Context Accounts deserialization.

`lib.rs` is the program entrypoint.

The program implements the following functionalities:

- Creating a `Safe` : A `Safe` can be created by any Solana account, using the instruction `create_safe`. The public key of this account will be stored in the `creator` field of the `Safe` and must be one of the owners of the `Safe`.
- Updating a `Safe` : A `Safe` can be updated by either adding/removing owners or changing the approval threshold for a `Flow`. Each update operation has its instruction and needs to undergo the `Safe` approval mechanism in order to be signed by the `Safe` itself.
- Creating a `Flow` : A `Safe` owner account can create a `Flow` by invoking the `create_flow` instruction. Any `Safe` owner can create a `Flow` and be its requester (`Flow::requested_by`). Such a proposal will

be `Pending` and will be executed only after it has enough approvals from the `Safe` owners.

- Deleting a `Flow` : A `Flow` can be only deleted by its requester by invoking the `delete_flow` instruction and only if it has never been executed.
- Approving a proposal : A `Safe` owner can approve a proposal by invoking the `approve_proposal` instruction. A proposal is accepted only if the number of votes it receives is at least as big as the number set in the `approvals_required` field of the `Safe` it points to.
- Executing a `Flow` : A `Flow` can be executed by one of the `Safe` owners by invoking the `execute_multisig_flow` instruction. In order for the instruction to be executed successfully, the `Flow` must have reached the approved stage. The call will immediately execute the `Flow`'s `Action`s if its trigger type is `Manual`. Instead, if the trigger type is `Program` or `Time`, the execution is scheduled for the specified time. In this event, the status will be saved as `ExecutionInProgress`. Then, a call to the `execute_scheduled_multisig_flow` instruction can be performed by any account (even the accounts not included in the linked `Safe`) in order to execute the `Flow` when it is due. In this case, the instruction caller is rewarded with a fee equal to the transaction cost.

The lifecycle of a `Flow`, and the effects of the instruction execution on its state are illustrated in the Flow State Diagram paragraph.

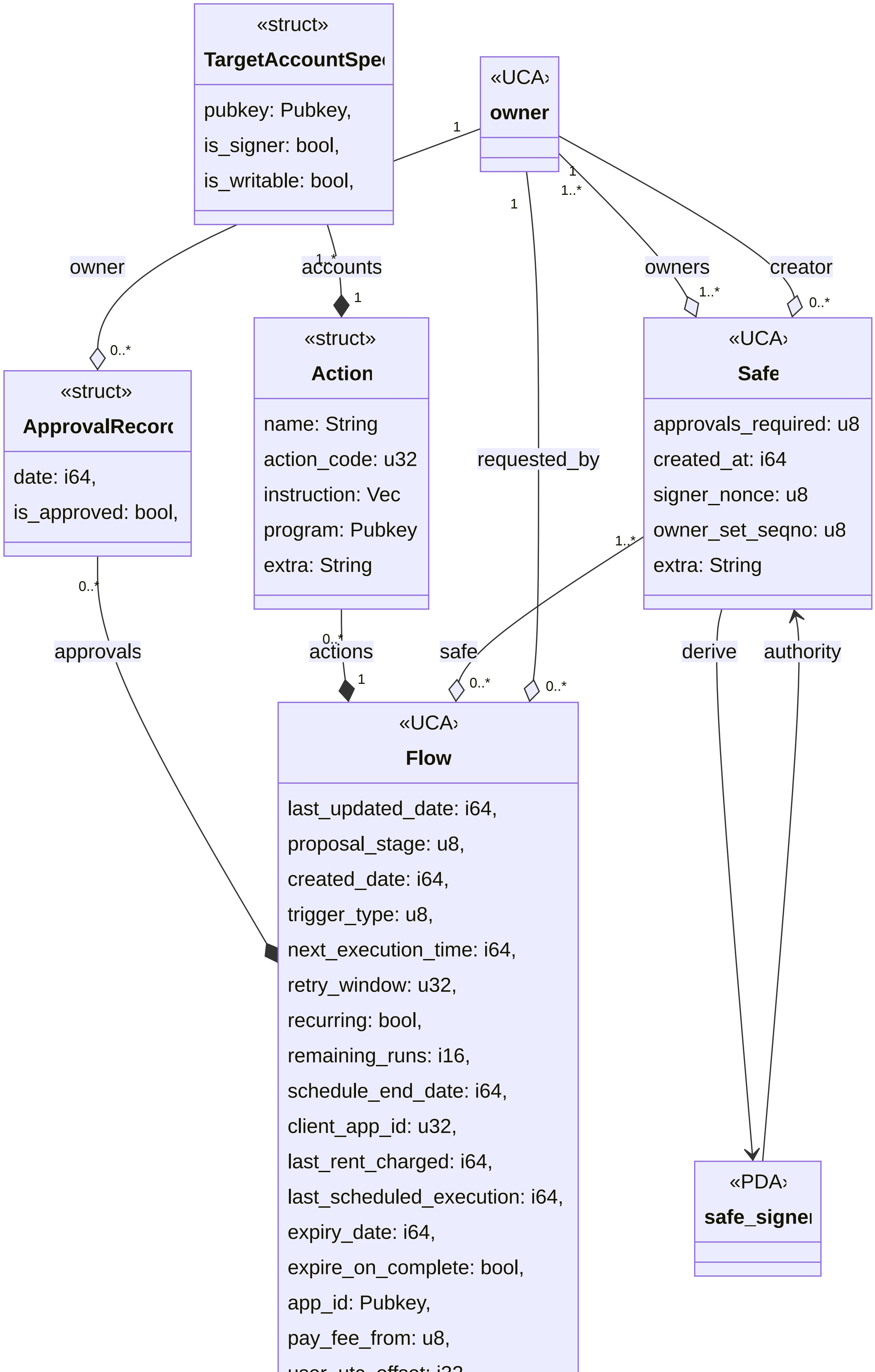
Diagrams

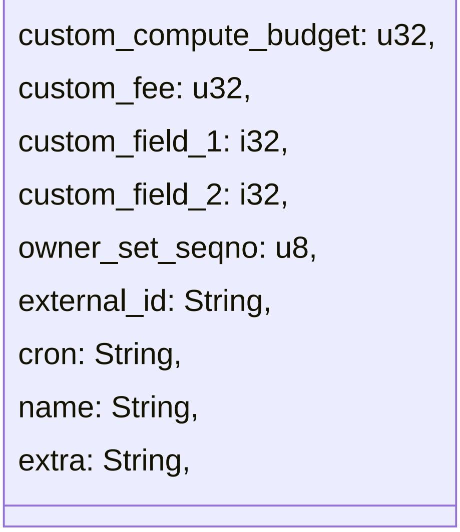
Account Diagram

This graph describes the relationships among the accounts and their attached data.

Legend:

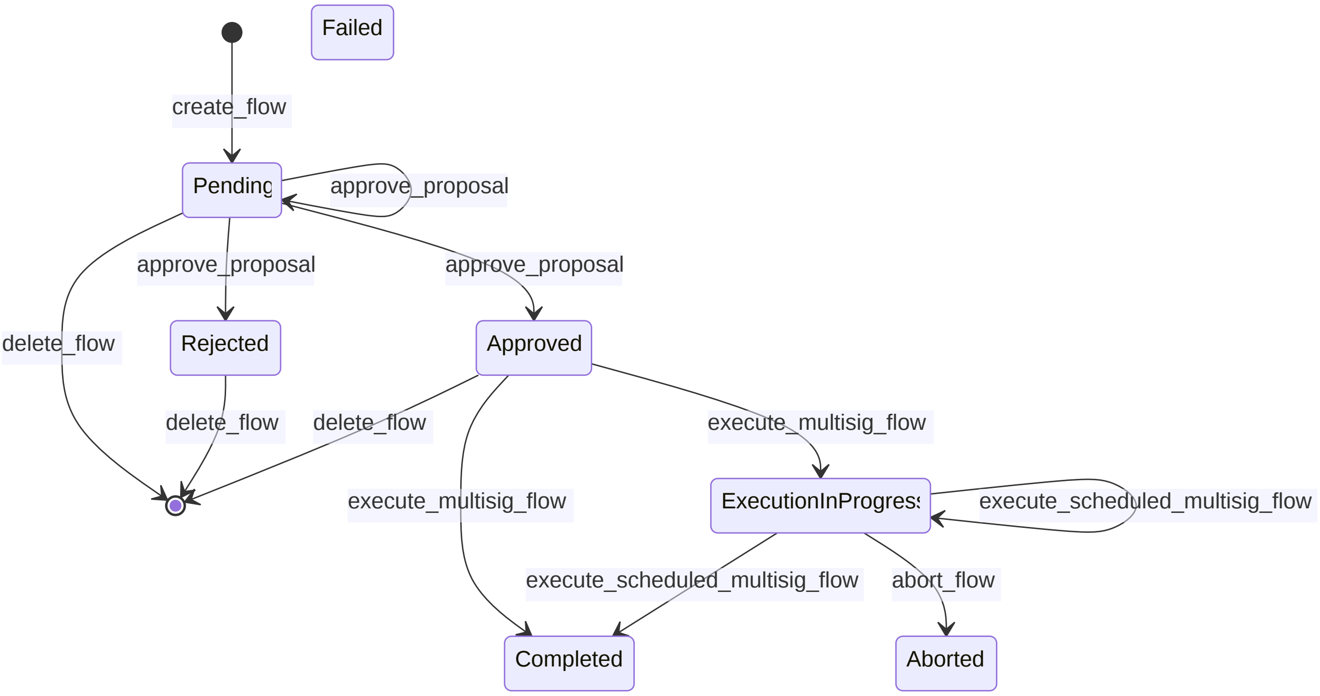
- UCA: User Controlled Address. An account that is not controlled by the audited program, so it can be either a Program Derived Address by a different program or an Address generated by a private key.
- PDA: Program Derived Address as defined in the Solana architecture and documentation. PDA are managed through the audited program logic.
- struct: simple aggregation of semantically homogeneous data that does not own on-chain account.
- `derive` states that the account address from which the arrow goes out is part of the seeds that generate the pointed PDA.
- `authority` states that the account from which the arrow goes out can perform privileged operation on the pointed account, e.g., move token funds, modify information, ...
- composition arrow states that the pointing entity data is saved, lives and dies in the pointed entity storage.
- aggregation arrow states that the pointing account address is saved in the pointed account storage.


«struct»
TargetAccountSpe
pubkey: Pubkey,
is_signer: bool,
is_writable: bool,
«UCA›
owner
1
1
1
1..*
owner
accounts
1..*
1
owners
creator
1..*
0..*
0..*
«struct»
ApprovalRecor
date: i64,
is_approved: bool,
«struct»
Action
name: String
action_code: u32
instruction: Vec
program: Pubkey
extra: String
«UCA›
Safe
approvals_required: u8
created_at: i64
signer_nonce: u8
owner_set_seqno: u8
extra: String
requested_by
1..*
0..*
approvals
0..*
actions
1
safe
0..*
0..*
derive
authority
«UCA›
Flow
last_updated_date: i64,
proposal_stage: u8,
created_date: i64,
trigger_type: u8,
next_execution_time: i64,
retry_window: u32,
recurring: bool,
remaining_runs: i16,
schedule_end_date: i64,
client_app_id: u32,
last_rent_charged: i64,
last_scheduled_execution: i64,
expiry_date: i64,
expire_on_complete: bool,
app_id: Pubkey,
pay_fee_from: u8,
user_utc_offset: i32
«PDA›
safe_signe

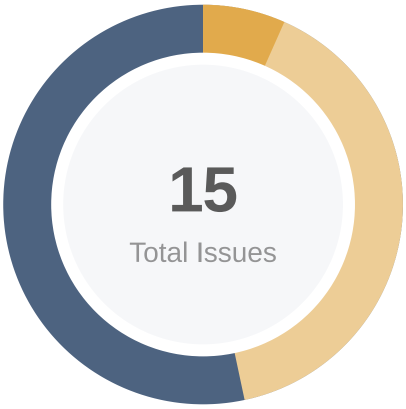


Flow State Diagram

The following graph describes the states, and their transitions. The entire diagram is defined in terms of the `Flow` account, its approval and execution process.



Findings



🟥 **Critical**	**0** (0.00%)
🟧 **Major**	**0** (0.00%)
🟨 **Medium**	**1** (6.67%)
🟨 **Minor**	**6** (40.00%)
🟦 **Informational**	**8** (53.33%)
🟩 **Discussion**	**0** (0.00%)

ID	Title	Category	Severity	Status
ABO-01	Circumvented `Safe` Approach On `abort_flow`	Logical Issue	● Informational	✓ Resolved
APP-01	Panic Instead Of Error When Approving Already Approved `Flow`	Coding Style	● Informational	ⓘ Acknowledged
CRA-01	Unverified Program Derived Address	Data Flow	● Minor	✓ Resolved
CRA-02	Clarification On `Safe::creator` Field	Logical Issue	● Informational	ⓘ Acknowledged
CRE-01	Unused Parameter `client_flow`	Coding Style	● Informational	✓ Resolved
EXE-01	`Safe` Changes Can Invalidate Approved `Flow` S	Logical Issue	● Minor	✓ Resolved
FEE-01	Missing Check On `Flow` Rent-exempt Condition	Data Flow	● Medium	✓ Resolved
FLO-01	Missing Input Validation For `Flow` Fields	Logical Issue	● Minor	✓ Resolved
FLO-02	Inconsistency On `Flow::recurring` When `RECURRING_FOREVER` Is Set	Data Flow	● Minor	✓ Resolved
FLO-03	Clarification On `Flow::schedule_end_date`	Data Flow	● Informational	ⓘ Acknowledged
INS-01	Usage Of Magic Numbers	Coding Style	● Informational	ⓘ Acknowledged

ID	Title	Category	Severity	Status
SAF-01	Incorrect Space Reservation For `Safe::extra`	Logical Issue	● Minor	✓ Resolved
SAF-03	Misleading `signer_nonce` Field Name	Coding Style	● Informational	✓ Resolved
STA-01	Unused `Flow` And `Safe` Fields	Data Flow	● Informational	ⓘ Acknowledged
STT-01	Unused `ProposalStateType::Failed` State	Logical Issue	● Minor	✓ Resolved

ABO-01 | Circumvented `Safe` Approach On `abort_flow`

Category	Severity	Location	Status
Logical Issue	● Informational	snowflake-rust-v2/programs/snowflake/src/instructions/abort_flow.rs: 7~33	✓ Resolved

Description

The `abort_flow` operation can be executed at any time by any `owner` included in the assigned `Safe` when the `Flow` is in the `ExecutionInProgress` state. Once in the `Aborted` state, despite of any other `Flow` condition (e.g. remaining runs, scheduled execution, ...), no more operations can be performed on the `Flow`.

Given that any `Safe` member can execute the `abort_flow` instruction, any compromise, or failure, regarding any `owner` constituting the `Safe` is a direct compromise of the `Flow`'s execution schedule.

Recommendation

According to the wanted business logic, one, or a combination including but not limited to, of the following remediation can mitigate the mentioned risk:

- restrict the functionality to the `Flow` requester (`Flow::requested_by`);
- constrain the functionality to the `Safe` threshold approval process;
- remove the functionality.

Alleviation

`[Snowflake]`: Thanks for your comment, this is actually intentional & by-design.

Our view is that it is more secure to allow any owners to be able to abort a flow. Imagine one of the owner sets up something recurring and went on a long vacation. We want the team to be able to co-manage that job efficiently.

Therefore, we intentionally designed the system such that any of the owners can stop the recurring flow (in case the team finds an issue with the flow, and needs to act quickly). Later, the team can always create a new proposal to replace the existing one.

APP-01 | Panic Instead Of Error When Approving Already Approved `Flow`

Category	Severity	Location	Status
Coding Style	● Informational	snowflake-rust-v2/programs/snowflake/src/instructions/approve_proposal.rs: 48	ⓘ Acknowledged

Description

The `approve_proposal` instruction does not manage the error condition in which a `Safe` owner approves a `Flow` that already reached the `approvals_required` threshold.

Example scenario is: `Safe` with 5 owners, and 3 as `approvals_required`. The condition to manage is a 4th owner approving the `Flow` after that 3 already did the same, so `Flow::proposal_stage` was already in `ProposalStateType::Approved`.

Behavior is not harmful since the check at the pointed line reverts the instruction through a `panic` but no specific error code is returned to the caller in order to manage the condition (e.g. prompt an error description, ...)

Recommendation

Provide a custom error code for the described condition.

CRA-01 | Unverified Program Derived Address

Category	Severity	Location	Status
Data Flow	● Minor	snowflake-rust-v2/programs/snowflake/src/instructions/create_safe.rs: 48	✓ Resolved

Description

Program Derived Addresses have the intrinsic property to not rely on the ed25519 curve, that implies the certainty that only the calling program can sign on its behalf.

Each `Safe` is assigned a `safe_signer` PDA generated using the `Safe` address and the `Safe::signer_nonce` bump as seeds. Anyway, no check is performed on the fact that the provided bump `Safe::signer_nonce` let `safe_signer` be a valid PDA (i.e. an address not corresponding to any public key).

Recommendation

We recommend to verify that the provided bump `client_safe.signer_nonce` in L~48, paired with the other seed used for the `safe_signer`, generates a valid PDA.

Solana CPI Pubkey::create_program_address

Alleviation

`[CertiK]`: The team heeded the advice and resolved the finding in the commit 4039ee827c6018cf063bbe8e9b3b6795ac6158c1

CRA-02 | Clarification On `Safe::creator` Field

Category	Severity	Location	Status
Logical Issue	● Informational	snowflake-rust-v2/programs/snowflake/src/instructions/create_safe.rs: 43~46	ⓘ Acknowledged

Description

At the `Safe` creation, the `Safe::creator` is required to be one of the `Safe::owners`. Such a check is absent in the `remove_owner_handler`, so the `Safe::creator` can be removed from the `Safe::owners`, voiding the condition enforced in the `Safe` creation.

Recommendation

We want to highlight this behavior in order to check if it is the expected one.

Alleviation

`[SnowFlake]`: At the time the safe is created, it makes sense to have the safe creator being one of the owners.

However, in the future, the creator can be removed from the owner list. This is for the situation where the creator leaves the team (and is no longer an owner). And the team does not have to create a new safe because of that.

CRE-01 | Unused Parameter `client_flow`

Category	Severity	Location	Status
Coding Style	● Informational	snowflake-rust-v2/programs/snowflake/src/instructions/create_flow.rs: 7	✓ Resolved

Description

The instruction parameter `client_flow` is referred in the account definition macro but it is not used.

Recommendation

Remove the unnecessary parameter reference from the macro.

Alleviation

`[CertiK]`: The team heeded the advice and resolved the finding in the commit 4039ee827c6018cf063bbe8e9b3b6795ac6158c1

EXE-01 | `Safe` Changes Can Invalidate Approved `Flow` S

Category	Severity	Location	Status
Logical Issue	● Minor	snowflake-rust-v2/programs/snowflake/src/instructions/execute_multisig_flow.rs: 39~43	✓ Resolved

Description

The `Flow::owner_set_seqno` field points to the incremental counter `Safe::owner_set_seqno` representing the configuration that was active at `Flow` creation time. This prevents changes in the `Safe` signing policy (in the `owners` set or on the `approvals_required`) to influence previously created `Flow`s.

Assumed this general logic, the check at the pointed location can let an approved `Flow` to be not executed anymore. In fact if `Safe::approvals_required` gets incremented before the call to `execute_multisig_flow` instruction, but after the `Flow` reached the `Approved` state, then the pointed check can returns false (if the new threshold is greater than the collected approvals) but new approvals (e.g. from `Safe` owner who did not vote, yet) can not be submitted because of the `owner_set_seqno` control.

Recommendation

According to the business logic needs, one or a combination of, including but not limited to, the following remediations can be taken:

- remove the pointed check, in the case in which incremented `Safe` thresholds should not impact previously approved `Flow`s;
- provide the functionality, for the `Flow` requester, to accept the new `Safe` configuration and update it in the `Flow::owner_set_seqno` field.

Alleviation

`[CertiK]`: The team heeded the advice and resolved the finding in the commit 4039ee827c6018cf063bbe8e9b3b6795ac6158c1

FEE-01 | Missing Check On `Flow` Rent-exempt Condition

Category	Severity	Location	Status
Data Flow	● Medium	snowflake-rust-v2/programs/snowflake/src/common/fee.rs: 17~28	✓ Resolved

Description

When `Flow::pay_fee_from` is set as `FeeSource::FromFlow`, the operator executing the scheduled `Flow` is given a fee in lamports that is deducted from the balance of the `Flow` account that is being executed.

No check is performed on the fact that such balance does not get decreased below the rent-exempt threshold. Such condition would let the runtime wipe the `Flow` from the blockchain state, resulting in an unwanted `Flow` deletion.

Recommendation

Perform a check that, when decreased, `Flow` account balance never goes below the rent-exempt threshold and provide unit tests.

Alleviation

`[CertiK]`: The team heeded the advice and resolved the finding in the commit 4039ee827c6018cf063bbe8e9b3b6795ac6158c1

FLO-01 | Missing Input Validation For `Flow` Fields

Category	Severity	Location	Status
Logical Issue	● Minor	snowflake-rust-v2/programs/snowflake/src/state/flow.rs: 50, 53	✓ Resolved

Description

The `create_flow` instruction lacks input validation on several `Flow` fields.

- There is no validation on `Flow::cron` to be a non-empty string if `Flow::recurring` and `Flow::has_remaining_runs()` are `true`. In fact, in that case, each time the `Flow::update_next_execution_time` is called, in `Flow::apply_flow_data` or in `execute_scheduled_multisig_flow::handler`, independently from the remaining runs, `Flow::next_execution_time` would be put to 0 and the `Flow` can not be executed anymore.
- There is no validation on the `Flow::user_utc_offset` to be a valid UTC offset. This can result in a wrong calculation of the `Flow::next_execution_time` for recurrent `Flow`s.

Recommendation

Add the input validations that prevent the pointed cases and provide unit test to document them and to ensure that the behavior do not break in subsequent iteration of the program development.

Alleviation

`[CertiK]`: The team heeded the advice and resolved the finding in the commit 4039ee827c6018cf063bbe8e9b3b6795ac6158c1

FLO-02 | Inconsistency On `Flow::recurring` When `RECURRING_FOREVER` Is Set

Category	Severity	Location	Status
Data Flow	● Minor	snowflake-rust-v2/programs/snowflake/src/state/flow.rs: 104~116	✓ Resolved

Description

`validate_flow_data` has check considering `remaining_runs==RECURRING_FOREVER`, but does not assert that `recurring==true`.

Moreover, the condition of `remaining_runs==RECURRING_FOREVER` and `recurring==false` in `apply_flow_data` overwrites `remaining_runs==RECURRING_FOREVER` to `remaining_runs=1`.

Recommendation

`validate_flow_data` should check if `recurring==true` is implied when `remaining_runs==RECURRING_FOREVER`.

Alleviation

`[CertiK]`: The team heeded the advice and resolved the finding in the commit 4039ee827c6018cf063bbe8e9b3b6795ac6158c1

FLO-03 | Clarification On `Flow::schedule_end_date`

Category	Severity	Location	Status
Data Flow	● Informational	snowflake-rust-v2/programs/snowflake/src/state/flow.rs: 22	ⓘ Acknowledged

Description

The `Flow::schedule_end_date` field seems to indicate an end date for a recurring schedule, overlapping with the `Flow::expiry_date` semantic.

Moreover `Flow::schedule_end_date` is only set at `Flow` creation time and never used by the on-chain logic.

Recommendation

Clarification should indicate if the pointed field refers to any on-chain logic.

Alleviation

`[SnowFlake]`: Issue acknowledged. I won't make any changes for the current version.

Flow::schedule_end_date is reserved for future use.

INS-01 | Usage Of Magic Numbers

Category	Severity	Location	Status
Coding Style	● Informational	snowflake-rust-v2/programs/snowflake/src/instructions/create_flow.rs: 22, 27, 32; snowflake-rust-v2/programs/snowflake/src/instructions/update_safe.rs: 29, 43	ⓘ Acknowledged

Description

Pointed location use literals values inside the code logic. This makes difficult having them up-to-date during the software development and the maintenance process, so, it can lead to bugs.

Recommendation

Define constants for those values in order to document and track them during software development and maintenance.

SAF-01 | Incorrect Space Reservation For `Safe::extra`

Category	Severity	Location	Status
Logical Issue	● Minor	snowflake-rust-v2/programs/snowflake/src/state/safe.rs: 20~25	✓ Resolved

Description

Space calculation for the `Safe` struct is performed in the `Safe::space` method. Considering the the `String` data type is dynamically sized, the space for the `Safe::extra` field is accounted as the Rust memory frame for dynamically sized types, resulting in reserving, on-chain, an incorrect amount of space, a 24 constant amount of bytes (in the case in which `std::mem::size_of::<usize>() == 8`), instead of relying on the `String` length.

This can lead to the failure of the `create_safe` instruction or the impossibility to add new `Pubkey` to the `Safe::owners` list through the `add_owner` instruction since space reserved for the `Safe::owners` vector could be occupied by `Safe::extra` data.

Recommendation

Account for the `Safe::extra` required (or maximum allowed) space in the `Safe::space` method.

Alleviation

`[CertiK]`: The team heeded the advice and resolved the finding in the commit 4039ee827c6018cf063bbe8e9b3b6795ac6158c1

SAF-03 | Misleading `signer_nonce` Field Name

Category	Severity	Location	Status
Coding Style	● Informational	snowflake-rust-v2/programs/snowflake/src/state/safe.rs: 11	✓ Resolved

Description

When dealing with signatures, accounts and multi-signature wallets, the term "nonce" is used to refer to an increment-only counter that uniquely assign a sequence number to the issued transactions/signatures.

In this case, the `Safe::signer_nonce` field, instead, represents the (constant) `bump` for the `safe_signer` PDA. Since the `Safe` implements a multi-signature mechanism to manage the `Flow` lifecycle, then the field name can be misleading.

Recommendation

Change the field name to reflect its usage in the program (e.g. `signer_bump`, ...).

Alleviation

`[CertiK]`: The team heeded the advice and resolved the finding in the commit 4039ee827c6018cf063bbe8e9b3b6795ac6158c1

STA-01 | Unused `Flow` And `Safe` Fields

Category	Severity	Location	Status
Data Flow	● Informational	snowflake-rust-v2/programs/snowflake/src/state/flow.rs: 22, 23, 24, 27, 28, 31, 32, 33, 34, 36, 38, 39; snowflake-rust-v2/programs/snowflake/src/state/safe.rs: 13	ⓘ Acknowledged

Description

Fields at the pointed location are either set but never used or not set at all.

Recommendation

Code should document with comments the rationale of unused fields, e.g. future usage, storage for off-chain logic, ...

This increases the traceability of their usage across iterations of the program development and lows the probability of introducing bugs on those fields.

Alleviation

`[SnowFlake]`: Issue acknowledged. I won't make any changes for the current version.

Those fields are reserved for future use.

STT-01 | Unused `ProposalStateType::Failed` State

Category	Severity	Location	Status
Logical Issue	● Minor	snowflake-rust-v2/programs/snowflake/src/state/static_config.rs: 13	✓ Resolved

Description

The pointed enum case, `ProposalStateType::Failed`, is never used.

This can be an alarm for missing logic or inconsistencies arisen during the development process.

Recommendation

We recommend checking which is the aim of the pointed enum case, and either include the missing logic, if any, documenting and testing its usage with unit tests, or remove the unused code.

Alleviation

`[CertiK]`: The team heeded the advice and resolved the finding in the commit 4039ee827c6018cf063bbe8e9b3b6795ac6158c1

Optimizations

ID	Title	Category	Severity	Status
SAF-02	Incorrect Space Calculation For `Safe`	Logical Issue, Gas Optimization	● Optimization	✓ Resolved
UPD-01	Unnecessary `owners` Vector Check	Gas Optimization	● Optimization	✓ Resolved

SAF-02 | Incorrect Space Calculation For `Safe`

Category	Severity	Location	Status
Logical Issue, Gas Optimization	● Optimization	snowflake-rust-v2/programs/snowflake/src/state/safe.rs: 20~25	✓ Resolved

Description

The on-chain space requirement for the `Safe` struct is computed through the `Safe::space` method that makes use of the `std::mem::size_of` function.

In the `Safe` struct case, the usage of `std::mem::size_of` computes the usage of such a struct according to the Rust Type Layout, that is and overestimation of what is required on-chain through the Borsh serialization. This happens for two reasons:

- `std::mem::size_of` accounts for the overall memory occupation of the `Safe` struct, including Type Layout and alignment constraints, that are not necessary when serializing/deserializing with Borsh;
- the dynamic sized field `Safe::extra` and `Safe::owners` are accounted for their Rust representation, so 24 more byte for each field (a `usize` triple for each one). In the specific `Safe` case the overestimation is 5 bytes caused by the alignment of the three `u8` fields, and 48 bytes caused by the two dynamically sized fields `String` and `Vec<Pubkey>`.

Recommendation

Implement specific space calculation for the `Safe` struct instead of the generic `std::mem::size_of::<Safe>()` and provide with unit tests in order to validate the implemented calculation logic and to ensure that such logic do not break in subsequent development iterations.

Alleviation

`[CertiK]`: The team heeded the advice and resolved the finding in the commit 4039ee827c6018cf063bbe8e9b3b6795ac6158c1

UPD-01 | Unnecessary `owners` Vector Check

Category	Severity	Location	Status
Gas Optimization	● Optimization	snowflake-rust-v2/programs/snowflake/src/instructions/update_safe.rs: 28	✓ Resolved

Description

The `assert_unique_owners` function checks that the `Pubkey`s in the passed slice are unique. If `n` is the slice length, than `(n^2 - n)/2` `Pubkey` comparisons are performed.

When adding a new key to the `Safe::owners`, since the vector was already initialized with unique keys, it is enough to check the new `owner` `Pubkey` is not present in the list of owners, instead of checking again the uniqueness of all elements in the vector.

This requires `n` `Pubkey` comparisons instead of the `(n^2 - n)/2` that are performed when adding a new `Pubkey`.

Recommendation

Provide a linear check at the pointed location, instead of a quadratic one.

Alleviation

`[CertiK]`: The team heeded the advice and resolved the finding in the commit 4039ee827c6018cf063bbe8e9b3b6795ac6158c1

Appendix

Finding Categories

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.

Data Flow

Data Flow findings describe faults in the way data is handled at rest and in memory, such as the result of a struct assignment operation affecting an in-memory struct rather than an in-storage one.

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.

Checksum Calculation Method

The "Checksum" field in the "Audit Scope" section is calculated as the SHA-256 (Secure Hash Algorithm 2 with digest size of 256 bits) digest of the content of each file hosted in the listed source repository under the specified commit.

The result is hexadecimal encoded and is the same as the output of the Linux "sha256sum" command against the target file.

Disclaimer

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to you ("Customer" or the "Company") in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes, nor may copies be delivered to any other person other than the Company, without CertiK's prior written consent in each instance.

This report is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts CertiK to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. CertiK's position is that each company and individual are responsible for their own due diligence and continuous security. CertiK's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

The assessment services provided by CertiK is subject to dependencies and under continuing development. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.

ALL SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF ARE PROVIDED "AS IS" AND

“AS AVAILABLE” AND WITH ALL FAULTS AND DEFECTS WITHOUT WARRANTY OF ANY KIND. TO THE MAXIMUM EXTENT PERMITTED UNDER APPLICABLE LAW, CERTIK HEREBY DISCLAIMS ALL WARRANTIES, WHETHER EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS. WITHOUT LIMITING THE FOREGOING, CERTIK SPECIFICALLY DISCLAIMS ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT, AND ALL WARRANTIES ARISING FROM COURSE OF DEALING, USAGE, OR TRADE PRACTICE. WITHOUT LIMITING THE FOREGOING, CERTIK MAKES NO WARRANTY OF ANY KIND THAT THE SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF, WILL MEET CUSTOMER’S OR ANY OTHER PERSON’S REQUIREMENTS, ACHIEVE ANY INTENDED RESULT, BE COMPATIBLE OR WORK WITH ANY SOFTWARE, SYSTEM, OR OTHER SERVICES, OR BE SECURE, ACCURATE, COMPLETE, FREE OF HARMFUL CODE, OR ERROR-FREE. WITHOUT LIMITATION TO THE FOREGOING, CERTIK PROVIDES NO WARRANTY OR UNDERTAKING, AND MAKES NO REPRESENTATION OF ANY KIND THAT THE SERVICE WILL MEET CUSTOMER’S REQUIREMENTS, ACHIEVE ANY INTENDED RESULTS, BE COMPATIBLE OR WORK WITH ANY OTHER SOFTWARE, APPLICATIONS, SYSTEMS OR SERVICES, OPERATE WITHOUT INTERRUPTION, MEET ANY PERFORMANCE OR RELIABILITY STANDARDS OR BE ERROR FREE OR THAT ANY ERRORS OR DEFECTS CAN OR WILL BE CORRECTED.

WITHOUT LIMITING THE FOREGOING, NEITHER CERTIK NOR ANY OF CERTIK’S AGENTS MAKES ANY REPRESENTATION OR WARRANTY OF ANY KIND, EXPRESS OR IMPLIED AS TO THE ACCURACY, RELIABILITY, OR CURRENCY OF ANY INFORMATION OR CONTENT PROVIDED THROUGH THE SERVICE. CERTIK WILL ASSUME NO LIABILITY OR RESPONSIBILITY FOR (I) ANY ERRORS, MISTAKES, OR INACCURACIES OF CONTENT AND MATERIALS OR FOR ANY LOSS OR DAMAGE OF ANY KIND INCURRED AS A RESULT OF THE USE OF ANY CONTENT, OR (II) ANY PERSONAL INJURY OR PROPERTY DAMAGE, OF ANY NATURE WHATSOEVER, RESULTING FROM CUSTOMER’S ACCESS TO OR USE OF THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS.

ALL THIRD-PARTY MATERIALS ARE PROVIDED “AS IS” AND ANY REPRESENTATION OR WARRANTY OF OR CONCERNING ANY THIRD-PARTY MATERIALS IS STRICTLY BETWEEN CUSTOMER AND THE THIRD-PARTY OWNER OR DISTRIBUTOR OF THE THIRD-PARTY MATERIALS.

THE SERVICES, ASSESSMENT REPORT, AND ANY OTHER MATERIALS HEREUNDER ARE SOLELY PROVIDED TO CUSTOMER AND MAY NOT BE RELIED ON BY ANY OTHER PERSON OR FOR ANY PURPOSE NOT SPECIFICALLY IDENTIFIED IN THIS AGREEMENT, NOR MAY COPIES BE DELIVERED TO, ANY OTHER PERSON WITHOUT CERTIK’S PRIOR WRITTEN CONSENT IN EACH INSTANCE.

NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS.

THE REPRESENTATIONS AND WARRANTIES OF CERTIK CONTAINED IN THIS AGREEMENT ARE SOLELY FOR THE BENEFIT OF CUSTOMER. ACCORDINGLY, NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH REPRESENTATIONS AND WARRANTIES AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH REPRESENTATIONS OR WARRANTIES OR ANY MATTER SUBJECT TO OR RESULTING IN INDEMNIFICATION UNDER THIS AGREEMENT OR OTHERWISE.

FOR AVOIDANCE OF DOUBT, THE SERVICES, INCLUDING ANY ASSOCIATED ASSESSMENT REPORTS OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

About

Founded in 2017 by leading academics in the field of Computer Science from both Yale and Columbia University, CertiK is a leading blockchain security company that serves to verify the security and correctness of smart contracts and blockchain-based protocols. Through the utilization of our world-class technical expertise, alongside our proprietary, innovative tech, we’re able to support the success of our clients with best-in-class security, all whilst realizing our overarching vision; provable trust for all throughout all facets of blockchain.