ASC Security Guidelines

Best practices when building smart contracts on Algorand

September 18, 2022 Joe Polny

Algorand™ FOUNDATION

Disclaimer

The content provided here is not an exhaustive list of security considerations. Before deploying any application in production, it is strongly advised to get the contract audited by one more third-party security companies.

Applications vs Smart Signature

- Smart Signatures
    - Require thorough input validation
    - Delegation puts users account and funds at risk
    - End-users must have smart contract logic
- Applications
    - Can hardcode exact transaction that should take place
    - Limited input validation necessary
    - End-users just need application ID

Code Example: Smart Signature

```
return Seq(
  Assert(Txn.Amount() == Int(1)),
  Assert(Txn.TypeEnum() == TxnType.Payment),
  Assert(Txn.Receiver() == intended_receiver)
)
```

- This contract can get drained via CloseRemainderTo
- This contract can lose authorization via RekeyTo
- This contract can get drained via fees

Code Example: Application

```
return Seq(
  InnerTxnBuilder.Begin(),
  InnerTxnBuilder.SetFields({
      TxnField.type_enum: TxnType.AssetTransfer,
      TxnField.receiver: intended_receiver,
      TxnField.amount: Int(1)
  }),
  InnerTxnBuilder.Submit(),
)
```

- Contract account can't be closed
- Can't be rekeyed
- Fee will always be minimum fee required

Use Applications!

- Applications have less security considerations
- Applications can access/manipulate chain state
- Smart signatures should only be used if absolutely necessary
    - For example, offloading expensive opcodes

App Guidelines: Fees

- Fees by default will be minimum required
    - Congestion fees could be exploited to drain contract
- Setting fee to 0 means the end-user covers the fee
    - Protects contract balance from high fees
    - Makes cost more transparent to end-user
    - Makes calculating contract balance/amounts easier

App Guidelines: Updating

- UpdateApplication OnComplete should always be handled
- Pros of updatable smart contracts
  - Can add features after creation
  - Can fix bugs after creation
- Cons of updatable smart contracts
  - Can introduce bugs in updates
  - Generally means some sense of trust is involved

App Guidelines: Local State

- Local state can be cleared at any time!
  - ClearState OnComplete deletes local state, even if logic fails
- Should not be used for crtitcal information or accounting
- It should be made clear to end-users what clearing state does
- There is no trustless way of recovering data

App Guidelines: Randomness

- There is no way to generate secure random numbers *on chain*
- VRF oracles can, however, be used for trustless randomness
  - `vrf_verify` opcode can be used to verify true randomness

App Guidelines: Secrets

- It should be assumed that nothing can be kept secret on chain
- Never use something like password authentication for an application
  - Node runners can "intercept" transactions to see password and use it themselves
  - Provided hashing opcodes are not intended for passwords
- Oracle networks using privacy-preserving tech such as Intel SGX also have vulernabilites

Auditing Options

Disclaimer: Being on this list does **NOT** signify endorsement. Due diligence is required.

- Ceritik
- Kudelski
- Coinspect
- VantagePoint
- Halborn
- Runtime Verification
- UlamLabs
- ImmuneFi (bug bounty platform)