(meta-)data analysis

1. Analysing output

Differences In Output Messages / Callbacks

Secret Contracts can output messages to be executed immediately following the current execution, in the same transaction as the current execution. Secret Contracts have decryptable outputs only by the contract and the transaction sender.

Very similar to previous cases, if contract output messages are different, or contain different structures, an attacker might be able to identify information about contract execution.

Let's see an example for a contract with 2handle functions:

\`\`\`

Copy

[derive(Serialize,Deserialize,Clone,Debug,PartialEq,JsonSchema)]

[serde(rename_all="snake_case")]

pubenumHandleMsg{ Send{ amount:u8, to:HumanAddr}, Tsfr{ amount:u8, to:HumanAddr}, }

pubfnhandle(deps:&mutExtern, env:Env, msg:HandleMsg,)->HandleResult{ matchmsg { HandleMsg::Send{ amount, to }=>Ok(HandleResponse{ messages:vec![CosmosMsg::Bank(BankMsg::Send{ from_address:deps.api.human_address(&env.contract.address).unwrap(), to_address:to, amount:vec![Coin{ denom:"uscrt".into(), amount:Uint128(amount.into()), }], })], log:vec![], data:None, }), HandleMsg::Tsfr{ amount, to }=>Ok(HandleResponse{ messages:vec![CosmosMsg::Staking(StakingMsg::Delegate{ validator:to, amount:Coin{ denom:"uscrt".into(), amount:Uint128(amount.into()), }, })], log:vec![], data:None, }), } }

\`\`\`

Those outputs are plain text as they are forwarded to the Secret Network for processing. By looking at these two outputs, an attacker will know which function was called based on the type of messages -BankMsg::Send vs.StakingMsg::Delegate .

Some messages are partially encrypted, likeWasm::Instantiate andWasm::Execute , but only themsg field is encrypted, so differences incontract_addr ,callback_code_hash ,send can reveal unintended data, as well as the size ofmsg which is encrypted but can reveal data the same way as previous examples.

\`\`\`

Copy

[derive(Serialize,Deserialize,Clone,Debug,PartialEq,JsonSchema)]

[serde(rename_all="snake_case")]

pubenumHandleMsg{ Send{ amount:u8}, Tsfr{ amount:u8}, }

pubfnhandle(_deps:&mutExtern, _env:Env, msg:HandleMsg,)->HandleResult{ matchmsg { HandleMsg::Send{ amount }=>Ok(HandleResponse{ messages:vec![CosmosMsg::Wasm(WasmMsg::Execute{ /*plaintext->*/contract_addr:"secret108j9v845gxdtfeu95qgg42ch4rwlj6vlkkaety".into(), /*plaintext->*/callback_code_hash: "cd372fb85148700fa88095e3492d3f9f5beb43e555e5ff26d95f5a6adc36f8e6".into(), /*encrypted->*/msg:Binary(format!(r#"{{\"aaa\":{}}}"#, amount) .to_string() .as_bytes() .to_vec(),), /*plaintext->*/send:Vec::default(), })], log:vec![], data:None, }), HandleMsg::Tsfr{ amount }=>Ok(HandleResponse{ messages:vec![CosmosMsg::Wasm(WasmMsg::Execute{ /*plaintext->*/contract_addr:"secret1suct80ctmt6m9kqmyafjl7ysyenavkmm0z9ca8".into(), /*plaintext->*/callback_code_hash: "e67e72111b363d80c8124d28193926000980e1211c7986cacbd26aacc5528d48".into(), /*encrypted->*/msg:Binary(format!(r#"{{\"bbb\":{}}}"#, amount) .to_string() .as_bytes() .to_vec(),), /*plaintext->*/send:Vec::default(), })], log:vec![], data:None, }), } }

\`\`\`

The same analysis applies for output differences like:

1. ## of output messages
2.

For example, a dev writes a contract with 2 functions, the first one always outputs 3 events and the second one always outputs 4 events. By counting the number of output events an attacker can know which function was invoked. This also applies with deposits, callbacks and transfers.

1. Type of message at output
2.

If a contract returns aStdError , the output looks like this:

```

Copy { "Error":"" }

```

Otherwise the output looks like this:

```

Copy { "Ok":"" }

```

Therefore similar to previous examples, an attacker might guess what happened in an execution. E.g. If a contract has only asend function, if an error was returned an attacker can know that themsg.sender tried to send funds to someone unknown and thesend didn't execute.

1. Other actions that happen after output
2.

3. "Push notifications" for GUIs with SecretJS

4. To make the tx searchable on-chain
5.

6. Logs

7.

8. Number of logs

9. Size of logs
10. Ordering of logs (short,long vs. long,short)
11.

1. Analysing Input and computation during runtime

Deanonymizing With Ciphertext Byte Count

No encryption padding, so a value of e.g. "yes" or "no" can be deanonymized by its byte count. Padding the inputs and variables during execution can limit this attack vector. By padding values the execution will be more constant-time and simultaneously you wont be able to track a specific sized value across the computation process inferring what is happening to it and where it is stored.

Padding inputs in many cases should mitigate practical attacks using data size analysis on computation.

Data Leakage Attacks By Analyzing Metadata Of Contract Usage

Depending on the contract's implementation, an attacker might be able to de-anonymization information about the contract and its clients. Contract developers must consider all the following scenarios and more, and implement mitigations in case some of these attack vectors can compromise privacy aspects of their application.

In all the following scenarios, assume that attackers have local control of a full node. They cannot break into SGX, but they can tightly monitor and debug every other aspect of the node, including trying to feed old transactions directly to the contract inside SGX (replay). Also, though it's encrypted, they can also monitor memory (size), CPU (load) and disk usage (read/write timings and sizes) of the SGX chip.

For encryption, the Secret Network is usingAES-SIV , which does not pad the ciphertext. This means it leaks information about the plain text data, specifically about its size, though in most cases it's more secure than other padded encryption schemes. Read more about the encryption specsHERE .

Most of the below examples talk about an attacker revealing which function was executed on the contract, but this is not the only type of data leakage attackers may target.

Secret Contract developers must analyze the privacy model of their contract - What kind of information must remain private and what kind of information, if revealed, won't affect the operation of the contract and its users.Analyze what it is that you need to keep private and structure your Secret Contract's boundaries to protect that.

This means developers have to be aware about how they structure their contract and what that might reveal like:

- Ordering of messages (E.g.Bank
- and thenStaking
- vs.Staking
- andBank
-)
- Size of the encryptedmsg
- field
- Number of messages (E.g. 3Execute

- vs. 2Execute
-)
-

Again, be creative if that's affecting your secrets.

1. Analysing Storage Access

For more information on the exact workings of this attack vector please read the Section"Network Level - Storage access"

The short summary is that a participating node in the network can monitor which parts of storage are accessed by a contract upon execution. As Strorage is separated by Contract ID and user ID a node can infer which parties are active in a specific compute transaction revealing potential transfer recipients or other metadata of the Transaction.

To prevent this attack in other Contract situations one can implement different storage access within their Contract like UTXO, not calling all items in the same contract, or ORAM.

Last updated7 months ago On this page *1. Analysing output * Differences In Output Messages / Callbacks * 2. Analysing Input and computation during runtime * Deanonymizing With Ciphertext Byte Count * Data Leakage Attacks By Analyzing Metadata Of Contract Usage * 3. Analysing Storage Access