Smart Contract Security Audit Report

[2021]

The SlowMist Security Team received the ICONation team's application for smart contract security audit of the ICONSafe on Mar. 12, 2021. The following are the details and results of this smart contract security audit.

Project description

ICONSafe is a multisig wallet with an advanced user management and built-in features for dApps running on ICON. It is mostly useful for teams willing to share mutual funds without having to risk sharing a same private key between members.

ICONSafe is able to send and receive any type of transactions or contract calls. It is also able to track any ICX and IRC2 tokens balance over time. All outgoing transactions require confirmations from the wallet owners based on a vote before being executed.

An outgoing transaction may contain multiple sub-transactions which are executed at the same time, so it is possible to create complex operations suiting for all type of situations.

The ICONSafe project is a smart contract that any developer may deploy and integrate in another smart contract project. Please note that for pratical use, we also developed a GUI heavily based on Gnosis Safe GUI in order to communicate with the ICONSafe contract.

Audit Scope

https://github.com/iconation/ICONSafe-SCORE/tree/master/contracts/transaction_manager

commit: afc628e7521475312859bacedc74dff1e1ec5ec2

https://github.com/iconation/ICONSafe-SCORE/tree/master/contracts/wallet_owners_manager

commit: afc628e7521475312859bacedc74dff1e1ec5ec2

Audit Result

Audit Result	Passed
Audit Number	0X002103190001
Audit Date	Mar. 19, 2021
Audit Team	SlowMist Security Team

Findings

NO.	Level	Title	Audit Type	Status
1	Suggestion	Do not need to check requirements while replace_wallet_owner	Resource Control	Fixed
2	Suggestion	_call_transaction seems an internal function but external	Others	Fixed
3	Medium	One participator can let other participators to confirm a transaction	Call Control	Fixed

[Suggestion] [N01] Do not need to check requirements while replace_wallet_owner

Content

contracts/wallet_owners_manager/wallet_owners_manager.py 202-231 lines,

```
@only_iconsafe
@external
```

```
def replace_wallet_owner(self, old_wallet_owner_uid: int, new_address: Address, new_name: str) -> None:
    # Access
    #   Only ICONSafe Proxy contract
    # Description
    #   Replace an existing owner to the multisig wallet
    # Parameters
    #   - wallet_owner_uid: the wallet owner UID
    #   - new_address : the new address for the owner
    #   - new_name : the new name for the owner
    # Returns
    #   - Same than WalletOwnersManager.remove_wallet_owner
    #   - Same than WalletOwnersManager.add_wallet_owner
    # Throws
    #   - Same than WalletOwnersManager.remove_wallet_owner
    #   - Same than WalletOwnersManager.add_wallet_owner

    # --- Checks ---
    self.__check_requirements(len(self._wallet_owners), self._wallet_owners_required.get())
    old_wallet_owner = WalletOwner(old_wallet_owner_uid, self.db)

    # Check if only the name is changed
    if not old_wallet_owner.same_address(new_address):
        self.__check_address_doesnt_exist(new_address)

    # --- OK from here ---
    new_wallet_owner_uid = WalletOwnerFactory.create(self.db, new_address, new_name)
    self.__remove_wallet_owner(old_wallet_owner_uid)
    self.__add_wallet_owner(new_address, new_wallet_owner_uid)
```

The __check_requirements function only check owner numbers, replace_wallet_owner doesn't change owner numbers, so don't need to check it.

Solution

Remove self.__check_requirements.

Status

Fixed on
https://github.com/iconation/ICONSafe-SCORE/commit/de5a0977feb2a8baad9917b38f97af411027fd26

[Suggestion] [N02] _call_transaction seems an internal function but external

Content

contracts/transaction_manager/transaction_manager.py 246-277 lines,

```
@external
@only_transaction_manager
def _call_transaction(self, transaction_uid: int) -> None:
    # Access
    #   - Only Transaction Manager calling itself
    # Description
    #   - Try to execute a given transaction
    # Parameters
    #   - transaction_uid : the transaction UID to execute
    # Returns
    #   - Nothing
    # Throws
    #   - AddressNotInRegistrar
    #   - SenderNotTransactionManagerException

    transaction = OutgoingTransaction(transaction_uid, self.db)
    # Get the execution time, even if the subtx fails
    transaction._executed_timestamp.set(self.now())

    # Handle all sub transactions
    for sub_transaction_uid in transaction._sub_transactions:
        sub_transaction = SubOutgoingTransaction(sub_transaction_uid, self.db)

        method_name = sub_transaction._method_name.get() or None
        params = sub_transaction.convert_params()
        destination = sub_transaction._destination.get()
        amount = sub_transaction._amount.get()

        if destination.is_contract and method_name != None:
            self.call(addr_to=destination, func_name=method_name, kw_dict=params, amount=amount)
        else:
            self.icx.transfer(destination, amount)
```

when _ as a function name prefix, it seems like an internal function

Solution

Rename function name _call_transaction to call_transaction.

Status

Fixed on
https://github.com/iconation/ICONSafe-SCORE/commit/196bca69e0285a8dd01e76c9fbebdbd949dc75b9

[Medium] [N03] One participator can let other participators to confirm a transaction

Content

contracts/transaction_manager/transaction_manager.py 355-284 lines

```
@external
@only_iconsafe
def confirm_transaction(self, transaction_uid: int) -> None:
    # Access
    #   - Only ICONSafe Proxy contract
    # Description
    #   - Confirm the given transaction for the tx sender.
    # Parameters
    #   - transaction_uid : the transaction uid to confirm
    # Returns
    #   - TransactionConfirmed
    #   - TransactionExecutionSuccess or TransactionExecutionFailure
    # Throws
    #   - InvalidState (wrong transaction state, it needs to be pending)
    #   - OutgoingTransactionAlreadyParticipated (the tx sender already participated)
    #   - Same than TransactionManager.update_all_balances

    transaction = OutgoingTransaction(transaction_uid, self.db)
    wallet_owner_uid = self.wallet_owners_manager.get_wallet_owner_uid(self.tx.origin)

    # --- Checks ---
    transaction._type.check(TransactionType.OUTGOING)
    transaction._state.check(OutgoingTransactionState.WAITING)
    transaction.check_hasnt_participated(wallet_owner_uid)
    # --- OK from here ---
    transaction._confirmations.add(wallet_owner_uid)
    self.TransactionConfirmed(transaction_uid, wallet_owner_uid)

    self.__try_execute_transaction(transaction_uid)
```

There is the main code, use self.tx.origin as a participator to confirm the transaction, not self.msg.sender, which is the origin owner who calls the function;

contracts/iconsafe/iconsafe.py 203-207 lines,

```
@external
@catch_exception
@only_multisig_owner
def confirm_transaction(self, transaction_uid: int) -> None:
    self.transaction_manager.confirm_transaction(transaction_uid)
```

participators should call this function to confirm transaction, the check function only_multisig_owner is:

contracts/interfaces/wallet_owners_manager.py 90-101 lines,

```
def only_multisig_owner(func):
    if not isfunction(func):
        raise NotAFunctionError

    @wraps(func)
    def __wrapper(self: object, *args, **kwargs):
        if not self.wallet_owners_manager.is_wallet_owner(self.msg.sender):
            raise SenderNotMultisigOwnerError(self.msg.sender)

        return func(self, *args, **kwargs)

    return __wrapper
```

So the attack scenario:

1. One participant (we call attacker) supports a contract address to participate the transaction manager, and set payable fallback function to call the iconsafe.confirm_transaction;

2. The attacker let other ones (we can call victim) to transfer values to him;
3. the victim will confirm the transaction which he didn't want to do.

Solution

Check that participants are not contract addresses.

Status

Fixed on

https://github.com/iconation/ICONSafe-SCORE/commit/39669107623d775bcf24e414890ba52876cd669b